MySQL

# What is a database

A database is structured collection of data. There are different types of databases:

* **An online store database** that stores products, customer details and orders
* **A database for a web forum** that stores members, forums, topics and posts
* **A database for a blog system**, such as WordPress, that stores users, blog posts, categories, tags, and comments

The software that manages databases is known as a **database management system (DBMS)**. A database model is a way to organize data in a database. It’s used for data storage, data retrieval, data manipulation and authentication & authorization. The most popular is MySQL, this one is known as a **relational database management system (RDBMS)**. In summary a database is a collection of related data, managed by a DMBS such as MySQL. A web script communicate with the DBMS in order to insert, update, delete and retrieve data in the database.

# Why use a database

If you have stored data in text files such as comma-separated value (CSV) files. This way is fine for simple data structures and small amounts of data. This way isn’t convenient for the storage of greater data. Searching and updating in these text file is slow and difficult. Relational databases are designed to take care of all these problems. It has many advantages:

* They’re fast. DBs use indexes, or keys, to find records of data.
* They’re reliable. A DBMS ensures that the data is read and written reliable, without corrupting the data.
* They let you link records together. It lets you store different types of data in groups known as tables.

# Why use MySQL

It has a few advantages compared with other systems:

* It’s open source.
* It’s widely available.
* It’s easy to use.
* It works well with PHP.

# Introduction

## Install MySQL on Linux

Version 2,5,1

* Sudo dnf install htpps://dev.mysql.com/get/mysq157-community-release-fc26-10.noarch.rpm
* Sudo yum install mysql-workbench-community.x
* Sudo service mysqld status
* Sudo yum install mysql-community-server
* Sudo service mysqld status
* Sudo service mysqld start

Passwoord necessary:

* Sudo grep ‘temporary password’ /var/log/mysqld.log
* Mysql –uroot –p (Add temporary psswd who was given by last command )
  + Zorgen dat je niet telkens opnieuw je psswd moet instellen
* Mysql
* >SET GLOBAL validate\_password\_policy = LOW;
* >ALTER USER ‘root’at’localhost’ IDENTIFIED BY ‘howest2017’;
* Exit mysql

Creating of a config.file, Best in your home directory

* Cat /etc/my.cnf 🡺 if bin-adres isn’t present
* Add via nano or edit the bin adres
* sudo nano /etc/my.cnf

Add to bind-adress=127.0.0.1

* Socket=/var/lib/mysql/mysql.cock
* Datadir=/var/lib/mysql

Create a database user

* Mysql –uroot –p 🡺 enter psswd
* Mysql
* >CREATE USER ‘newuser’at’localhost’ IDENTIFIED BY ‘password’; (moet verschillend zijn van root psswd 🡺 zelf kiezen)
* Log uit als roor en log in als nieuwe gebruiker
* Myslq –unewuser -p

In home a new confi file aanmaken

* Sudo toucht .my.cnf
* Gedit .my.conf
  + [client]
  + User = ‘newuser’
  + Password= ‘password’ (ingeven password)
* Mysql
* Quit
* Sudo chmod 600 .my.cnf
* Test: $mysql (inloggen zonder username and psswd)

Aanmaken van een db:

* quit
* $ Mysql –uroot -p
* Mysql > create database biodb;
* Mysql >GRANT all on biodb.\* tbl TO user@localhost; (or \*.\*)
* Mysql > exit
* $ mysql
* Mysql > show database 🡺 the db should be visible
* Mysql > exit
* Download the file bioinf\_testdb.sql
* $ Mysql (biodb) < bioinf\_testdb
* $ mysql
* Mysql > show database;
* Mysql > use bioinf\_testdb

Connect to database service as a normal user:

* Nano .my.cnf
  + [client]
  + User=‘guest’
  + Password=‘howest2017’

# Open MySQL in windows 10

To open a terminal window in windows go to start 🡪 all programs 🡪 MySQL 🡪 MySQL 5.7 Command Line Client. As password = root.

# Commands

## Creating a database

Mysql> select now(); |get current date and time and display’s it.

Mysql>SOURCE namedb; |load DB in mysql, add direction of database

## Creating a table

Mysql>CREATE DATABASE name\_of\_new\_database; |create a simple new database

Mysql>USE name\_of\_new\_database; |tells to MySQL to switch to the DB

Mysql>DROP TABLE IF EXIST name\_of\_new\_database; |deletes any previous table from DB

Mysql>CREATE TABLE name\_of\_new\_database ; |creates a new table in the DB

Mysql>id int unsigned NOT NULL auto\_increment; |type of field that assigns a unique numeric |ID to |each record in table. Int (integer), |auto\_increment will automatically assign a |new |unique value to record’s id field.

Mysql>title varchar(255) NOT NULL; |varchar: hold a text string

Mysql>price decimal(10,2) NOT NULL; |decimal: 10-digit decimal number, with 2 |0,01

Mysql>PRIMARY KEY (id); |uniquely identifies records in table

## Adding records and deleting

Mysql>INSERT INTO db ( , , )VALUES(“”,””,””); |Insert: Add record to table; INTO: add to DB;

|values: the values of the records

Mysql>DELETE FROM table WHERE …=”” |delete a record from a table

Mysql>ALTER TABLE tablenaam ADD … AFTER … |add a column

Mysql>ALTER TABLE tablenaam DROP …; |delete a column

Mysql>UPDATE table; |update the data

## Retrieving records

Mysql>SELECT fieldNames FROM tableNAME; |retrieve one or more records from a table

Mysql>SELECT \* FROM table; |retrieves all fields (\*) from the table

Mysql>SELECT \*FROM table WHERE id=2; |where: narrow down the selection

Mysql>SELECT \*FROM table WHERE … < …AND > …; |less and greater than and the Boolean AND, |retrieve a range of records

Mysql>SELECT columns FROM tbl; |select a specific column from the table

Mysql>SELECT col1, col2 FROM tbl; |Select more columns from the table

Mysql>SELECT col1 FROM tbl ORDER BY … ASC/DESC; |order columns ion ascending or descending |order.

Mysql>SELECT CONCAT(col1,” “, col2) FROM tbl; |concatenate two columns

Mysql>SELECT col AS … |as: alias, this can be used in the order by

Mysql>SELECT … aggregatefunctions FROM src |Sort dat into groups for aggregation

GROUP BY … |purposes

## View items

Mysql>DESCRIBE table; |show all info on all columns of table

Mysql>SHOW databases; |shows all the MySQL databases

Mysql>SHOW tables; |show the tables of the DB

## Dealing with NULL-values

Mysql>SELECT … WHERE col IS NULL; |testing for null-ness, give as col is null

Mysql>SELECT … WHERE col IS NOT NULL. |give as it is not null

Mysql>SELECT ifnull (col,value) |col if col is not null, value if col is null

# Relational databases with MySQL

## Column types

* INT() 🡺 integer, number
* FLOAT() 🡺 numbers with decimal point
* DATE 🡺 YYYY-MM-DD, datetime
* VARCHAR() & CHAR() 🡺 string with a certain number of characters
* TEXT & BLOB 🡺 Used for texts that are not queried often or do not have to be searchable, BLOB for binary data (images,...)
* ENUM 🡺 LIST of permitted values (vb: set of colours: ‘red’,’green’)

## Functions

|  |  |  |
| --- | --- | --- |
| **Calculated rows - numbers** | **Calculated rows - dates** | **Calculated rows - strings** |
| sqrt(x), power(x, y), …  exp(x), ln(x), …  sin(x), cos(x)  round(x), ceil(x), floor(x), …  rand(), rand(x) | currentdate(), now() year(d), month(d), week(d) dayofmonth(d), dayofweek(d)  hour(d), minute(d), second(d) | length(s) concat(s1, …) upper(s), lower(s)  trim(s), ltrim(s), rtrim(s) substr(s, …)  reverse(s), truncate(s) |
| **Aggregation** |  |  |
| count(*col*), count(*\**), count(distinct *col*), sum(*col*)  min(*col*), max(*col*), avg(*col*), stddev(*col*), variance(*col*) |  |  |

## Data control language

DCL statements manage database rights with the commands GRANT and REVOKE.

## Filtering:

### BOOLEANS

* not x 🡺 Evaluates TRUE if x is FALSE
* x and y 🡺 Evaluates TRUE if both x and y are TRUE
* x or y 🡺Evaluates TRUE if x or y is TRUE, or both
* x xor y (exclusive or) 🡺Evaluates TRUE if x or y is TRUE, but not both

### Duplicates

* SELECT DISTINCT (cols) FROM … |eliminate duplicate rows

### Limiting output

* SELECT … LIMIT n [OFFSET r] |limit number of rows in a result set offset r:

|the first row are skipped

### Filter on based on result of aggregate functions

* SELECT … aggregatefunctions FROM … GROUP |filtering based on having defined conditions

BY … HAVING cond2

## SQL: structured Query Language

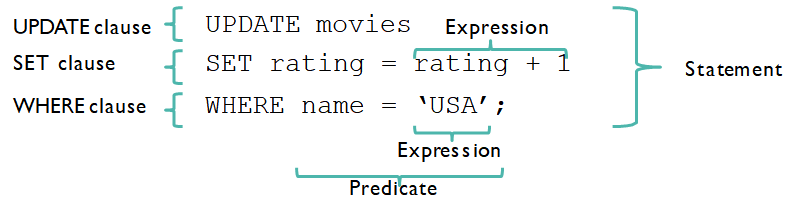


Figure 1: The rating score of the movies with the name USA will be increase with 1.

## Execution order

1. Input columns are determined
   * JOIN clause
2. WHERE – input columns are filtered
3. GROUP BY – sorting & grouping of filtered input
4. Aggregation functions are calculated
5. HAVING – aggregation functions are filtered
6. ORDER BY – output is sorted
7. LIMIT/OFFSET – output is chopped

## Joins

Databases are very sensitive for mistakes. To prevent this, tables are split into different tables. Therefore foreign keys are used. Within a table, there is a new trajectory made. In this way data types are linked between different tables. To link to tables together a cross linking table could be made.

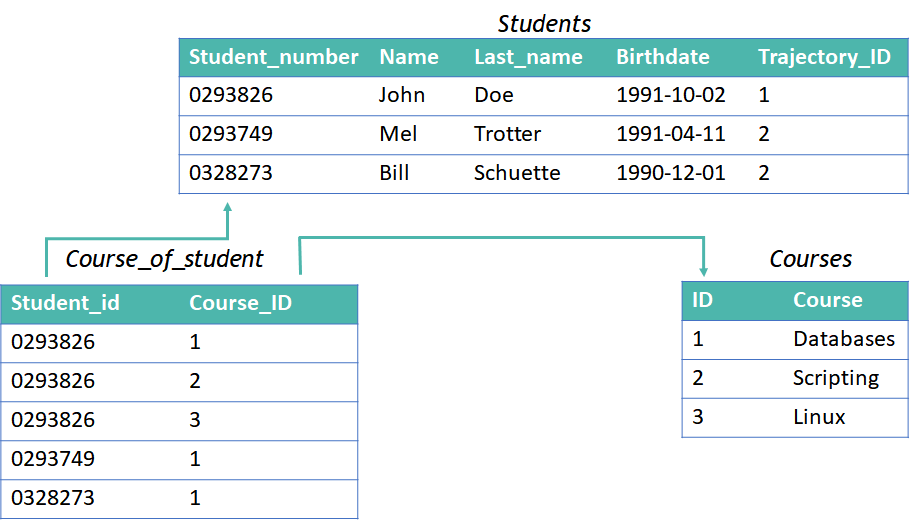


Figure 2: Cross linking table

The creation of a databases is based on a database model called Entity Relationship Diagram (ERD). For example 1 student could follow more courses. This is called one-to-many relationship. Another is many-to-many relationship, 1 student could follow more courses and one courses has more students.

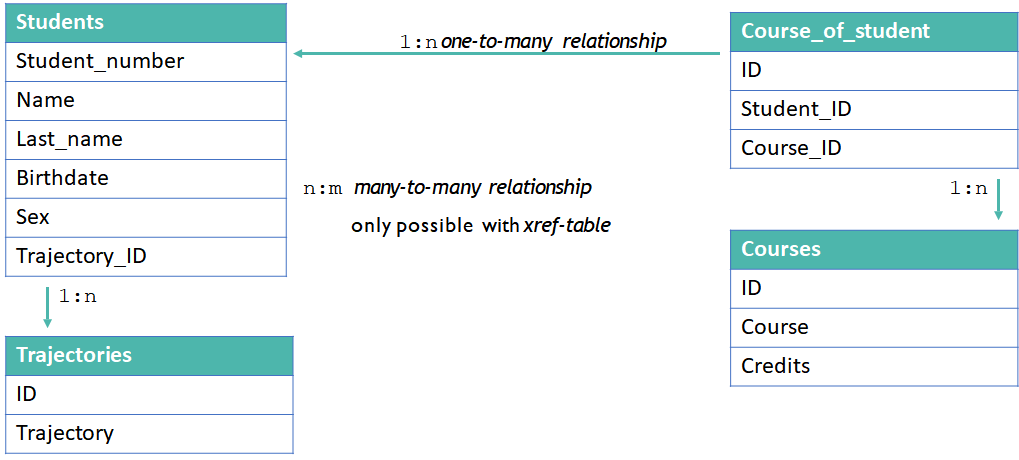


Figure 3: different relationships

### Retrieve linked rows from different tables

* SELECT \* FROM student JOIN trajectories ON student.trajectory\_id=trajectories.id;
  + The trajectory from the student table will be linked with the trajectory table’s ID.

### Cartesian product

An cartesian product is that multiple tables in a query a database server generate all possible combinations. Filtering cartesian product:

* SELECT \* FROM modorg, class WHERE modorg.class\_id = class.id;
* SELECT \* FROM modorg [INNER] JOIN class ON modorg.class\_id = class.id;

IF we want to select something from column a and b, use alias for data source.

* SELECT a.col, b.col FROM src1 [as] a, src2 [as] b WHERE …

IF we want to take every class, name, datum from modorg table and group them according class\_id and use them further as alias.

* SELECT class\_id, min(draft) as dr FROM modorg GROUP BY class\_id;

### Data source alias

The part between () is save an alias and will be linked to the class\_id. In this way the name can be selected.

* SELECT name, dr FROM (SELECT class\_id, min(draft) as dr FROM modorg GROUP BY class\_id) as s, class WHERE s.class\_id=class\_id;

### 4 types

* INNER JOIN: Only rows present in both tables, only results where trajectory is not null.
* LEFT JOIN: All rows from left table, even without linked data in right table, regardless results of the right table, tables are fit together.
* RIGHT JOIN: All rows from right table, even without linked data in left table. Takes all rows of the right table together with the same rows of the left table.

## View

A view is a virtual table that is used to serve up data in an orderly fashion. Create a new view:

* CREATE VIEW *viewname* AS SELECT …

To use a view as table use:

* SELECT … FROM viewname WHERE … ORDER BY
* Create a view (genevw) from the query in the previous exercise
* Select all genes containing hemoglobin in the description and sort the result set by gene length.
* Next:
  + What is the minimum and maximum gene length?
  + What is the average gene length?
  + And the standard deviation?

### Examples creating views<http://www.mysqltutorial.org/create-sql-views-mysql.aspx>

Create a view that represents total sales per order.

* CREATE VIEW SalePerOrder AS SELECT orderNumber, SUM(quantityOrdered \* priceEach) total FROM orderDetails GROUP BY orderNumber ORDER BY total DESC;

Create a view called big sales order based on the SalesPerOrder.

