Data & Databases

DBW

Outline

- Data modelling and Databases
  - Databases. Types. SQL vs noSQL
- Data models
- Database design
- ETLs
- Interaction with Web apps
- MySQL & SQL Language

What is data modelling? What is a Database?

- All applications manage data
- Simple data can be managed with primitive data types and simple arrays (dictionaries, …)
- Complex data require to design a data model
- Data models provide Classes in object-oriented programming and are the basis for designing database structures.

- Collection of data organized and stored according to some purpose.
  - Pile of papers, Flat text file, indexed store,…
- Ideally, data is organized following a specific data model
- Provide permanent storage for data structures
- DBMS (Database management software) takes care of storing and retrieving data
  - MySQL, PostgreSQL, SQLite, Oracle, Access, MongoDB, …
- Types
  - Relational DB, Column DB, Document DB, XML DB, …

Databases & Web Applications

- Databases in Web appls. are used for
  - Storing Data and Meta-Data
  - Managing user/session credentials

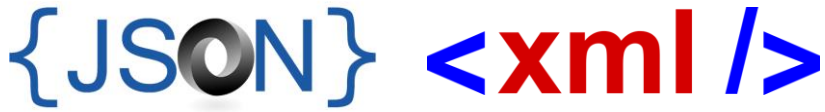
- Databases always need an access application
  - Databases can be accessed directly but this not practical for end users (permanent conections, not enough expertise)
  - Most usual way is a **REST API** (a web service)

REST-ful APIs (quick remind)

Web services to serve “resources” (data) using only HTTP (GET, POST, PUT)

/api/{store}/{id}/option.format?options

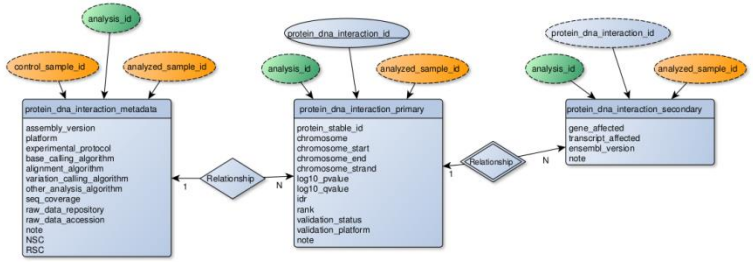
/api/pdb/2ki5/entry



SQL *vs* noSQL

Oracle, **MySQL, PostgreSQL,…**

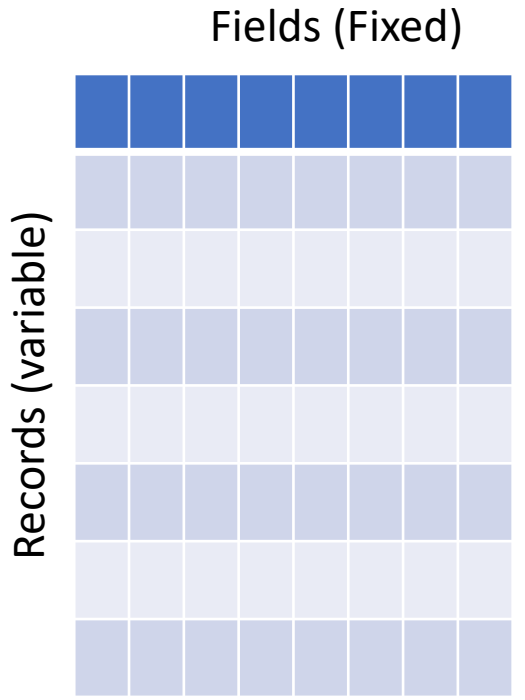
- Poorer scaling abilities
- A.C.I.D. (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
- More difficult design. Fixed structure.
- Do not map transparently on object-oriented data
- Libraries everywhere