* CREATE VIEW BigSalesOrder AS SELECT orderNumber, ROUND(total, 2) AS total FROM saleperorder WHERE total >600000;

Create a views with joins, the views contain the order number, customer name and total sales per order.

* CREATE VIEW customerOrders AS SELECT d.orderNumber,  customerName, SUM(quantityOrdered \* priceEach) total FROM orderDetails d INNER JOIN orders o ON o.orderNumber = d.orderNumber INNER JOIN customers c ON c.customerNumber = c.customerNumber GROUP BY d.orderNumber ORDER BY total DESC;

## Database backup

To create a backup, first a complete database is dump into a text file using the command mysqldump db > db.sql. The database can be restored using mysql db < db.sql. Bellow is an example given.

* Mysqaldump biodb > biodb.sql
* Mysaldump biod –u root –p> biodb.sql
* Gedit \*.sql
* Ll
* Drop database naamdb;

## Exercises

**Exercise: make a table**

* SHOW DATABASES;
* CREATE DATABASE student;
* Use student;
* CREATE TABLE `studentdata.student` (
  + `student\_number` INT(7) NOT NULL,
  + `name` VARCHAR(45) DEFAULT NULL,
  + `last\_name` VARCHAR(45) DEFAULT NULL,
  + `birthdate` DATE DEFAULT NULL,
  + `seks` EMUM(‘M’,’V’) DEFAULT NULL,
  + PRIMARY KEY (`student\_number`)
  + );
* INSERT INTO `sudentdata\_student` ( `student\_number`, `name`, `last\_name`, `birthdate`, `seks`)
  + VALUES(0293826, ‘John’, ‘Doe’, 19911002, ‘M’),
  + VALUES (0293749, ‘mel’, ‘trotter’, 19990411, ‘V’),
  + VALUES (0328273, ‘bill’, ‘schuette’, 199901201, ‘V’)
  + );

**Exercise: Show the names (genus & species) of all model organisms in the order of the publishing date of the draft**

* Download database
* Mysql biodb < 2.sql
* Mysql
* SHOW DATABASES;
* USE biodb;
* SHOW TABLES;
* SELECT \* FROM modorg;
* SELECT genus, species, draft FROM modorg;
* SELECT genus, species, draft FROM modorg ORDER BY draft;

**Exercise: Show the names (genus & species) of all model organisms sorted by the number of chromosomes (most chromosomes on top) and then alphabetically by name**

* SELECT \* FROM modorg;
* SELECT genus, species, nchr FROM modrog;
* SELECT \* FROM modorg ORDER BY nchr DESC, genus, species;

**Exercise: show model organism full name, genome size, rounded to 4 digits, average chromosome size and publication year. Sort by average chromosome size**

* SELECT\*FROM modorg;
* SELECT concat(genus,” “,species), round(gsize/1000,4), gsize/nchr, year(draft) FROM modorg ORDER BY gsize/nchr DESC;

**Exercise: show model organism full name as name, average chromosome size as avgsize and publication year as pubyear. Sort rows by avgsize.**

* SELECT CONCAT(genus, “ “, species) AS name, gsize/nchr AS avgsize, year(draft) as pubyear FROM modorg ORDER by avgsize DESC;
* SELECT CONCAT(geuns,” “,species) AS name, ROUND(gsize/nchr,4) AS avgsize, year(draft) AS pubyear FROM modorg ORDER BY avgsize DESC;

**Exercise: Select all mammals with genomes published after 2005**

* *SELECT \* FROM modorg WHERE class=‘Mammals’ AND year (draft) > 2005;*

**Exercise: Select all organisms that have an average chromosome size between 10 and 100 Mbp**

* *SELECT \* FROM modorg WHERE gsize/nchr BETWEEN 10 AND 100;*

**Exercise: Select all organisms whose genus starts with A, B, C, D, or E**

* SELECT \* FROM modorg WHERE genus LIKE ‘A%’ OR genus LIKE ‘B%’ OR genus LIKE ‘C%’ OR genus LIKE ‘D%’ OR genus LIKE ‘E%’;

OR

* SELECT \* FROM modrog WHERE genus >= ‘A’ AND genus < ‘F’;

OR

* SELECT \* FROM modorg WHERE genus RLIKE ‘^[A-E]’; |aan het begin ^

**Exercise: Give an overview of all organism classes in the dataset (sorted alphabetically)**

* SELECT DISTINCT(class) FROM modorg ORDER BY class;

**Exercise: Show the organism names of the top 3 largest genome sizes**

* SELECT genus, species FROM modorg ORDER BY gsize DESC LIMIT 3;

**Exercise: What is the result and why**

1. SELECT count(\*) FROM modorg; 🡺10
2. SELECT count(nchr) FROM modorg; 🡺 9
3. SELECT count(class)FROM modorg; 🡺10
4. SELECT count(DISTINCT class) FROM modorg; 🡺 6

**Exercise: How many mammals are in the database**

* SELECT COUNT(class) FROM modorg WHERE class LIKE ‘mammals’; 🡺 4

**Exercise: How many organisms are present in the dataset for each class? Note the sort order.**

* SELECT class, count(\*) FROM modorg GROUP BY class;

**Exercise: Show the minimum and the maximum genome sizes for each class. Take only those organisms into account for which genome sizes are known. Sort the results such that the biggest maximum genome size is on top.**

* SELECT class, min(gsize) as min, max(gsize) as max FROM modorg WHERE gsize IS NOT NULL GROUP BY class ORDERD BY max DESC;

**Exercise: For each class with more than 1 organism, show the average number of chromosomes. Sort the result such that the biggest average is on top.**

* SELECT class, AVG(nrchr) as avg FROM modorg GROUP BY class HAVING COUNT(\*) > 1 ORDER BY avg DESC;

**Exercise: Find the gene named HOTAIR in the gene table (bioinf\_testdb). In addition to the gene name and the chromosomal position, return the size of the gene**

* SELECT gene\_name, chromosome, seq\_region\_start, seq\_region\_end, (seq\_region\_end – seq\_region\_start +1) FROM gene WHERE gene\_name=’HOTAIR’;

|Count always +1 extra because, the count starts from 0

**Exercise: Find the known lincRNA that is located the most distal on the p-arm of chromosome 14**

* SELECT \* FROM gene WHERE biotype=’lincRNA’ AND status=’KNOWN’ AND chromosome=’14’ ORDER BY seq\_region\_start ASC LIMIT 1;

**Exercise: Find genes related to prostate cancer (use description field)**

* SELECT gene\_name, description FROM gene WHERE description LIKE ‘%prostate%cancer%’;

|% between results in both searched separately

**Exercise: Return a list of genes located on chromosome X in alphabetical order**

* SELECT DISTINCT(gene\_name) FROM gene WHERE chromosome=’X’ ORDER BY gene\_name;

**Exercise: How many occurrences does each biotype have in the gene table?**

* SELECT COUNT(\*), biotype FROM gene GRÖUP BY biotype;

**Exercise: A mutation was found on chromosome 20, position 44002590. Which gene(s) have an overlap with this position?**

* SELECT gene\_name FROM gene WHERE chromosome=20 AND seq\_region\_start<=44002590 AND seq\_region\_end >=44002590;

Or

* SELECT gene\_name FROM gene WHERE chromosome=20 AND 44002590 BETWEEN seq\_region\_start AND seq\_region\_end;

**Exercise: Which chromosome has the second most genes?**

* SELECT COUNT(\*), chromosome FROM gene GROUP BY chromosome ORDER BY COUNT(\*) DESC LIMIT 1 OFFSET1;

**Exercise: On which chromosome strand lie the most genes?**

* SELECT COUNT(\*) seq\_region\_strand FROM gene GROUP BY seq\_region\_strand ORDER BY COUNT(\*) DESC LIMIT 1;

**Exercise: What is the most common biotype for novel transcripts?**

* SELECT COUNT(\*), biotype FROM transcript WHERE status=’NOVEL’ GROUP BY biotype ORDER BY COUNT(\*) DESC LIMIT 1;

**Exercise: For each biotype with more than 10 known transcript, give the lowest start position and the highest end position.**

* SELECT biotype, min(seq\_region\_start), max(seq\_region\_end) FROM transcript WHERE status=’KNOWN’ GROUP BY biotype HAVING COUNT(\*) > 10;

**Exercise: How many exons have a different end phase than their current phase?**

* SELECT COUNT(\*) FROM exon WHERE phase != end\_phase;

**Exercise: Return a list of the 3 biotypes with the most genes. Which biotype is a close fourth? Return a list of the number of genes per status**

* USE bioinf\_testdb;
* SELECT COUNT(\*), biotype FROM gene WHERE status=’KNOWN’ GROUP BY biotype ORDER BY COUNT(\*) DESC limit 3.

**Exercise: Return a list of the number of genes per status**

* SELECT COUNT(\*), status FROM gene GROUP BY status;

**Exercise: Combining the 2 previous results, which biotype is most known?**

* SELECT COUNT(\*), status, biotype FROM gene WHERE status=’KNOWN’ GROUP BY status, biotype ORDER BY COUNT(\*) DESC;

**Exercise: Select only those biotypes that cover at least 3% of the human genome (hint: ‘size’ in previous exercise, human genome is approx. 3 billion bp) and return this percentage**

* SELECT biotype, SUM(seq\_region\_end -seq\_region\_start +1)/300000000 AS size FROM gene GROUP BY biotype HAVING size >= 0.03 ORDER BY sizze DESC;

**Exercise: For all rows in table gene, show organism name, class name, accession, length an description of the gene**

* Exit
* Mysql -uroot -p
* CREATE database bioinf;
* GRANT ALL ON bioinf.\* TO newuser@localhost;
* Exit
* Mysql bioinf < 2.sql
* Mysql
* Show datbases;
* USE bioinf;
* SHOW TABLES;
* SELECT \*FROM gene;
* SELECT acc, len, descr FROM gene;
* SELECT acc, len, descr FROM gene JOIN modorg ON gene.mo\_id=modorg.id;
* SELECT CONCAT(genus,” ”,species) AS org\_name, acc, len, descr FROM gene JOIN modorg ON gene.mo\_id=modorg.id;
* SELECT CONCAT(genus,” ”,species) AS org\_name, acc, len, descr FROM gene JOIN modorg ON gene.mo\_id=modorg.id JOIN class ON modorg.class\_id=class.id;
* SELECT CONCAT(genus,” ”,species) AS org\_name, name, acc, len, descr FROM gene JOIN modorg ON gene.mo\_id=modorg.id JOIN class ON modorg.class\_id=class.id;

|table: gene, gelinkt met class, kijken nr ID

**Exercise: Examine the gene,transcript and exon table, how are they connected? How many transcript does the MALAT1 gene have? Return the position of the exons of transcript 237999**

* USE bioinf\_testdb;
* SELECT COUNT(\*) FROM gene JOIN transcript ON transcript.gene\_id=gene.gene\_id WHERE gene\_name=’MALAT1’;
* SELECT transcript\_id, exon.seq\_region\_start, exon.seq\_region\_end FROM exon\_transcript JOIN exon ON exon\_transcript.exon\_id=exon.exon\_id WHERE transcript\_id=237999;

OR

* SELECT transcript\_id, seq\_region\_start, seq\_region\_end FROM exon JOIN exon\_transcript ON exon.exon\_id=exon\_transcript.exon\_id WHERE transcript\_id=237999;

**Exercise: Return the transcripts of the TP53 gene. Return their exons as well.**

* SELECT transcript.\*FROM gene JOIN transcript ON transcript.gene\_id=gene.gene\_id WHERE gene\_name=’TP53’;
* SELECT transcript.\* exon.\* FROM gene JOIN transcript ON transcript.gene\_id=gene.gene\_id JOIN exon\_transcript ON transcript.transcript\_id=exon\_transcript.transcript\_id JOIN exon ON exon\_transcript.exon\_id=exon.exon\_id WHERE gene\_name=’TP53’;

**Exercise: Find the longest spliced transcript of TP53 (taking into account the intron-exon structure)**

* SELECT transcript.transcript\_id, sum(exon.seq\_region\_end-exon.seq\_start +1) AS size FROM gene JOIN transcript ON gene.gene\_id=transcript.gene\_id JOIN exon\_transcript ON transcript.transcript\_id=exon\_transcript..transcript\_id JOIN exon on exon\_transcript.exon\_id=exon.exon\_id WHERE gene\_name=’TP53’ GROUP BY transcript.transcript\_id ORDER BY size DESC LIMIT 1;

**Exercise: How many exons does each transcript have?**

* SELECT transcript\_id, COUNT(\*) FROM exon\_transcript JOIN exon ON exon\_transcript.exon\_id=exon.exon\_id GROUP BY exon\_transcript.transcript ORDER BY COUNT(\*) DESC LIMIT 10;

**Exercise: What is the name of the gene associated with transcript 260392?**

* SELECT gene\_name FROM gene JOIN transcript ON gene.gene\_id=transcript.gene\_id WHERE transcript\_id=260392;

**Exercise: Which chromosome has the most genes and how many are there?**

* SELECT chromosome, COUNT(\*) AS total\_genes FROM gene GROUP BY chromosome ORDER BY total\_genes DESC LIMIT 1;

**Exercise: Which exon is the largest in the genome and how many base pairs are there?**

* SELECT exon\_id, (seq\_region\_end – seq\_region\_start +1) AS length FROM exon ORDER BY length DESC LIMIT 1;

**Exercise: A mutation was found on chromosome 20, position 44002590. Which gene(s) overlap(s) with this position? Select only those genes that have exons that overlap with this mutation. Which genes are they?**

* SELECT gene\_name, exon.seq\_region\_start, exon.seq\_region\_end FROM gene JOIN transcript ON gene.gene\_id=transcript.gene\_id JOIN exon\_transcript ON transcript.transcript\_id=exon\_transcript.transcript\_id JOIN exon ON exon\_transcript.exon\_id=exon.exon\_id WHERE gene.chromosome=20 AND exon.seq\_region\_start <44002590 AND exon\_region\_end 44002590;

**Exercise: A biotype column can be found in both the gene and transcript table. Are there transcripts that have a different biotype from the gene they’re part of? What are their names? Does the same go for status?**

* SELECT COUNT(\*) FROM gene JOIN transcript ON gene.gene\_id=transcript.gene\_id WHERE gene.biotype=transcript.biotype;
* SELECT COUNT(\*) FROM gene JOIN transcript ON gene.gene\_id=transcript.gene\_id WHERE gene.status=transcript.status;

**Exercise: Which transcript has the most exons and how many are there?**

* SELECT transcript.transcript\_id, COUNT(\*) AS total\_exon FROM transcript JOIN exon\_transcript ON transcript.transcript\_id=exon\_transcript.transcript\_id JOIN exon ON exon\_transcript.exon\_id=exon.exon\_id GROUP BY transcript.transcript\_id ORDER BY total\_exon DESC LIMIT 1;

**Exercise: Create a new database for your lab and include following data**

|  |  |
| --- | --- |
| Structure:   * All trainings   + Subject, duration * All lab members   + Name, lastname, birth\_date, training * All equipment   + Name, manufacturer, purchase\_date * All experiments   + Name, performed\_by, equipment\_used, date * All results   + Directory, experiment, status | Solution:  Schermopname |

**Exercise: All experiment equipment purchased after 1st of January 1985.**

* SELECT eq\_name, purchase\_date, ex\_name FROM equipment JOIN experiment ON equipment.equipment\_id=experiment.experiment\_id WHERE purchase\_date > ‘1985-01-01’;

**Exercise: All lab members born in april**

* SELECT \* FROM lab\_members WHERE MONTH(birth\_date)=4;

**Exercise: The number of experiments each lab member conducted**

* SELECT CONCAT(firstname,’ ‘,lastname) AS name, COUNT(\*) AS number\_of\_experiments FROM experiment JOIN lab\_members ON experiment.performed\_by=lab\_members.members\_id GROUP BY name;

**Exercise: A list with all equipment used in a successful experiment**

* SELECT DESTINCT(eq\_name), status, ex\_name FROM results JOIN experiment ON results.results\_id=experiment.experiment\_id JOIN equipment ON experiment.equipment\_id=equipment.equipment\_id WHERE status=’completed’;

**Exercise: A list with all lab members that failed an experiment**

* SELECT CONCAT(firstname,’ ‘,lastname) AS name, ex\_name, status FROM results JOIN experiment on results.experiment=experiment.experiment\_id JOIN lab\_members on experiment.performed\_by=lab\_members.members\_id WHERE status=’failed’;

**Exercise: Who followed which trainings? Number of participants per training**

* SELECT CONCAT(firstname,’ ‘,lastname) AS name, subject FROM lab\_members JOIN members\_in\_training ON lab\_members.members\_id=members\_in\_training.members\_id JOIN trainings ON members\_in\_training.training\_id=trainings.training\_id;
* SELECT subject, COUNT(\*) AS number\_of\_participants FROM trainings JOIN members\_in\_training ON trainings.training\_id=members\_in\_training.training\_id JOIN lab\_members ON members\_in\_training.members\_id=lab\_members.members\_id GROUP BY subject;

# Databases and schema

## What is MySQL Workbench

MySQL Workbench is a visual tool for creating, executing and optimizing SQL queries. It enables us to visually design model, generate and manage databases.

Data modelling enables us to determine the structure of date. There are 3 different types, the first one is the conceptual data model. This determines the structure and/or the relationship between entities with the use of a Entity Relationship Diagram. The second one is the local data model. It determines the structure and/or relationship between tables using foreign keys and visualised in a data structure diagram. The third one is the physical data model. This means the way by which data are stored.

There are different databases models.

* Flat model: single 2-dimensional array of data elements. E.g. spreadsheet
* Hierarchical model: data is organized into tree-like structure and records are connected through links.
* Network model: each record can have multiple parents and child records.
* Relational model: tables are relations, the links are defined by the use of keys.
* Object-relational model: with object-oriented features.
* Object oriented model: data is represented in the form of objects.

## Databases normalization <https://opentextbc.ca/dbdesign01/chapter/chapter-12-normalization/>

Database normalization is the process of organizing the columns and tables of a relational database to reduce data redundancy and improve data integrity.

Unnormalized form (UNF) is the simple database data model. Groups all data in one entity.

1NF (first normal form)

* eliminate repeating groups in individual tables
* create separate table for each set of related data
* identify each set of related data with a primary key
* Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

2NF (second normal form)

* Every non-prime attribute[[1]](#footnote-1) of table is dependent on the whole key of every candidate key.
* There must not be any partial dependency of any column on primary key.

3NF (third normal form)

* Every non-prime attribute is non-transitively dependent[[2]](#footnote-2) on every key.
* Every non-prime attribute of table must be dependent on primary key.

BNCF (Boyce and Codd normal form)

* Any attribute on which some other attribute is full functionally dependent = determinant, every determinant is a candidate key.
* Used in case there is by 3NF overlapping in candidate keys.

## Exercises

**Exercises: Branch**



UNF Branch(branchNo, branchAddress, telNos)

1NF Branch(branchNo,branchAddress)

BranchTelephone(branchNo, telNos)

2NF 1NF

3NF 2NF

**Exercise:** **HEALTH HISTORY REPORT**



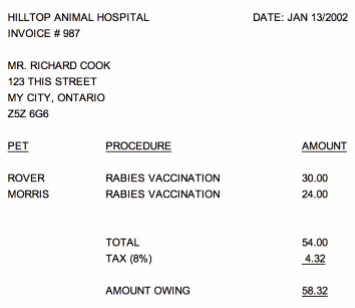
UNFpet(pet\_id, pet\_name, pet\_age, pet\_type, owner, (visit\_date, procedure\_nr, procedure\_name))

1NF pet(pet\_id, pet\_name, pet\_age, pet\_type, owner) pet\_visit(pet\_id, visit\_date, procedure\_nr, procedure\_name)

2NF pet(pet\_id, pet\_name, pet\_age, pet\_type, owner) pet\_visit (pet\_id, visit\_date, procedure\_nr) procedure(procedure\_nr, procedure\_name)

3NF2NF

**Exercise: INVOICE**



UNF INIVOICE(invoice\_no, invoice\_date, cust\_name, cust\_adress, (pet\_name, procedure, amount))

1NF INIVOICE(invoice\_no, invoice\_date, cust\_name, cust\_adress)

INVOICE\_pet(invoice\_no, pet\_id, pet\_name, procedure, amount)

2NF INIVOICE(invoice\_no, invoice\_date, cust\_name, cust\_adress)

INVOICE\_pet(invoice\_no, pet\_id, procedure, amount)

Pet(pet\_id, pet\_name)

3NF INIVOICE(invoice\_no, invoice\_date, cust\_no)

Customer(cust\_no, cust\_name, cust\_street, cust-city, cust\_pstlcd)

INVOICE\_pet(invoice\_no, pet\_id, procedure, amount)

Pet(pet\_id, pet\_name)

**Exercise: company**



UNF STOCKS(company, symbol, headquarters, date, close\_price) 🡺 close price is dependent on company and date

1NF STOCKS(company, symbol, headquarters, date, close\_price) 🡺 close price is dependent on company and date

2NF STOCKS(symbol, date, closing\_price)

Company(company, symbol, headquarters)

3NFSTOCKS(symbol, date, closing\_price)

Company(company, symbol)

Head(company, headquarters)

**Exercise: grade**



UNF student\_grade\_report(StudNo, StudName, Major, CourseNo, CourseName, InstructorNO, InstructorName, InstructorLoc, Grade)

1NF StudentCourse(StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLoc, grade)

Student(StudentNo, StudentName, Major)

2NF CourseGrade (StudentNo, courseNo, Grade)

CourseInstructor(CourseNo, CourseName, InstructorNo, InstuctorName, InstructorLoc)

Student(StudentNo, StudentName, Major)

3NFCourseGrade (StudentNo, courseNo, Grade)

Student(StudentNo, StudentName, Major)

Course(CourseNo, CourseName, InstuctorNo)

Instructor(InstructorNo, InstructorName, InstructorLoc)

BCNF CourseGrade (StudentNo, courseNo, Grade)

Student(StudentNo, StudentName)

Course(CourseNo, CourseName, InstuctorNo)

Instructor(InstructorNo, InstructorName, InstructorLoc)

Advisor(advisor, StudentNo)

Major(major, advisor)

1. Each Student may major in several subjects.
2. For each Major, a given Student has only one Advisor.
3. Each Major has several Advisors.
4. Each Advisor advises only one Major.
5. Each Advisor advises several Students in one Major.

**Exercise: video**

video(title,director,serial)

customer(name,addr,memberno)

hire(memberno,serial,date)

title->director,serial

serial->title

serial->director

name,addr -> memberno

memberno -> name,addr

serial,date -> memberno

1NF video (title, director, serial)

Customer( name, addr, serial, date, memberno)

2NF video(title,director,serial)

customer(name,addr,memberno)

hire(memberno,serial,date)

3NF video(title,serial)

Serial(directoryl, serial)

Customer(name, addr, memberno)

Hire(memberno, serial, data)

3BCNF 3NF

**Exercise: drinker**

Convert to BCNF 🡺 Drinkers(name, addr, beersLiked, manf, favBeer)

UNF  Drinkers(name, addr, beersLiked, manf, favBeer)

1NF  Drinker(name, addres, favbeer) id(name, beersliked, manf)

2NF beerliked -->manfn name-->beersliked

3NF Drinker(name, addres, favbeer)  liked(name, beersliked)  man(beersliked, manf)

BCNF 3NF

**Exercise: ABC manufacturing order form**

ABC MANUFACTURING ORDER FORM DATE: AUG 30, 2002

ORDER # 9932

MR. S.D. KURTZ SHIPPING ADDRESS: 456 NO STREET

123 THAT STREET HAMILTON, ONTARIO

TORONTO, ONTARIO L6K 5J4

A9B 8C7

PHONE: (416) 879-0045 (416) 786-3241 CUSTOMER DISCOUNT: 3%

ITEM # PRODUCT CODE DESCRIPTION QTY BACKORDERED FILLED PRICE/UNIT AMOUNT

1 FR223 HALF SIZE REFRIGERATOR 2 0 2 750.99 1501.98

2 TB101 PATIO TABLE 5 2 3 150.00 450.00

3 CH089 PATIO CHAIRS 20 0 20 35.00 700.00

TOTAL 2651.98

DISCOUNT AMT 79.56

AMOUNT OWING 2572.42

UNFORDER(order\_nr, date, cust\_name, cust\_address, ship\_adress, phone1,phone2, cust\_disc(item, product\_code, description, QTY, filled, Price))

1NF ORDER(order\_nr, date, cust\_name, cust\_address, ship\_adress, phone1,phone2, cust\_disc)

ORDER\_id(order\_nr, product\_code, description, QTY, filled, Price)

2NF ORDER(order\_nr, date, cust\_name, cust\_address, ship\_adress, phone1,phone2, cust\_disc)

ORDER\_id(order\_nr, product\_code QTY, filled)

PRODUCT(product\_code, description, price)

3NFORDER(order\_nr, date, cust\_nr)

ORDER\_id(order\_nr, product\_code QTY, filled)

PRODUCT(product\_code, description, price)

CUSTOMER(cust\_nr, cust\_name, cust\_adress, ship\_adress, phone1, phone2, cust\_disc,

**Exercise: Gallery Customer History Form**

**Gallery Customer History Form**

Customer Name

Jackson, PhonePhone (206) 284-6783

123 – 4th Avenue

Fonthill, ON

L3J 4S4

Purchases Made

Artist Title Purchase DateSales Price

03 - Carol ChanningLaugh 09/17/200009/17/2000 7000.00

15 - Dennis SouthSouth toward Emerald Sea05/11/2000 1800.00

03 - Carol Channing At the Movies 02/14/2002 5550.00

15 - Dennis SouthSouth toward Emerald Sea07/15/2003 2200.00

UNFPurchase(cust\_id, cust\_name, cust\_adres, cust\_phone, (artist\_nr, artist\_name, title, Pur\_date, sale\_price))

1NF Purchase(cust\_id, cust\_name, cust\_adres, cust\_phone)

Cust\_art (cust\_id, art\_nr, artist\_name, title, Pur\_date, sale\_price)

2NF Purchase(cust\_id, cust\_name, cust\_adres, cust\_phone)

Cust\_art (cust\_id, art\_nr, Pur\_date, sale\_price)

Art (art\_nr, artist\_name, title, artist\_id)

3NF Purchase(cust\_id, cust\_name, cust\_street, cust\_city, cust\_prov, cust\_pstlcd, cust\_phone)

Cust\_art (cust\_id, art\_nr, title, Pur\_date, sale\_price)

Art(art\_nr, title, artist\_id)

Artist(artis\_id, artis\_fname, artist\_lname)

**Exercise: Good News Grocers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Department** | **Product Code** | **Aisle Number** | **Price** | **Unit of Measure** |
|  |  |  |  |  |
| Produce | 4081 | 1 | 0.35 | lb |
| Produce | 4027 | 1 | 0.90 | ea |
| Produce | 4108 | 1 | 1.99 | lb |
|  |  |  |  |  |
| Butcher | 331100 | 5 | 1.50 | lb |
| Butcher | 331105 | 5 | 2.40 | lb |
| Butcher | 332110 | 5 | 5.00 | lb |
|  |  |  |  |  |
| Freezer | 411100 | 6 | 1.00 | ea |
| Freezer | 521101 | 6 | 1.00 | ea |
| Freezer | 866503 | 6 | 5.00 | ea |
| Freezer | 866504 | 6 | 5.00 | ea |

UNFdept(dept, aisle\_nr, (prod\_cod, price, um))

1NF dept(dept\_id, dept\_name, aisle\_nr)

Dept\_product(dept, prod\_code, price, um)

2NF dept(dept\_id, dept\_name, aisle\_nr)

Product(prod\_cod, price, um, dept\_id)

3NF 2NF

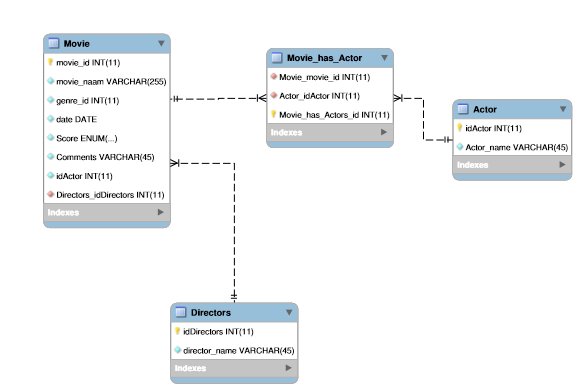
## Forward/reverse engineering

Forward engineering is a function in MySQL Workbench. It’s used to export your schema design to a MySQL server. It’s located under Database > Forward engineer.

Reverse engineering is a function in MySQL Workbench. It’s used to export a table to a schema. It’s located under Database > reverse Engineer.

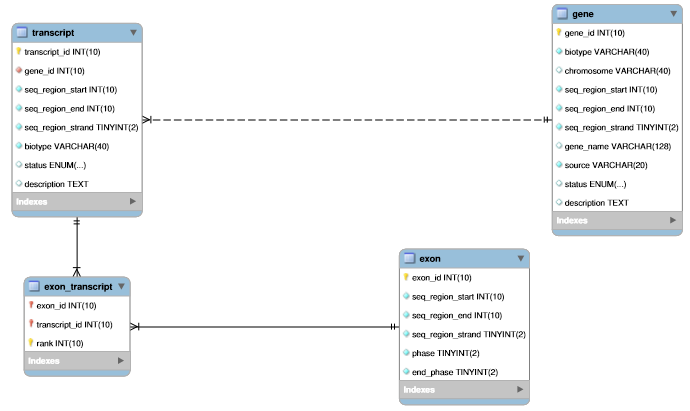
## Exercises

**Exercise movie database: forward engineering**

* Create a MySQL table to track the movies you have watched:
  + Movie title
  + Genre: action, comedy, drama, horror, science fiction
  + Date you watched to movie
  + Score: 0-10
  + Comments
* Create a table to store your favourite directors and link it with the movie table
* Create a table to store your favourite actors and link it with the movie table
* Forward engineer your tables to your database
* Add some rows to the table you have created

**Exercises: reverse engineering**

* Reverse engineer the model of the bioinf\_testdb
  + Check out the relationships between the different tables
  + Give the names of all the keys used



## Exporting/import data

The command $ mysqldump [opt] *db* > *db.sql*. This command will dump a MySQL database, compress and save it into a file. It’s is useful for backup. The data can be than exported to JSON, CSV, HTML or XML file. In import data is the entire dump file imported. The data has to be an CSV, existing table, SQL or JSON file.