Google BigTable, **MongoDB**, Hbase**, …**

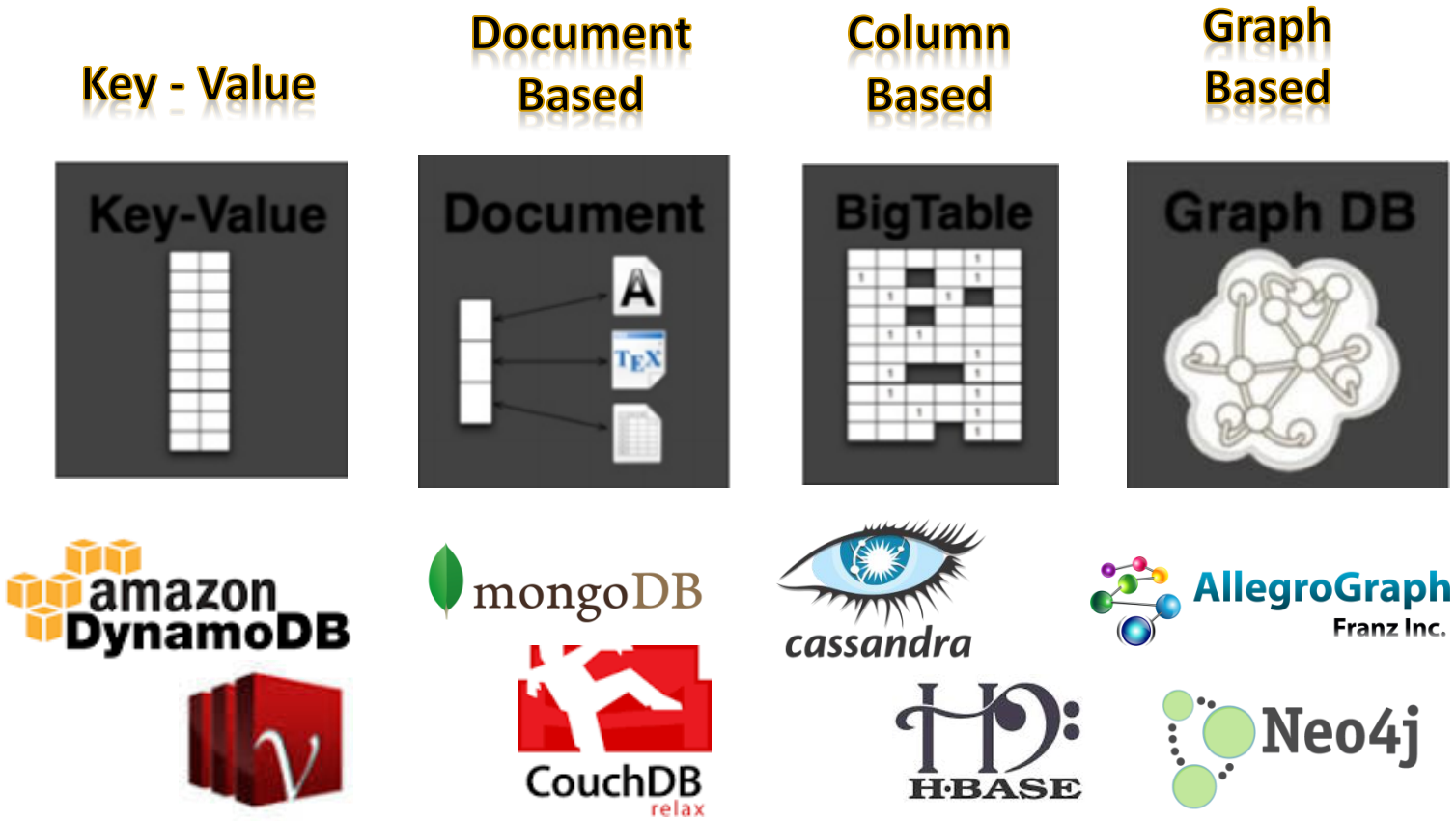
- Great scalability, but require larger resources
- B.A.S.E. (**B**asically **A**vailable, **S**oft state, **E**ventually consistent)
- Map complex data structures directly. No additional design
- Align better with “modern” data representations (JSON or XML)
- Libraries everywhere

Relational databases (SQL)



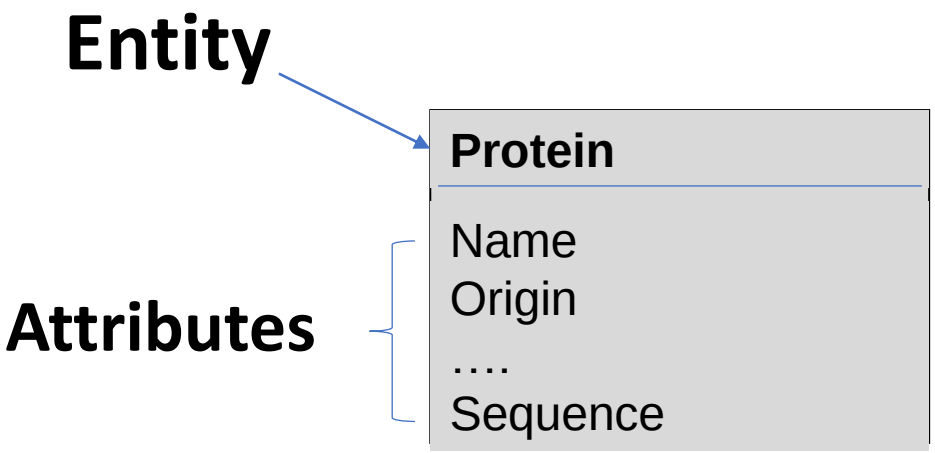
- Most used in general, and especially in bioinformatics
  - This is changing, however…
- Data organized in “tables”
  - Tables contain a number of “records” (rows)
  - Each record has a number of “fields” (columns)
- “Relational” means that logical relationships could be established between fields on different tables.
  - DB manager uses those relationships to build complex queries
- Efficiency on data management depends on a “correct” DB design.