**Exercise**

* Export the data in the trajectories table 🡺 export data from the trajectory table in a CSV file
* Empty your table (TRUNCATE) 🡺 TRUNCATE trajectory
* Import data into the trajectories table using your export file 🡺 import the data from the CSV file into the trajectories table

## MySQL and other languages

You can execute SQL SELECT command to take a backup of any table. To take a complete database dump you will need to write separate query for separate table. Each table will be stored into separate text file. <https://www.w3schools.com/php/php_mysql_select.asp>

|  |  |
| --- | --- |
| **PHP – MySQLi Procedural: Connect** | **PHP – MySQLi Procedural: Query** |
| The following example is used to prepare a statement and bound parameters in MySQL  <?php  $servername = "localhost";  $username = "username";  $password = "password";  // Create connection  $conn = mysqli\_connect($servername, $username, $password);  // Check connection  if (!$conn) {  die("Connection failed: " . mysqli\_connect\_error());  }  echo "Connected successfully";  ?> | <?php  // Create database  $sql = "CREATE DATABASE myDB";  if (mysqli\_query($conn, $sql)) {  echo "Database created successfully";  } else {  echo "Error creating database: " . mysqli\_error($conn);  }  mysqli\_close($conn);  ?> |
| **PHP – MySQLi Procedural: INSERT (prepared)** | **PHP – MySQLi Procedural: SELECT** |
| <?php  // prepare and bind  $stmt = mysqli\_stmt\_init($conn);  mysqli\_stmt\_prepare($stmt, "INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");  mysqli\_stmt\_bind\_param ($stmt,"sss", $firstname, $lastname, $email);  // set parameters and execute  $firstname = "John";  $lastname = "Doe";  $email = "john@example.com"; mysqli\_stmt\_execute ($stmt);  echo "New record created successfully";  mysqli\_stmt\_close($stmt); mysqli\_close($conn);  ?> | <?php  $sql = "SELECT id, firstname, lastname FROM MyGuests";  $result = mysqli\_query($conn, $sql);  if (mysqli\_num\_rows($result) > 0) {  // output data of each row  while($row = mysqli\_fetch\_assoc($result)) {  echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .  $row["lastname"]. "<br>";  }  } else {  echo "0 results";  }  mysqli\_close($conn);  ?> |
| **PHP – MySQLi Object-Oriented: Connect, Query, Insert** | **PHP – MySQLi Object-Oriented: Insert (prepared)** |
| <?php  $conn = new mysqli($servername, $username, $password);  $sql = "CREATE DATABASE myDB";  $conn->query($sql);  $sql = "INSERT INTO MyGuests (firstname, lastname, email) VALUES ('John', 'Doe', 'john@example.com')";  if ($conn->query($sql) === TRUE) {  $last\_id = $conn->insert\_id;  echo "New record created successfully. Last inserted ID is: " .  $last\_id;  }  ?> | <?php  // prepare and bind  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");  $stmt->bind\_param("sss", $firstname, $lastname, $email);  // set parameters and execute  $firstname = "John";  $lastname = "Doe";  $email = ["john@example.com";](mailto:john@example.com)  $stmt->execute();  echo "New records created successfully";  $stmt->close();  $conn->close();  ?> |
| **PHP – MySQLi Object-Oriented: SELECT** |  |
| <?php  $sql = "SELECT id, firstname, lastname FROM MyGuests";  $result = $conn->query($sql);  if ($result->num\_rows > 0) {  // output data of each row  while($row = $result->fetch\_assoc()) {  echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .  $row["lastname"]. "<br>";  }  }  $conn->close();  ?> |  |

# NoSQL

A NoSQL database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases. It’s more flexible but it doesn’t provide atomicity, consistency, isolation and durability (it is a set of properties of DB transactions intended to guarantee validity even in the event of errors).

There have been various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap. For example: column store, key-value store, graph store, multi-model and document store.

* The column store consist of an unique name, value and timestamp. Number of columns could be different from the row.
* They key-value store consist of data stored in dictionary or hash[[3]](#footnote-3). It has unique key per record. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.
* The graph store is designed for data whose relations are well represented as a [graph](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)) consisting of elements interconnected with a finite number of relations between them. It’s represented using nodes, edges and properties. It can simple and rapid retrieve data.
* The multi model is designed to support multiple data models against a single, integrated backend. Document, graph, relational, and key-value models are examples of data models that may be supported by a multi-model database.
* The document store is a [computer program](https://en.wikipedia.org/wiki/Computer_program) designed for storing, retrieving and managing document-oriented information, also known as [semi-structured data](https://en.wikipedia.org/wiki/Semi-structured_model). It’s a subclass of the key-value store. It relies on internal structure in the *document* in order to extract [metadata](https://en.wikipedia.org/wiki/Metadata)[[4]](#footnote-14274). All information for given object stored in single instance. Every stored object can be different from every other. The documents are identified by unique key.

## MongoDB

MongoDB is document store that has no predefined data formats. Change in type and form has no effect on database and existing stored documents. There is no normalization. MongoDB stores data in flexible, JSON-like documents. JSON format is a JavaScript Object notation. It exists of objects and arrays and attribute-value pairs.

Figure 4: JSON format

## Install mongoDB

Sudo nano /etc/yum.repos.d/mongodb-org-3.6.repo

Sudo yum install –y mongodb.x86\_64

[mongodb-org-3.6]  
name=MongoDB Repository  
baseurl=https://repo.mongodb.org/yum/redhat/7/mongodb-org/3.6/x86\_64/  
gpgcheck=1  
enabled=1  
gpgkey=https://www.mongodb.org/static/pgp/server-3.6.asc

Sudo yum install –y mongodb-server.x86\_64

Sudo service mongod start

Start mongod.service

Mongo

## mongoDB – databases and collections

BSON documents are stored in collections, collections in databases

* use create |select a DB in mongo shell or create a DB in

|mongo shell or

* db.mynewcollection.insert() |create a collection and insert
* show |see all DB on server all DB on the server
* db.help () |list all methods you can use on your

|db object

* show |see all collections in a database all

|collections in a database

* db.collection.help() |list all methods you can us on you

|collection object

* db.collection.find().help() |list all cursor methods
* db.collection.insert() |add new documents to a collection
* db.collection.insertOne () |insert a single document into a

|collection

* db.collection.insertMany() |insert multiple documents into a

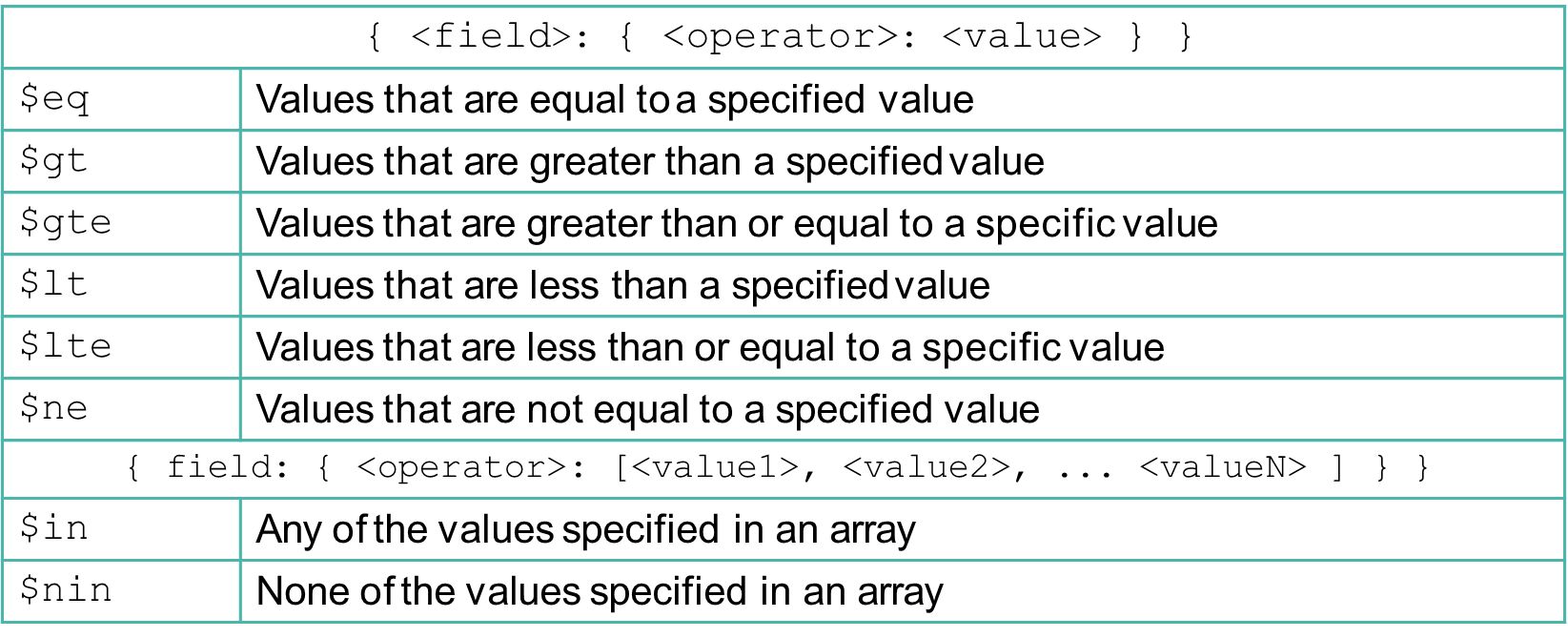
|collection

* db.collection.find() Or findOne() |retrieve documents from collection

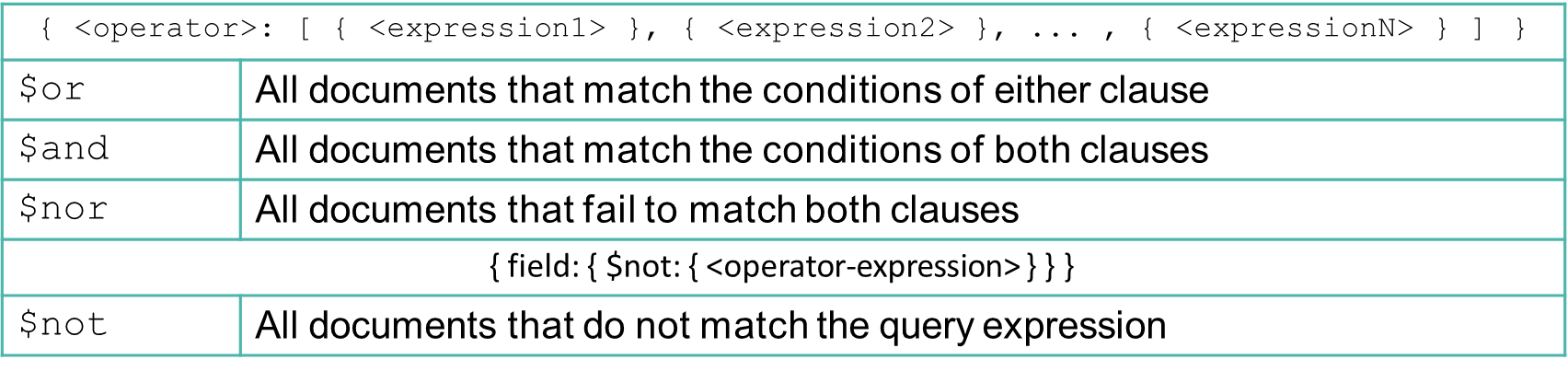
**Data types:**

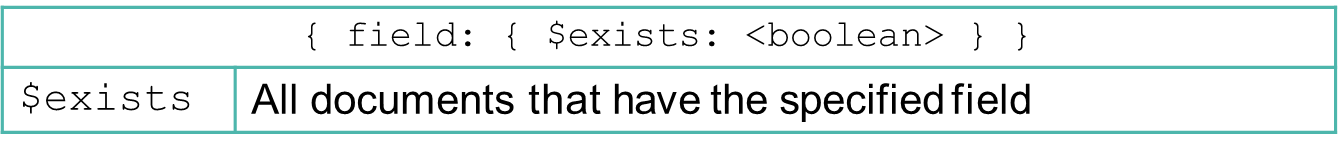
* MongoDB uses the dot notation to access the elements of an array and to access the fields of an embedded document.
* Primary key: By default, MongoDB creates a unique index on the \_id field during the creation of a collection.
* ObjectId
  + small, likely unique, fast to generate, and ordered
    - a 4-byte value representing the seconds since the Unix epoch,
    - a 3-byte machine identifier,
    - a 2-byte process id, and
    - a 3-byte counter, starting with a random value.
* String
* Timestamp
  + For Internal MongoDB use and values are 64 bit values where:
    - the first 32 bits are a time\_t value (seconds since the Unix epoch)
    - the second 32 bits are an incrementing[[5]](#footnote-4) ordinal[[6]](#footnote-5) for operations within a given second.
* Date
  + number of milliseconds since the Unix epoch

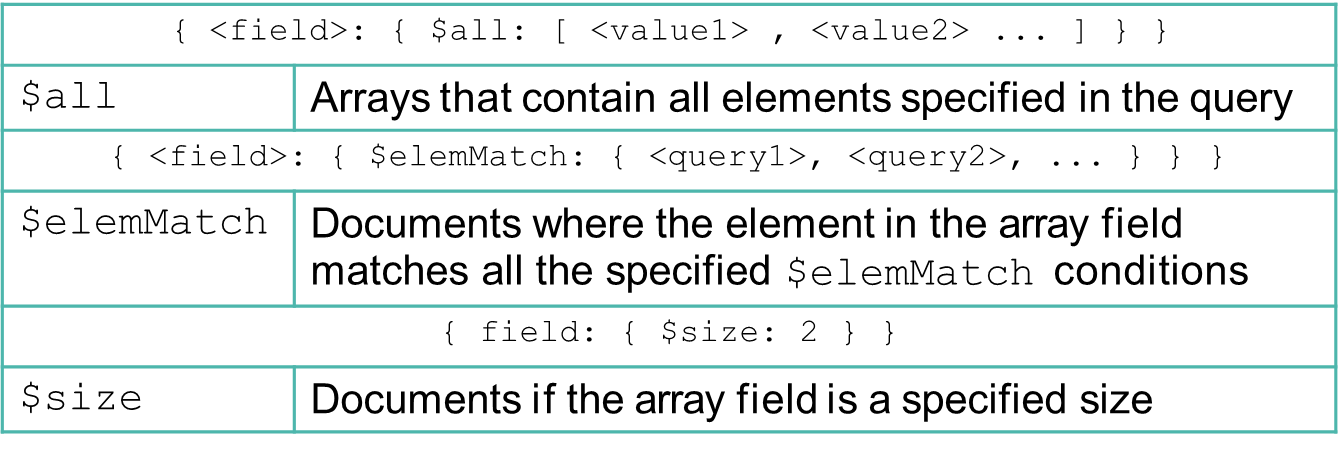
### Comparison



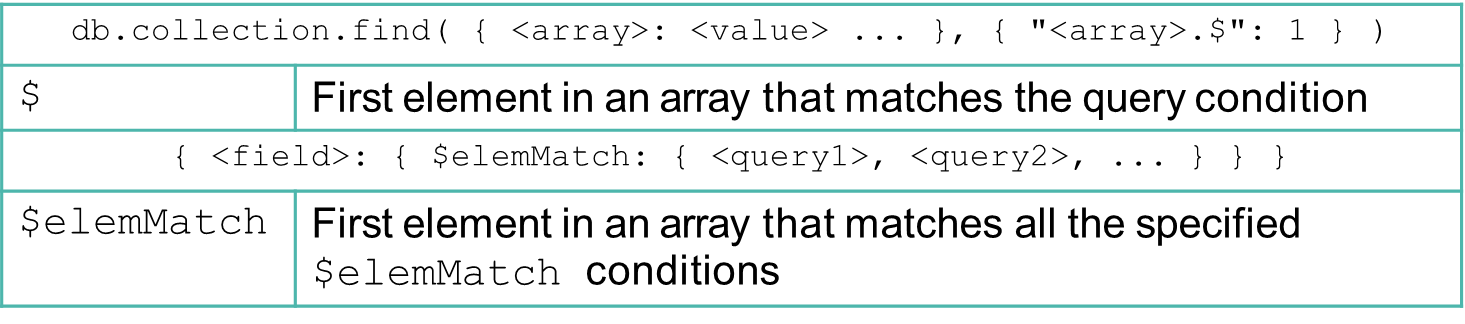
### Query operators







### Projection operators

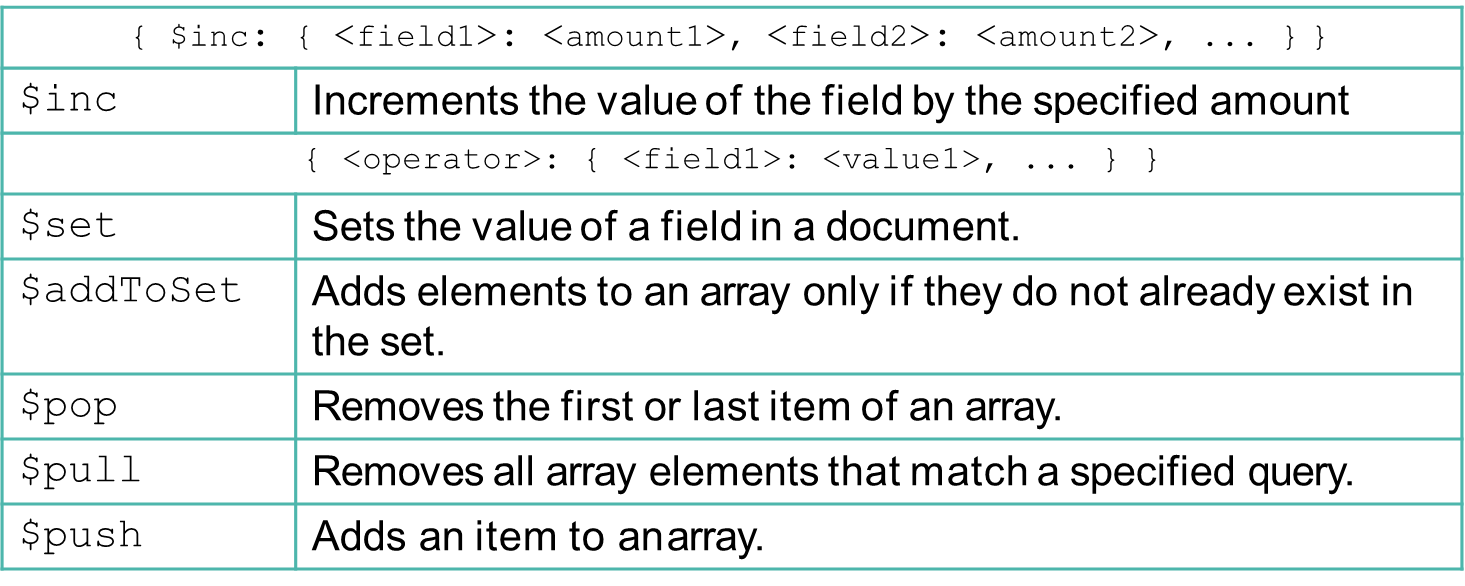


### Cursor modifier

* Methods that modify the way that the underlying query is executed
  + sort()= Returns results ordered according to a sort specification
  + count() = Returns the number of documents in the result set
  + hasNext() = Returns true if the cursor has documents and can be iterated
  + next() = Returns the next document in a cursor.
  + limit() = Constrains the size of a cursor’s result set.
  + skip() = Returns a cursor that begins returning results only after passing or skipping a number of documents.
  + size() = Returns a count of the documents in the cursor after applying skip() and limit() methods

### Update operations

* Modify existing documents in a collection
  + db.collection.update()
  + db.collection.updateOne()
  + db.collection.updateMany()
  + db.collection.replaceOne()



### Delete operations

* Add new documents to a collection
  + db.collection.remove()
  + db.collection.deleteOne()
  + db.collection.deleteMany()

### Aggregation

* aggregate() method
  + Return states with populations above 10 Million
  + db.zipcodes.aggregate( [{ $group: { \_id: "$state", totalPop: { $sum: "$pop" } } },{ $match: { totalPop: { $gte: 10\*1000\*1000 } } } ] )
* Many aggregation pipeline operators
  + $sum, $avg, $group, $match, $limit, $skip, $sort, $concat, …

## Exercises <https://www.w3resource.com/mongodb-exercises/> and <http://nicholasjohnson.com/mongo/course/workbook/>

**Exercises: Create a collection users and add yourself to it (name, age, sex). Add some more users using a different command**

db.users.insert({name:"caroline",age:23, sex:"V"})

db.users.insert({name:"car",age:23, sex:"M"})

db.users.insert({name:"coar",age:123, sex:"M"})

db.users.insert({name:"oar",age:13, sex:"V"})

db.users.insert({name:"oear",age:413, sex:"V"})

db.users.insertOne({name:"oear",age:413, sex:"V"})

db.users.find()

**Exercise: Populate your collection using the contents of the file mongo1.txt**

* Copied the mongo1.txt and past on the mongo server.

**Exercise: Select all documents in the collection**

* Db.useres.find()

**Exercise: Select all documents where status field has the value A**

* Db.users.find({status:”A"})

**Exercise: Select all documents in the collection where the status equals "A" and either age is less than 30 or type equals 1**

* Db.users.find({status:”A”, age :{$lt:30}})

OR

* Db.users({status:”A”,$or:[{age:{$lt:30}},{type:1}]})

**Exercise: Retrieve all documents from the users collection where status equals either "P"or "D”**

* Db.users.find({$or: [{status:"P"},{status:"D"}]})

OR

* db.users.find({"status":{$in:["P","D"]}})

**Exercise: Retrieve all documents where the favourites field is an embedded document that contains  only the fields artist equal to "Picasso" and food equal to "pizza", in that order**

* db.users.find( { favorites: { artist: "Picasso", food: "pizza" } } )

**Exercise: Use the dot notation to match all documents where badges is an array that contains "black" as the first element**

* Db.users.find({"badges.0": "black"})

OR

* db.users.find{"badges:"black"},{"badges.$":1}})

**Exercise: Retrieve all documents where the finished array contains at least one element that is greater than ($gt) 15 and less than ($lt) 20**

* db.users.find({finished: {$elemMatch:{$gt:15,$lt:20}}})

**Exercises: Update all documents where Picasso is the favorite artist so that the value of the favorites.artist field is "Pisanello" and the value of the type field is 3**

* Db.users.updateMany({"favorites.artist":"Picasso"},{$set{"favorites.artist":"Pisanello",type:3}})

OR

* db.users.update({"favorites.artist":"Picasso"},{$set:{"favorites.artist":"Pisanello",type:3}},{multi:true})

🡺favorite.artist and not favorites:{} because other ways we will search for an array favorites who only contains artist. We want to search within favorites to artist

**Exercises: Replace the first document that matches the filter name equals "abc" with the new document { name: "amy", age: 34, type: 2, status: "P", favorites: { "artist": "Dali",  food: "donuts" } }**

* db.users.replaceOne({ name: "abc" }, { name: "amy", age: 34, type: 2, status: "P", favorites: { "artist": "Dali", food: "donuts" } })

**Exercises: Replace the first document that matches the filter name equals "xyz" with the new document (do not use the same method as before) { name: "mee", age: 25, type: 1, status: "A", favorites: { "artist": "Matisse", food: "mango" } }**

* db.users.update( { name: "xyz" }, { name: "mee", age: 25, type: 1, status: "A", favorites: { "artist": "Matisse", food: "mango" } })

**Exercises: Remove all documents from the users collection where the status field equals "P”**

* db.users.remove( { status : "P" } )

**Exercises: Remove the first document from the users collection where the status field equals ”D” (2 ways)**

* db.users.deleteOne( { status: "D" } )

OR

* db.users.remove({status: "D"},1)

🡺1 staat vr waar 🡺 dus er w slechts 1 record verwijderd.

**Exercise: Transform these SQL statements to mongo statements**

* INSERT INTO users(user\_id, age, status) VALUES ("bcd001", 45, "A");
  + db.users.insert({ user\_id: "bcd001", age: 45, status: "A" })
* SELECT \* FROM users WHERE status != "A”;
  + db.users.find({status:{$ne:"A"}})
* SELECT \* FROM users WHERE age > 25;
  + Db.users.find({age: {$gt:25}})
* SELECT \* FROM users WHERE status = "A" ORDER BY user\_id DESC;
  + db.users.find( { status: "A" } ).sort( { user\_id: -1 } )
* SELECT COUNT(\*) FROM users;
  + Db.users.count()

OR

* + db.users.find().count()
* SELECT COUNT(user\_id) FROM users;
  + Db.users.count({user\_id: {$exists:true}})

OR

* + db.users.find({user\_id:{$exists:true}}).count()
* SELECT DISTINCT(status) FROM users;
  + Db.users.distinct("status")
* SELECT \* FROM users LIMIT 1;
  + Db.users.find().limit(1)

OR

* + db.users.findOne()
* SELECT \* FROM users LIMIT 5 SKIP 10;
  + Db.users.find().limit(5).skip(10)
* UPDATE users SET age = age + 3 WHERE status = "A”;
  + db.users.update({ status: "A" } ,{ $inc: { age: 3 } },{ multi: true })
* DELETE FROM users WHERE status = "D”;
  + Db.users.remove({status:"D"})

**Exercises: Return average city population by state**

* db.zips.aggregate([{$group:{\_id:{state:"$state",city:"$city"},pop:{$sum:"$pop"}}},{$group:{\_id:"$\_id.state",avgCityPop:{$avg:"$pop"}}},{$sort:{avgCityPop:-1}}] )

**Exercises: Return largest and smallest cities by state**

* db.zipcodes.aggregate( [  { $group:  { \_id: { state: "$state", city: "$city" },pop: { $sum: "$pop" } }, { $sort: { pop: 1 } }, { $group: {  \_id : "$\_id.state", biggestCity:  { $last: "$\_id.city" }, biggestPop:   { $last: "$pop" },smallestCity: { $first: "$\_id.city" },smallestPop:  { $first: "$pop" } } },{ $project: { \_id: 0, state: "$\_id",biggestCity:  { name: "$biggestCity",  pop: "$biggestPop" }, smallestCity: { name: "$smallestCity", pop: "$smallestPop" } }}] )

OR

* Db.zips.aggregates([{$group: {$\_id:{state: "$state", city:"$city"}, pop: {$sum:"$pop"}}},{$sort:{pop:1}}, {$group:{\_id:"$\_id.state",biggestCity:{$last:"$\_id.city"}, smallestCity:{$first:"$\_id.city"}}} ] )

**Exercises: Return the first 10 genes (alphabetically) on chromosome 22**

* Import the protein coding genes from file protCodingGenes.json
* $ mongorestore --collection protCodingGenes --db test protCodingGenes.bson
* Db.protCodingGenes.find({“Chromosome Name”:22}).{“Associated Gene Name”:1}.sort({“Associated Gene Name”:1}).limit(10)

**Exercises: Return the last but one group of 10 genes (by position) on chromosome 12**

* Db.protCodingGenes.find({“Chromosome Name”:12},{}).{“Associated Gene Name”:1}).sort({“Gene Start (bp)”:1}).limit(10).skip(10)

**Exercises: Return the number of unique gene names**

* Db.protCodingGenes.distinct(“Associated Gene Name”).length

**Exercises: Return the 50 most common genes and their number of occurrences**

* Db.protCodingGenes.aggregate([{$group:{\_id:”$Associated Gene Name”.count{$sum:1}}}.{$sort:{count:-1}}.{$limit:50}])

| Sum 1= takes the sum from each record that is generated

| $associated = already exist so now an id is created

**Exercises: Return a sorted list of the number of genes per chromosome**

* Db.protCodingGenes.aggregate([{$group:{\_id:”$Chromosome Name”,count:{$sum:1}}},{$sort:{count:-1}}])

# Databases, APIs & version control

## Databases in bioinformatics

Databases in bioinformatics are libraries of life sciences information, collected from scientific experiments, published literature, high-throughput experiment technology, and computational analysis. They contain information from research areas including [genomics](https://en.wikipedia.org/wiki/Genomics), [proteomics](https://en.wikipedia.org/wiki/Proteomics), [metabolomics](https://en.wikipedia.org/wiki/Metabolomics), [microarray](https://en.wikipedia.org/wiki/Microarray) gene expression, and [phylogenetics](https://en.wikipedia.org/wiki/Phylogenetics).﷟HYPERLINK "https://en.wikipedia.org/wiki/Biological\_database" Information contained in biological databases includes gene function, structure, localization (both cellular and chromosomal), clinical effects of mutations as well as similarities of biological sequences and structures. The characterisation of different types of databases is based on several properties. The properties are:

* Type of data: Nucleotide sequences, Protein sequences, Gene expression data, Metabolic pathways or 3D structures
* Data entry and quality control
  + Data deposited directly
  + Appointed curators add and update data
  + Treatment of erroneous[[7]](#footnote-6) data: removed, or marked
  + Type and degree of error checking
* Primary or derived data
  + Primary databases: experimental results
  + Secondary databases: results of analysis of primary databases
  + Aggregate of many databases
  + Consolidation[[8]](#footnote-7) of data refers to the collection and integration of data from multiple sources into a single destination.
  + Combination of data
* Technical design
  + Flat files is a [database](https://en.wikipedia.org/wiki/Database) stored as an ordinary unstructured file called a "flat file".
  + Relational database
  + Object oriented database is a database model where they work with objects.
* Maintainer status
  + Large, public institution (EMBL, NCBI)
  + Quasi-academic institute (Swiss Institute of Bioinformatics, TIGR)
  + Academic group or scientist
  + Commercial company
* Availability
  + Publicly available, no restrictions
  + Available, but with copyright
  + Accessible, but not downloadable
  + Academic, but not freely available
  + Commercial

## Identifiers and accession codes

There are 2 different ways to identify an entry. The first one is the identifier. It’s a string of letters and digits, it can usually change. The second one is accession code. This is a number that uniquely identifies an entry in its database and it stays the same, so it is stable.

## Primary, secondary and others databases

### Primary databases

In bioinformatics, and indeed in other data-intensive research fields, databases are often categorised as primary or secondary. **Primary databases** are populated with experimentally-derived data such as nucleotide sequence, protein sequence or macromolecular structure. Experimental results are submitted directly into the database by researchers, and the data are essentially archival in nature. Once given a database [accession](https://www.ebi.ac.uk/training/online/glossary/accession) number, the data in primary databases are never changed: they form part of the scientific record.

Some examples are:

* The European Nucleotide Archive (ENA) provides a comprehensive record of the world's nucleotide sequencing information, covering raw sequencing data, sequence assembly information and functional annotation. It contains DNA and RNA sequences and exist of 3 databases namely Sequence Read Archive, Trace Archive and EMBL Nucleotide Sequence Database. The database is maintained by European Bioinformatics Institute. The downloaded information can be stored in XML, HTML, FASTA, FASTQ files. (<http://www.ebi.ac.uk/ena>)
* GenBank ® is the NIH genetic sequence database, an annotated collection of all publicly available DNA sequences and their protein translation. It contains more than 100000 distinct organisms. The database is maintained by National Centre for Biotechnology Information (NCBI). The entries are retrievable by NCBI GenBank webpage. It GenBank format is flatfile who contains the following sections (see figure 6).
* The DNA Data Bank of Japan (DDBJ) is a biological database that collects DNA sequences. It’s the only nucleotide sequence data bank in Asia. (<http://www.ddbj.nig.ac.jp/>)

Tabel 1: diffrence between primare and secundaire database

|  |  |  |
| --- | --- | --- |
|  | **Primary database** | **Secondary database** |
| **Synonyms** | Archival database | Curated database; knowledgebase |
| **Source of data** | Direct submission of experimentally-derived data from researchers | Results of analysis, literature research and interpretation, often of data in primary databases |
| **Examples** | * [ENA](https://www.ena.org/), [GenBank](http://www.ncbi.nlm.nih.gov/genbank/) and [DDBJ](http://www.ddbj.nig.ac.jp/) (nucleotide sequence) * [ArrayExpress Archive](https://www.ebi.ac.uk/arrayexpress/) and [GEO](http://www.ncbi.nlm.nih.gov/geo/) (functional genomics data) * [Protein Data Bank](https://www.ebi.ac.uk/pdbe/node/1) (PDB; coordinates of three-dimensional macromolecular structures) | * [InterPro](https://www.ebi.ac.uk/interpro/) (protein families, motifs and domains) * [UniProt Knowledgebase](http://www.uniprot.org/help/uniprotkb) (sequence and functional information on proteins) * [Ensembl](http://www.ensembl.org/) (variation, function, regulation and more layered onto whole genome sequences) |

### Secondary database

By contrast, **secondary databases** comprise[[9]](#footnote-8) data derived from the results of analysing primary data. Secondary databases often draw upon information from numerous sources, including other databases (primary and secondary), controlled vocabularies and the scientific literature. They are highly curated, often using a complex combination of computational algorithms and manual analysis and interpretation to derive new knowledge from the public record of science. Some examples are:

* The RefSeq contains DNA, RNA and their protein products. It is annotated[[10]](#footnote-9) and curated[[11]](#footnote-10). There is a single record for each natural biological molecule.
* OMIM[[12]](#footnote-9728) is a catalogue of human genes and genetic disorders and traits. It is based on selection and review of published peer-reviewed literature.
* The HapMap is a haplotype map of the human genome. It contains the genetic variants affecting health, disease and responses to drugs and environmental factors.

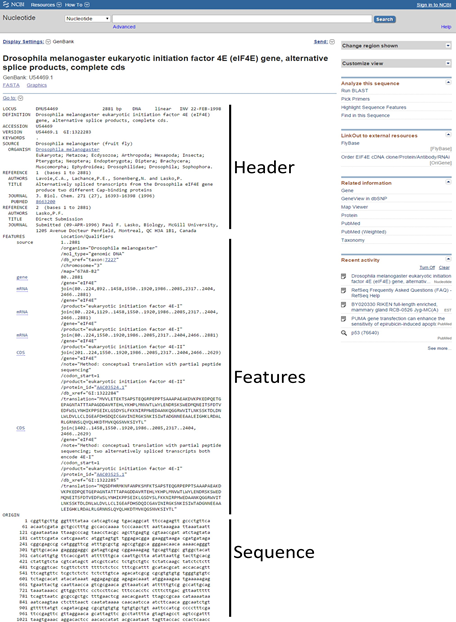


Figure 6: flatfile with 3 main sections

### Other nucleic acid databases

* Gene expression databases
  + Mostly microarray data
  + a.o. Gene Expression Omnibus, Expression Atlas, …
* Gene ontology
  + Relationships between concepts within a domain
* Genome databases
  + Annotated and analyzed genome sequences
  + a.o. Ensembl (Genomes), Flybase, Wormbase, …
* Phenotype databases
  + a.o. PhenCode
* RNA databases
  + a.o. miRBase, LNCipedia, …

### Sequencing databases

* Datasets from sequencing experiments
  + Sequence Read Archive
    - Hosted by NCBI
    - Raw data in BAM-format
    - Experimental metadata available
  + European Genome-phenome Archive
    - Hosted by EMBL-EBI
    - Data not publicly available

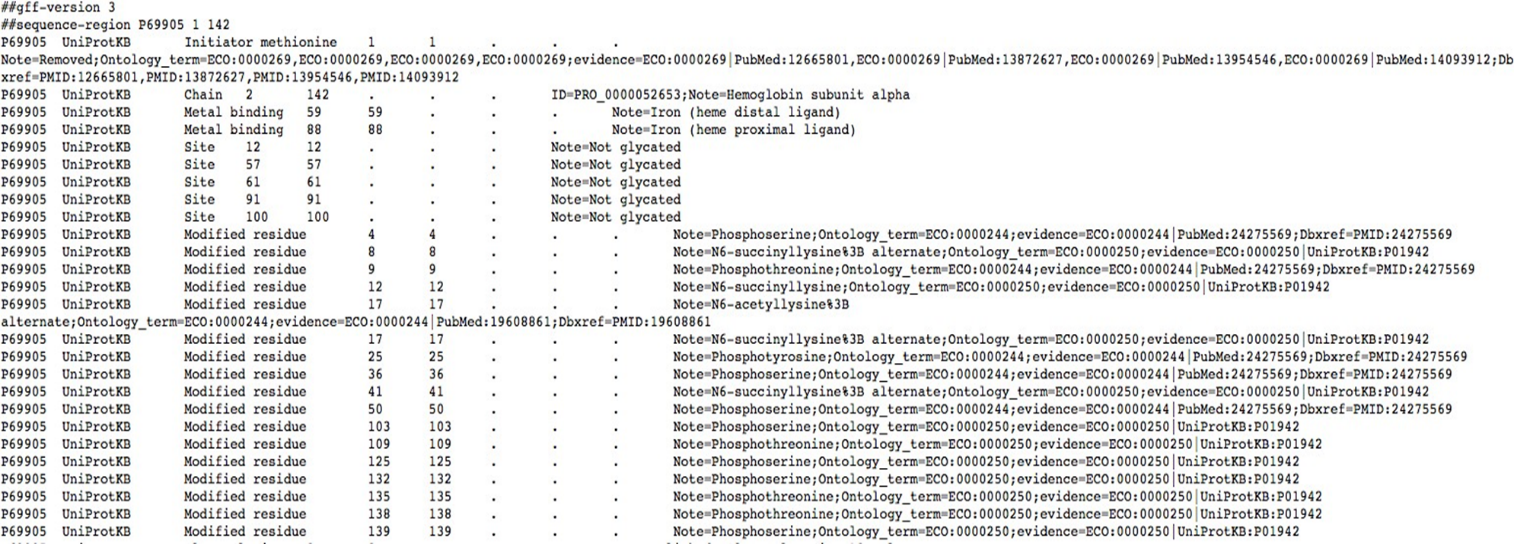
### Protein databases

* Protein sequence
  + Derived from translation of nucleotide sequences
    - secondary databases: NCBI Protein and trEMBL
  + Computational analysis, manual review and annotation
    - SwissProt
* Protein structure
  + a.o. Protein Data Bank, NCBI Structure

## The General Feature Format (GFF)

The general feature format (gene-finding format, generic feature format, GFF) is a [file format](https://en.wikipedia.org/wiki/File_format) used for describing [genes](https://en.wikipedia.org/wiki/Gene) and other features of [DNA](https://en.wikipedia.org/wiki/DNA), [RNA](https://en.wikipedia.org/wiki/RNA) and [protein](https://en.wikipedia.org/wiki/Protein) sequences. The file type is a tab delimited[[13]](#footnote-11), this is a text format which is widely accepted for creation and viewing by most spreadsheet programs and editor applications. There is one line per feature, all but the final field in each feature line must contain a value; "empty" columns should be denoted with a '.’ (see figure)

* **seqname** - name of the chromosome or scaffold; chromosome names can be given with or without the 'chr' prefix. **Important note**: the seqname must be one used within Ensembl, i.e. a standard chromosome name or an Ensembl identifier such as a scaffold ID, without any additional content such as species or assembly. See the example GFF output below.
* **source** - name of the program that generated this feature, or the data source (database or project name)
* **feature** - feature type name, e.g. Gene, Variation, Similarity
* **start** - Start position of the feature, with sequence numbering starting at 1.
* **end** - End position of the feature, with sequence numbering starting at 1.
* **score** - A floating point value.
* **strand** - defined as + (forward) or - (reverse).
* **frame** - One of '0', '1' or '2'. '0' indicates that the first base of the feature is the first base of a codon, '1' that the second base is the first base of a codon, and so on..
* **attribute** - A semicolon-separated list of tag-value pairs, providing additional information about each feature.



## Genome browsers

A genome browser is a Graphical interface for genomic data. An example is the UCSC[[14]](#footnote-2556) genome browser. Here you can search by Search by gene name and by location (chrN:startposition-stoppostion). Another example is the Ensembl genome browser. This annotated genes aligned to a reference genome. The data can be exported in multiple format (FASTA, GFF, EMBL, …).

## Exercises

**Exercises: What information about the rabies virus sequence can you obtain from its annotations in the NCBI Sequence Database? Give the accession number, definition, organism and PubMed ID of the record.**

* Go to Nucleotide database: "Rabies lyssavirus"[Organism] OR rabies virus[All Fields]) AND ("has refseqgene"[Properties] AND alive[prop]) 🡺 NC\_001542, def: rabies viurs, complete genoom, org: rabies lyssavirus, pubmed: 5 id

**Exercises: How many nucleotide sequences are there from the bacterium Chlamydia trachomatis?**

* Go to Nucleotide database: ("chlamydia trachomatis") AND "Chlamydia trachomatis"[porgn:\_\_txid813] 🡺 by tree indicate the org 🡺 91939

**Exercises: How many nucleotide sequences are there from the bacterium Chlamydia trachomatis in  the RefSeq part of the NCBI Sequence Database?**

* Go to Nucleotide database: ("chlamydia trachomatis") AND "Chlamydia trachomatis"[porgn:\_\_txid813] 🡺click on RefSeq 🡺 854

**Exercises: How many nucleotide sequences were submitted to NCBI by Matthew Berriman?**

* Go to Nucleotide database: "Beriman M." [all fields] 🡺 419221

**Exercises: How many nucleotide sequences from the nematode worms are there in the RefSeq Database?**

* Go to Nucleotide database: "nematod"[ORGN]🡺 RefSeq 🡺219039

**Exercises: How many nucleotide sequences for collagen genes from nematode worms are there in the NCBI Database?**

* Go to Nucleotide database: collagen AND txid6231[Organism:exp] (find in taxonomy browsers) 🡺 5286

**Exercises: How many mRNA sequences for collagen genes from nematode worms are there in the NCBI Database?**

* Go to Nucleotide database: collagen[All Fields] AND txid6231[Organism:exp] AND biomol\_mrna[PROP] 🡺 1277

**Exercises: How many protein sequences for collagen proteins from nematode worms are there in the NCBI database?**

* Go to protein database: collagen[All Fields] AND txid6231[Organism:exp] 🡺 7712

**Exercises: What is the accession number for the Trypanosoma cruzi genome in NCBI?**

* Go to genome database: Trypanosoma cruzi [orgn]  🡺 go to RefSeq 🡺 AccNr: Z\_AAHK00000000

**Exercises: How many fully sequenced nematode worm species are represented in the NCBI Genome database?**

* Go to genome database: Nematoda[ORGN] 🡺 93

**Exercises: Find the accession number of human beta-globin mRNA sequence. What is the accession number of the encoded protein ? How many amino acids does it contain?**

* Go to Nucleotide database: beta-globin[All Fields] AND "Homo sapiens"[porgn] AND biomol\_mrna[PROP]🡺 ACCESSION   NM\_000518 🡺RefSeq protein 🡺 147bp
* <https://github.com/avrilcoghlan/LittleBookofRBioinformatics/blob/master/src/chapter_answers.r>

**Exercises: Find the publication in PubMed with the following ID: 8663200. Use the search record of this publication in PubMed to obtain this data entry in the Nucleotide database. Download the GenBank formatted flatfile.**

* Fill in id in pubmed 🡺 go below to nucleotide 🡺download🡺 complete record🡺 file

**Exercises: Find the corresponding record from the previous exercise in the ENA database. Compare the EMBL format to the GenBank format you downloaded before.**

* ENA DB🡺 ACC in ENA / id 🡺 drosphila🡺 ctext file downloaden

**Exercises: How many alternative transcripts are know for Drosophila melanogaster eIF-4E.**

* Go to gene database: (eIF - 4E) AND "Drosophila melanogaster"[porgn:\_\_txid7227] 🡺 click on the first link 🡺 below the number of RefSeq are the alternative transcripts 🡺 9

**Exercises: Find the human hemoglobin alpha protein in UniprotKB. What is the entry name?**

* UniprotKB 🡺 HBA1 🡺 P69905

**Exercises: How many genes are associated with Huntington Disease (HD), with Alzheimer's disease (AD) and with Parkinson's disease (PD)?**

* Go to OMIM: give in Huntington Disease (HD) Alzheimer's disease (AD) and Parkinson's disease (PD)🡺 104300 ALZHEIMER DISEASE; AD, #143100. HUNTINGTON DISEASE; HD and #168600. PARKINSON DISEASE 🡺 1 (HD) 6 (AD) 6 (PD)

**Exercises: How many transcripts does Ensembl predict for the human gene ACHE?**

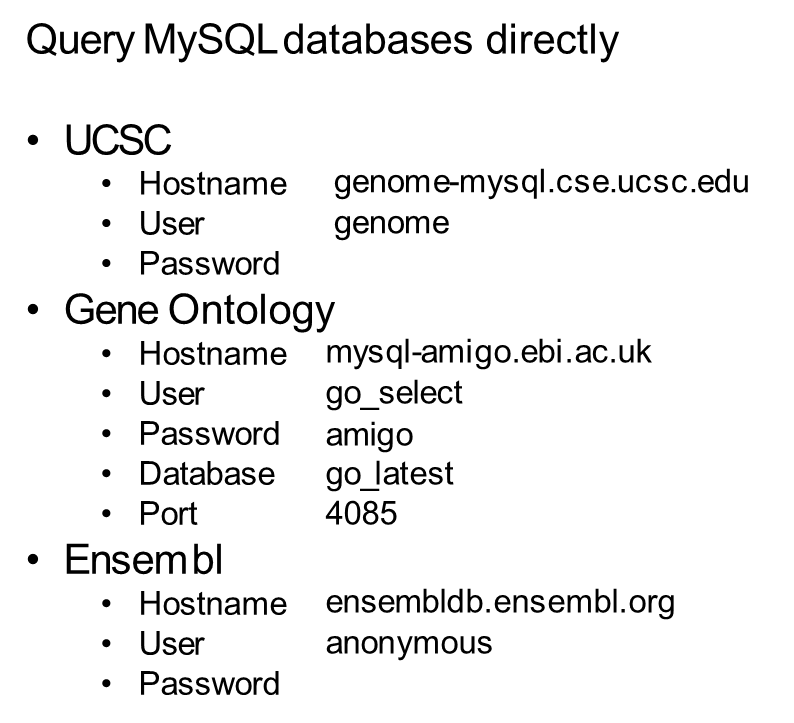
* Ensemble 🡺 ACHE 🡺 the gene has 14 transcripts

**Exercises: Find the mouse orthologue of the human SSBP4. Does this gene have paralogues?**

* Enslemble 🡺 SSB4 🡺 request orthologue left 🡺 idem paralogues

## Query MySQL databases directly

Use MySQL to connect to publicly-accessible MySQL servers. For example connect to Ensembl. Go to workbench, click on + , then give in Ensemble hostname and user as last click on OK. Ensembl exists of a complex database schemas and is in that way not suited to retrieve sequences.



## API

An application programming interface (API) is a set of subroutine definitions, protocols, and tools for building application software[[15]](#footnote-12). In general terms, it is a set of clearly defined methods of communication between various software components. It has an uniform method of accessing data.

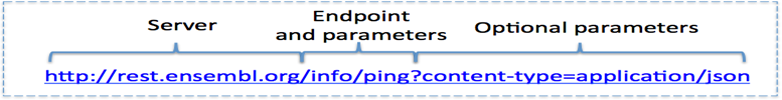
Ensembl uses MySQL relational databases to store its information. A comprehensive set of Application Program Interfaces (APIs) serve as a middle-layer between underlying database schemes and more specific application programmes. The APIs aim to encapsulate the database layout by providing efficient high-level access to data tables and isolate applications from data layout changes. Within Ensemble there are several databases.

* Core (genes, transcripts, translations, assembly, sequence): annotation information for each organism. It’s a species specific database.
* Compara (SNVs, CNVs, somatic variations, phenotypes)
  + Cross-species database
  + Genome-wide species comparisons
    - DNA-sequence level
      * Whole genome alignments
      * Sensitive regions
      * Conservation scores / constrained elements
    - Gene level
      * Phylogenetic trees
      * Homology predictions
* Variation (gene trees, homologies, multiple and pairwise genomic alignments)
  + Areas of the genome that differs between individual genomes
  + Associated disease and phenotype information
  + Different types of variants
    - Sequence variants
      * SNP (Single Nucleotide Polymorphism)
      * Insertion (one or more nucleotides)
      * Deletion (one or more nucleotides)
      * Indel (insertion and deletion, affecting 2 or more nucleotides)
      * Substitution (no change in length)
    - Structural variants
      * CNV (Copy Number Variation)
      * Inversion
      * Translocation
* Regulation (regulation, motifs, array probes): gene expression and its regulation in human and mouse. Focus on transcriptional and post-transcriptional mechanisms.

## REST API

Representational state transfer (REST) or RESTful web servers are a way of providing interoperability[[16]](#footnote-13) between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. The languages are JAVA, Perl, Python and Ruby.

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. The base URL used for all operations is formatted as follows: https://{server}/restapi/{apiVersion}/{accountId}/.



In the terminal curl is used. The **parameters** specify what is required and type of returned data from REST API. The different output formats are given below.

|  |  |
| --- | --- |
| * curl ‘<*url*>’   + [-H ‘<*header*>’]   + [-X <*request\_method*>]   + [-d ‘<*data*>’] | * JSON, FASTA, BED, XML, … * Depends on client and operation   + GET     - Content-type HTTP header     - Content-type HTTP parameter     - Accept HTTP header     - File extension   + POST     - Accept HTTP header |

Some examples of **endpoints**.

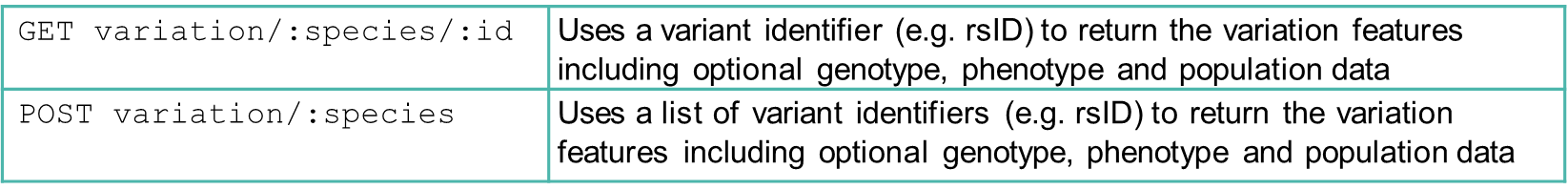
* Archive



* Comparative genomics



* Variation



* Sequence



## Exercises

**Exercises: Return the archived sequence with Ensembl id ENSG00000141510**

* Curl ‘http://rest.ensembl.org/archive/id/ ENSG00000141510' – H ‘Content-type:application/json’

**Exercises: Return the archived sequence for both ENSG00000012048 and ENSG00000136997**

* curl '<http://rest.ensembl.org/archive/id>' -H 'Content-type:application/json' \-H 'Accept:application/json' -X POST -d '{ "id" : ["ENSG00000141510", "*ENSG00000136997*"] }'

OR

* curl '[http://rest.ensembl.org/](http://rest.ensembl.org/archive/id)[archive/id](http://archive/id)' -H 'Content-type:application/json' -H 'Accept:application/json' -X POST –d '{"id":["*ENSG00000012048","ENSG00000136997"]}'*

**Exercises: Return a condensed XML-list of all orthologues in Mus musculus for ENSG00000159763. Do the same for BRCA2**

* Curl ‘http://rest.ensembl.org/homology/id/*ENSG00000159763?format=condensed;type=orthologues;target\_taxon=10090’ -H ‘Content-type:text/xml’*
* Curl’http://rest.ensembl.org/homology/symbol/human/BRCA2?format=condensed;type=orthologues;target\_taxon=10090’ -H ‘Content-type:text/xml’

**Exercises: Retrieve the genomic FASTA sequence for ENST00000288602.10**

* Curl’http://rest.ensembl.org/sequence/id/ENST00000288602.10?type=genomic’ -H ‘Content-type:text/x-fasta’

**Exercises: Get a sequence from 100 nucleotides located on human chromosome 2 starting at position 100000**

* Curl’http://rest.ensembl.org/sequence/region/human/2:100000..100099:1’ -H ‘Content-type:text/plain’

|1 stands for the order

**Exercises: Show the taxonomy information of the mouse**

* Curl’http://rest.ensembl.org/taxonomy/id/10090’ -H ‘Content-type:application/json’

**Exercises: Find the species and the database for ENSMUSG00000059552**

* Curl’http://rest.ensembl.org/lookup/id/ ENSMUSG00000059552?format=condensed’ -H ‘Content-type:text/xml’

**Exercises: Return the length of following chromosomes in human and mouse 2, 7 and X. Which are the longest?**

* curl 'http://rest.ensembl.org/info/assembly/homo\_sapiens/2'  -H 'Content-type:text/xml'
* curl 'http://rest.ensembl.org/info/assembly/homo\_sapiens/7'  -H 'Content-type:text/xml'
* curl 'http://rest.ensembl.org/info/assembly/homo\_sapiens/X'  -H 'Content-type:text/xml'
* curl 'http://rest.ensembl.org/info/assembly/mus\_musculus/2'  -H 'Content-type:text/xml'
* curl 'http://rest.ensembl.org/info/assembly/mus\_musculus/7'  -H 'Content-type:text/xml'
* curl 'http://rest.ensembl.org/info/assembly/mus\_musculus/X'  -H 'Content-type:text/xml'

## GIT – Track and store revisions/versions of files

The major difference between Git and any other VCS is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they keep as a set of files and the changes made to each file over time. Git thinks of its data more like a set of snapshots of a mini filesystem. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again—just a link to the previous identical file it has already stored. (<https://try.github.io/>).

* Help: $ git help [<git\_command>]
* Configuration: $ git config
* Initialize: $ git init
* Show status: $ git status
* Track files: $ git add <filename>
* Commit changes: $ git commit [-m “<commit\_message>”] and $ git reset
* Show logs: $ git log
* Checkout a commit: $ git checkout <checksum>
* Show differences between revisions: $ git diff [<checksum1> [<checksum2>]]
* Branching: $ git branch and $ git branch <‘new\_branch’>
* Merging: $ git merge <new\_branch> (Merge conflicts: same file modified on 2 seperate branches)
* Delete branch: $ git branch –d <new\_branch>
* Remotes: GitHub (public repositories)
* Clone repository: $ git clone <repository\_name> <local\_dir>
* Update repository: $ git pull <remote\_name> <branch\_name>
* Submit changes: $ git push <remote\_name> <branch\_name>

## Exercises

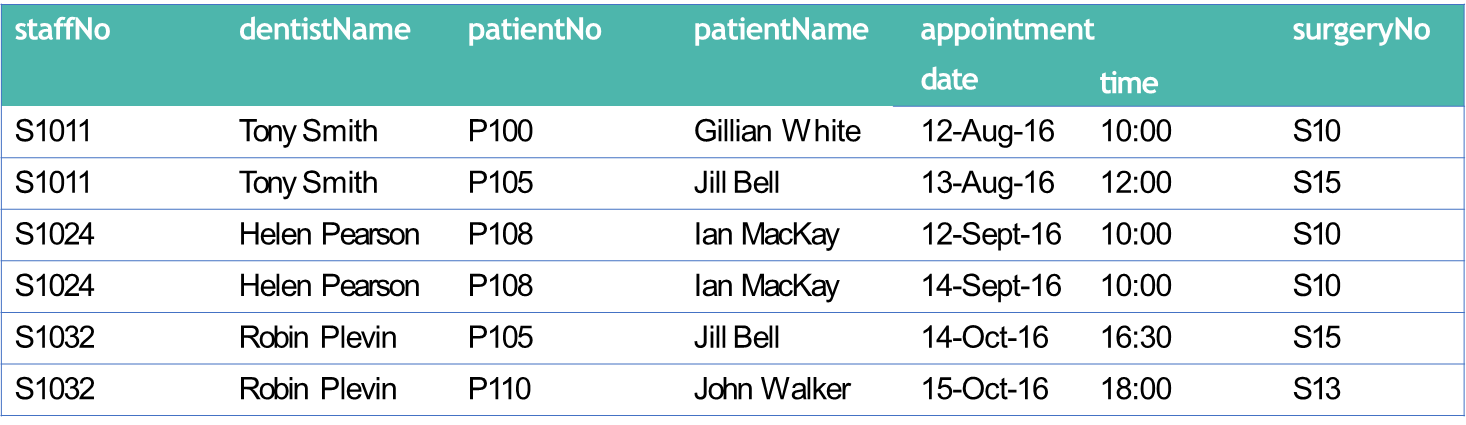
|  |  |
| --- | --- |
| Assignment | Solution |
| * Create a directory db\_git and copy some of the course files to this new directory. * Create a git repository in this directory   + Make sure your user\_name and user\_email are set correctly (HINT: git config) * Commit all .sql files and all other files with two different commit messages   + Check your commit history * Add a README file to your repository * Create a second branch in your repository   + Change to this new branch   + Add and delete some files   + Add some lines to your README file   + Show the differences between your 2 branches * Include the changes from your new branch into your original branch * Delete your second branch * Go to https://github.com/ and create a new repository *db\_github* * Add the contents of your existing *db\_git* repository to your newly created remote one. * Check the results | * Git init * Git config --global user.name “newuser” * Git config –global user.email [newuser@mail.com](mailto:newuser@mail.com) * Git add \*.sql * Git commit – m “add all \*.sql file” * Git log * Nano README (add some lines to the file) * Git add README * Git commit -m “add README” * Git branch new\_branch * Git checkout new\_branch * Git log * Git diff master new\_branch * Git checkout master * Git merge new\_branch * Git branch -d new\_branch (brings all changes made in new\_branch into the master(original) one) * Make an account * Add at the website by repository db\_github. * Got to the terminal * git remote add origin https://github.com/carolinevdb/db\_github.git git push -u origin master * Vul in username: carolinevdb * Vul in psswd: DwSg10+10 * <https://github.com/carolinevdb/db_github> |

# Extra exercises

**Exercise 1**

1. create a folder called exercises: $ mkdir exercises
2. copy some files in the folder
3. got to the exercises folder
4. Create a repository on github.com
5. Initialize the local directory as a Git repository: $ git init
6. add some file to you new local repository: $ git add \*.sql
7. Commit the files that you've staged in your local repository: $ git commit -m 'Add all \*.sql file to git'
8. add the URL for the remote repository where your local repository will be pushed. sets the new remote: $ git remote add exercises <https://github.com/carolinevdb/exercices>
9. Push the changes in your local repository to GitHub: $ git remote -v
10. Pushes the changes in your local repository up to the remote repository you specified as the origin: $ git push exercises master

**Exercise 2**



1) List the attributes of the entity, identify the main key - this must be unique, identify the repeating group of attributes and identify the key from the repeating group. In the 1NF we remove all the repeating groups (appointment), assigning new column (apptDate and apptTime) and assigned primary keys (candidate keys)

UNF DB(stafNo, dentistName, patienNo, patientName, appdate, apptime, surgeryNo)

2) As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. There are no partial dpendency

1NF DB(staffNo, dentistName, surgencyNo, patienNo, patienName)

App( appdate, apptime)

3) No non-prime attribute is dependent on the proper subset of any candidate key of table. There is no transitive dpendency

2NF App(staffNo, apptime, patientNo, patientName)

Surgery(staffNo, appDate, surgeryNo)

Dentist(staffNo, dentistName)

4) Transitive functional dependency(indirectly formed by two functional dependencies) of non-prime attribute on any super key should be removed.

3NF app(staffNo, appdate, aptime, patienNo)

surgery(staffNo, appdate, sugeryNo)

dentist(staffNo, dentistName)

patient(patientNo, patienName)

5) Any attribute on which some other attribute is full functionally dependent = determinant, every determinant is a candidate key.

BCNF 3NF

**Exercise 3:** The process of normalizing the following data to first (1NF), second (2NF), third (3NF) and BCNF.​

Patient No → Full Name

Ward No → Ward Name​

Drug No → Name, Description, Dosage,​Method of Admin​

Patient No, Drug No, Start Date → Units per Day, Finish date

1NF form(patientNo, patientName, BedNo, WardNo, WardName, DrugNo,DrugName, DrugDescription, DrugDosage, DrugAdm, UnitDay, StartDate, FinishDate)

2NF patient(patientNo, Patientname)

form(Patientno, BedNo, WardNo, WardName, DrugNo,StartDate, FinishDate, UnitDay)

drug(DrugNo, DrugName, DrugDescription, DrugDosage,DrugAdm)

3NF patient(patientNo, Patientname)

form(Patientno, BedNo, WardNo, DrugNo,StartDate, FinishDate, UnitDay)

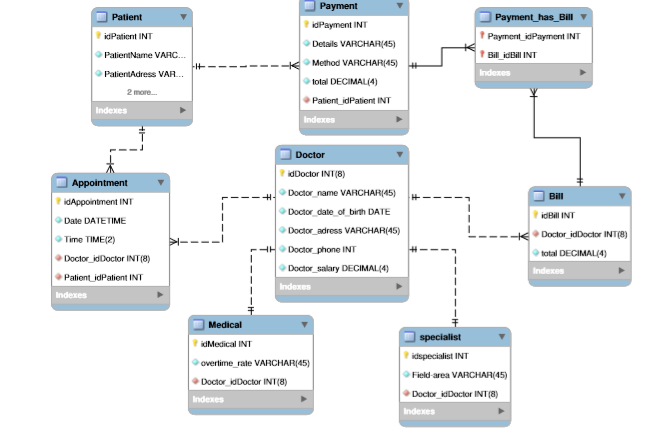
drug(DrugNo, DrugName, DrugDescription, DrugDosage,DrugAdm)

ward(wardNo, wardName)

BCNF 3NF

**Exercise 4:**

|  |  |
| --- | --- |
| * Create an ERD schema with 3 tables   + Doctor     - Name     - Date\_of\_birth     - Address     - phone\_number     - Salary   + Medical     - Overtime\_rate   + Specialist     - Field\_area | * Add 2 tables to your schema   + Patient     - Name     - Address     - Phone\_numer     - Date\_of\_birth   + Appointment     - Date     - Time * Think carefully how you will integrate this with your existing tables |
| * Add 2 tables to your schema   + Payment     - Details     - Method   + Bill     - Total | * + Careful: there is a many-to-many relation between bill and payment   + Save your .mwb file and add it to your repository   + Forward enigneer your model |



**Exercise 5:** Create a new database called vacation​

1) Log in with root user: $ mysql -uroot -p

2) create a new database called vacation: myql> CREATE database vacation;

3) Grand all acces to newuser@localhost: myql> GRANT ALL ON vacation.\* TO newuser@localhost;

4) import the DB in the newly created DB: $ mysql vacation < vacation.sql

5) see the result: myql> SHOW DATABASES;

Exercises:

1) List the names and addresses of all guests in London, alphabetically ordered by name.​

SELECT guestname, guestaddress FROM guest WHERE guestaddress LIKE '%London%' ORDER BY guestname;

2) List all double or family rooms with a price below £40.00 per night, cheapest first.​

select type, price FROM room WHERE price < 40.00 AND (type='double' OR type='family') ORDER BY price ASC;

OR

SELECT \* FROM room WHERE price < 40 AND type IN ('Double', 'Family') ORDER BY price:

3) List the bookings for which no date\_to has been specified.​

SELECT \* FROM booking WHERE dateto IS NULL;

4) What is the average price of a room?​

SELECT AVG(price) AS average\_price FROM room;

5) What is the total revenue per night from all double rooms?​

SELECT SUM(price) AS total\_revenue FROM room WHERE type='double';

6) How many different guests have made bookings for August?

SELECT COUNT(DISTINCT(guestNo)) from booking WHERE (datefrom>='2004-08-01' AND datefrom <='2004-08-31') OR (dateto >='2004-08-01' AND dateto<='2004-08-31');

OR

SELECT COUNT(DISTINCT guestno) FROM booking WHERE (datefrom >= '2004-08-01' AND datefrom <= '2004-08-31');

7) List the price and type of all rooms at the The Tower Hotel.​

SELECT price, type FROM room JOIN hotel ON room.hotelno = hotel.hotelno WHERE hotelname = 'The Tower Hotel';

8) List all guests currently staying at the The Tower Hotel.​

SELECT \* FROM guest JOIN booking ON guest.guestno = booking.guestno JOIN hotel on booking.hotelno = hotel.hotelno WHERE datefrom <= CURRENT\_DATE AND dateto >= CURRENT\_DATE AND hotelname = 'The Tower Hotel';

9) List the details of all rooms at the The Tower Hotel, including the name of the guest staying in the room, if the room is occupied.​

SELECT \* FROM room JOIN booking on room.hotelno = booking.hotelno JOIN hotel ON hotel.hotelno = booking.hotelno WHERE hotelname = "The Tower Hotel" AND dateto IS NOT NULL;

10) What is the total income from bookings for the The Tower Hotel today?​

SELECT SUM(price) FROM booking, room, hotel WHERE room.hotelno=hotel.hotelno AND room.roomno=booking.roomno AND hotelname='The Tower Hotel';

OR

SELECT SUM(price) FROM room JOIN booking ON room.hotelno = booking.hotelno JOIN hotel ON booking.hotelno = hotel.hotelno WHERE hotelname = "The Tower Hotel" AND datefrom <= CURRENT\_DATE AND dateto >= CURRENT\_DATE GROUP BY hotelname;

11) List the rooms that are currently unoccupied at the The Tower Hotel.​

SELECT \* FROM room JOIN booking ON room.hotelno = booking.hotelno JOIN hotel ON booking.hotelno = hotel.hotelno WHERE hotelname = "The Tower Hotel" AND (datefrom > CURRENT\_DATE OR dateto < CURRENT\_DATE);

12) What is the lost income from unoccupied rooms at the The Tower Hotel?​

SELECT SUM(price) FROM room r, hotel h WHERE r.hotelno=h.hotelno AND h.hotelname='The Tower Hotel' AND roomno NOT IN (SELECT roomno FROM booking b, hotel h where (datefrom <= curdate() AND dateto >=curdate()) AND b.hotelno=h.hotelno AND r.roomno=b.roomno and h.hotelname='The Tower Hotel');

OR

SELECT SUM(price) FROM room JOIN booking on room.hotelno = booking.hotelno JOIN hotel ON hotel.hotelno = booking.hotelno WHERE hotelname = "The Tower Hotel" AND (datefrom > CURRENT\_DATE OR dateto < CURRENT\_DATE) GROUP BY hotelname;

13) List the number of rooms in each hotel in London.​

SELECT hotel.hotelno, COUNT(roomno) AS count FROM hotel, room WHERE room.hotelno = hotel.hotelno AND city LIKE '%London%' GROUP BY hotelno;

14) What is the most commonly booked room type for each hotel in London?​

SELECT hotelname, type, COUNT(type) AS count FROM booking b, hotel h, room r WHERE r.roomno = b.roomno AND b.hotelno = h.hotelno AND h.city LIKE '%London%' GROUP BY type, hotelname ORDER BY hotelname, count DESC;

15) Update the price of all rooms by 5%.

update room set price=price\*1.05;

**Exercice 6:**

1) $ mysql -uroot -p

2) mysql> create database college;

3) mysql> grant all on college.\* to newuser@localhost;

4) mysql> exit

5) $ mysql college < college.sql

6) $ mysql

Exercices:

​1) Retrieve the name, city and grade point average (GPA) of students, greater than or equal to 3.7​

SELECT stdFirstName, stdCity, stdGPA FROM student WHERE stdGPA >3.7;

2) List the name and hiring date of faculty hired in 1999 or 2000​

SELECT concat(FacFirstName,' ',FacLastName) AS facultyname, FacHireDate FROM faculty WHERE FacHireDate BETWEEN '1999-01-01' AND '2000-12-31';

3) List the offer number, course number and faculty Social Security number for course offerings scheduled in fall 2005 or winter 2006​

SELECT OfferNo, courseNo, facssn FROM offering WHERE (offterm='fall'AND offyear=2005) OR (offterm='winter'AND offyear=2006);

4) List the offering number, course number, days and time of offerings containing the words 'database' or 'programming' in the course description and taught in spring 2006.

SELECT offerno, offering.courseno, offterm, offtime, offdays, course.crsDesc from offering, course WHERE offterm='spring' and offyear=2006 and offering.courseno=course.courseno and (course.crsdesc like '%database%' OR course.crsdesc like '%programming%');

5) List the offer number, course number and name of the instructor of IS course offerings scheduled in fall 2005 taught by assistant professors​

SELECT offerno, offering.courseno, faculty.facfirstname, faculty.faclastname from offering, faculty WHERE offering.offterm='fall' AND offering.offyear=2005 AND offering.Facssn = faculty.facssn and facrank='asst' and offering.courseno like "IS%";

6) Summarize the averageGPA of students by major.​

SELECT stdmajor, AVG(stdGPA) as averageGPA from student group by stdmajor;

7) Summarize the number of offerings and unique courses by year.​

SELECT offyear, COUNT(\*) AS NrofOfferings, COUNT(DISTINCT Courseno) as NrofCourses FROM offering group by offyear;

8) Summarize the average GPA of upper-division (junior or senior) students by major. Only list the major with average GPA greater than 3.1.​

SELECT stdmajor, avg(stdgpa) as avg from student where stdclass='JR' OR stdclass='SR' group by stdmajor having avg>3.1;

OR

SELECT StdMajor, AVG(StdGPA) AS AvgGpa FROM Student WHERE StdClass IN ('JR', 'SR') GROUP BY StdMajor HAVING AVG(StdGPA) > 3.1;

9) Summarize the minimum and maximum GPA of students by major and class.​

SELECT min(stdgpa), max(stdgpa) from student group by stdmajor, stdclass;

10) Summarize the number of IS course offering by course description.​

SELECT crsdesc, count(\*) as offeringnr from course, offering where course.courseno=offering.courseno and offering.courseno like 'IS%' GROUP by crsdesc;

11) List the GPA, name, city and state of juniors, lowest GPA first.​

SELECT stdgpa, concat(stdfirstname,' ',stdlastname) as studentname, stdcity from student where stdclass='JR' order by stdgpa DESC;

12) List the unique city and state combination in the Faculty table.​

SELECT DISTINCT(faccity), facstate from faculty;

13) List the student name, offering number and grade of students who have a grade >=​3.5 in a course offering.​

SELECT concat(stdfirstname,' ',stdlastname) as Student\_Name, stdGPA from student, enrollment where Enrgrade >=3.5 and student.stdSSN=enrollment.stdSSN;

OR

SELECT StdFirstName, StdLastName, OfferNo, EnrGrade FROM Student JOIN Enrollment ON Student.StdSSN = Enrollment.StdSSN WHERE EnrGrade >= 3.5;

14) List the names of students (without duplicates) who have grades >= 3.5 in a course offering.​

SELECT distinct stdfirstname,stdlastname, stdGPA from student, enrollment where Enrgrade >=3.5 and student.stdSSN=enrollment.stdSSN;

OR

SELECT DISTINCT StdFirstName, StdLastName FROM Student JOIN Enrollment ON Student.StdSSN = Enrollment.StdSSN WHERE EnrGrade >= 3.5;

15) List Bob Norbert's course schedule in spring 2006. For each course, list the offering number, course number, days, location, time and faculty name.

SELECT offering.OfferNo, offering.CourseNo, OffDays, OffLocation, OffTime, FacFirstName, FacLastName FROM faculty, offering, enrollment, student WHERE offering.OfferNo = enrollment.OfferNo AND student.StdSSN = enrollment.StdSSN AND faculty.FacSSN = offering.FacSSN AND OffYear = '2006' AND OffTerm = 'SPRING' AND StdFirstName = 'BOB' AND StdLastName = 'NORBERT';

OR

SELECT offering.OfferNo, offering.CourseNo, OffDays, OffLocation, OffTime, FacFirstName, FacLastName FROM faculty JOIN offering ON faculty.FacSSN = offering.FacSSN JOIN enrollment ON offering.OfferNo = enrollment.OfferNo JOIN student ON student.StdSSN = enrollment.StdSSN WHERE OffYear = '2006' AND OffTerm = 'SPRING' AND StdFirstName = 'BOB' AND StdLastName = 'NORBERT';

**Exercise 7:**

$ mongoimport --db test --collection restaurants --drop --file restaurants.json

$ mongo

> db.restaurants.find()

Exercise:

1) Display the fields restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.​

> db.restaurants.find({},{"restaurant\_id":1,"name":1,"borough":1, "cuisine":1})

2) Display the second group of 5 restaurants which are in the borough Bronx.​

> db.restaurants.find({"borough": "Bronx"}).skip(5).limit(5)

3) Find the restaurants who achieved a score more than 90.​

> db.restaurants.find({grades:{$elemMatch:{"score":{$gt:90}}}})

4) Find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.​

> db.restaurants.find({$and:[{"cuisine" :{$ne:"American"}},{"grades.score": {$gt:70}}, {"address.coord":{$lt:-65.754168}}]})

5) Find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.​

> db.restaurants.find({"borough": "Bronx", $or:[{"cuisine": "American"},{"cuisine":"Chinese"}]})

6) Find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronxor Brooklyn.​

> db.restaurants.find({"borough":{$nin:["Staten Island","Queens", "Bronx","Brooklyn"]}}, {"restaurant\_id":1,”name":1,"borough":1,"cuisine":1})

7) Find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.

> db.restaurants.find({"grades.score":{$not:{$gt:10}}},{"restaurant\_id":1,"name":1, "borough":1,"cuisine":1})

8) Find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates.​

> db.restaurants.find({"grades.date": ISODate("2014-08-11T00:00:00Z"),"grades.grade" :"A","grades.score":11},{"restaurant\_id":1,"name":1,"grades":1})

9) Give a list of all restaurants, ordered by cuisine alphabetically and borough in reverse alphabetical order​

> db.restaurants.find().sort({"cuisine":1,"borough":-1})

10) Write a query to know whether all the addresses contain a street or not.

> db.restaurants.find({"address.street":{$exists:true}})

**Exercise 8:**

1) Get the sequence for the region from basepair 32889000 to 32891000 of human chromosome 13 in FASTA format. Hardmask and softmask the sequence. How many repeat regions are there in this sequence?​

curl 'http://rest.ensembl.org/sequence/region/human/13:32889000..32891000:1' -H 'Content-type:text/x-fasta'

curl 'http://rest.ensembl.org/sequence/region/human/13:32889000..32891000:1?mask=soft' -H 'Content-type:text/x-fasta'

curl 'http://rest.ensembl.org/sequence/region/human/13:32889000..32891000:1?mask=hard' -H 'Content-type:text/x-fasta'

* + - * + 3 repeat regions

2) Get the Ensembl Gene ID for the human CCR5 (chemokine (CC motif) receptor 5) gene.​

curl 'http://rest.ensembl.org/xrefs/symbol/homo\_sapiens/CCR5' -H 'Content-type:application/json'

[{"type":"gene","id":"ENSG00000160791"}]

3) Has an orthologue for this gene been identified in chimpanzee?​

curl 'http://rest.ensembl.org/xrefs/symbol/homo\_sapiens/CCR5?target\_species=pan\_troglodytes' -H 'Content-type:application/json'

* + - * + ENSP00000292303 (homos sapien) and ENSPTRP00000025599 (pan\_troglodytes) zijn de orthologue

4) Retrieve the genomic FASTA sequence for ENST00000001146​

curl 'http://rest.ensembl.org/sequence/id/ENST00000001146​?type=genomic' -H 'Content-type:text/x-fasta'

5) Return the archived sequence for both ENSG00000002587 and​ ENSG00000003056​

curl 'http://rest.ensembl.org/archive/id' -H 'Content-type:application/json' -H 'Accept:applicatid": ["ENSG00000002587", "ENSG00000003056​"]}'

6) Get a sequence from 250 nucleotides located on human chromosome 7 starting at position 35182

curl 'http://rest.ensembl.org/sequence/region/human/7:35182..35431:1?' -H 'Content-type:text/plain'

1. Relation(id, name, adress): id is a prime attribute and name & adress is a non-prime attribute [↑](#footnote-ref-1)
2. Following condition (1. A->B, 2. B->A not, 3. B->C) then is A->C transitive dependency [↑](#footnote-ref-2)
3. hashing is the transformation of a string of [character](http://searchcio-midmarket.techtarget.com/definition/character)s into a usually shorter fixed-length value or key that represents the original string. [↑](#footnote-ref-3)
4. [data](https://en.wikipedia.org/wiki/Data) that provides information about other data [↑](#footnote-ref-14274)
5. an increase or addition, especially one of a series on a fixed scale. [↑](#footnote-ref-4)
6. a statistical data type consisting of numerical scores that exist on an arbitrary numerical scale [↑](#footnote-ref-5)
7. wrong [↑](#footnote-ref-6)
8. the action or process of making something stronger or more solid. [↑](#footnote-ref-7)
9. consist of; be made up of. [↑](#footnote-ref-8)
10. add notes to (a text or diagram) giving explanation or comment. [↑](#footnote-ref-9)
11. select, organize, and look after the items in (a collection or exhibition [↑](#footnote-ref-10)
12. Mendelian Inheritance in Man [↑](#footnote-ref-9728)
13. determine the limits or boundaries of [↑](#footnote-ref-11)
14. Universiteit van Californië - Santa Cruz [↑](#footnote-ref-2556)
15. is a computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user. [↑](#footnote-ref-12)
16. the ability of computer systems or software to exchange and make use of information. [↑](#footnote-ref-13)