NoSQL Databases



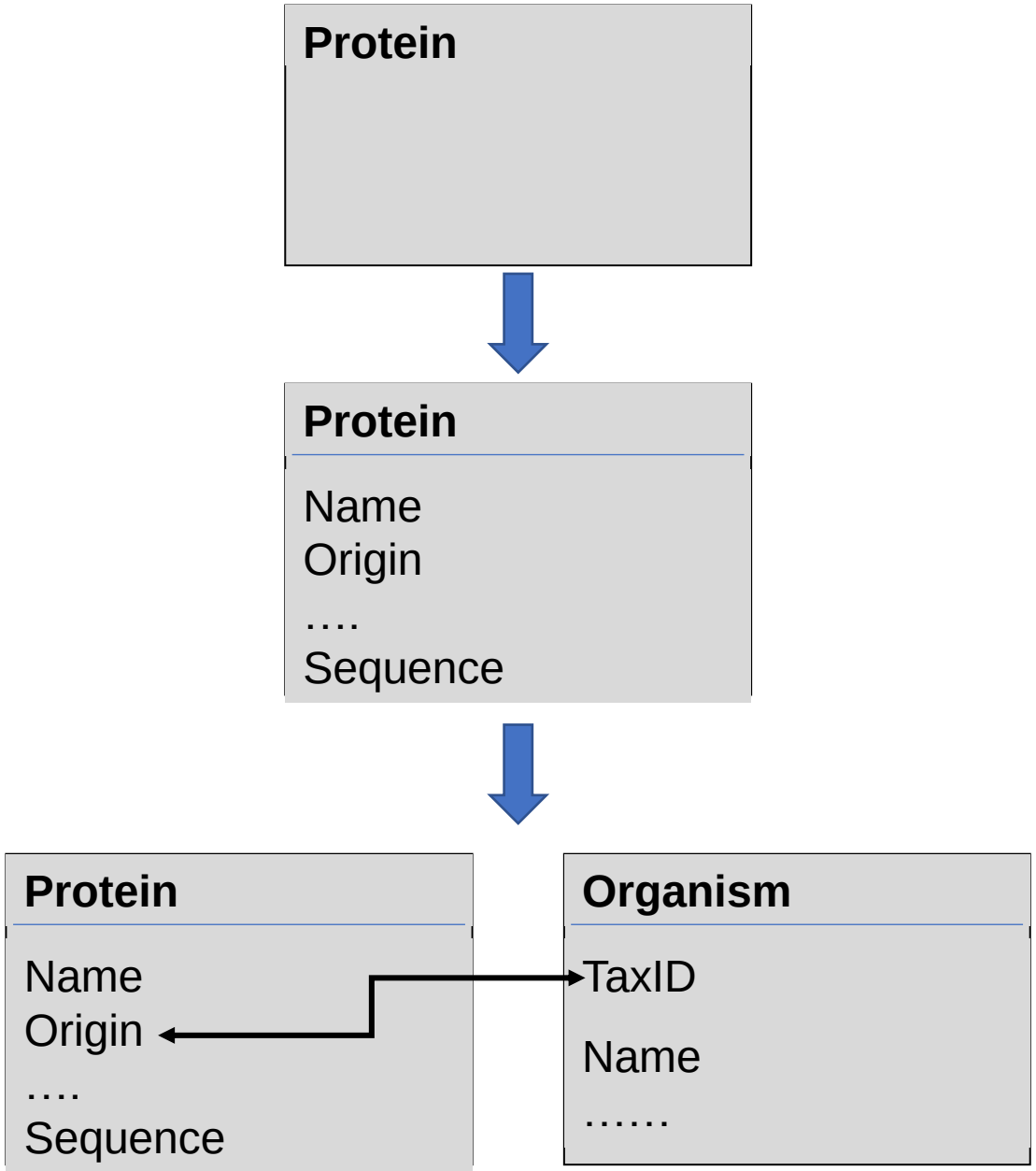
Data modelling

- **Aim:** Define the structure of data types and components to be managed by the application
- First step in designing any application
- Data entities: everything that should be stored/managed
- Entity attributes: information about the entity



Data model building

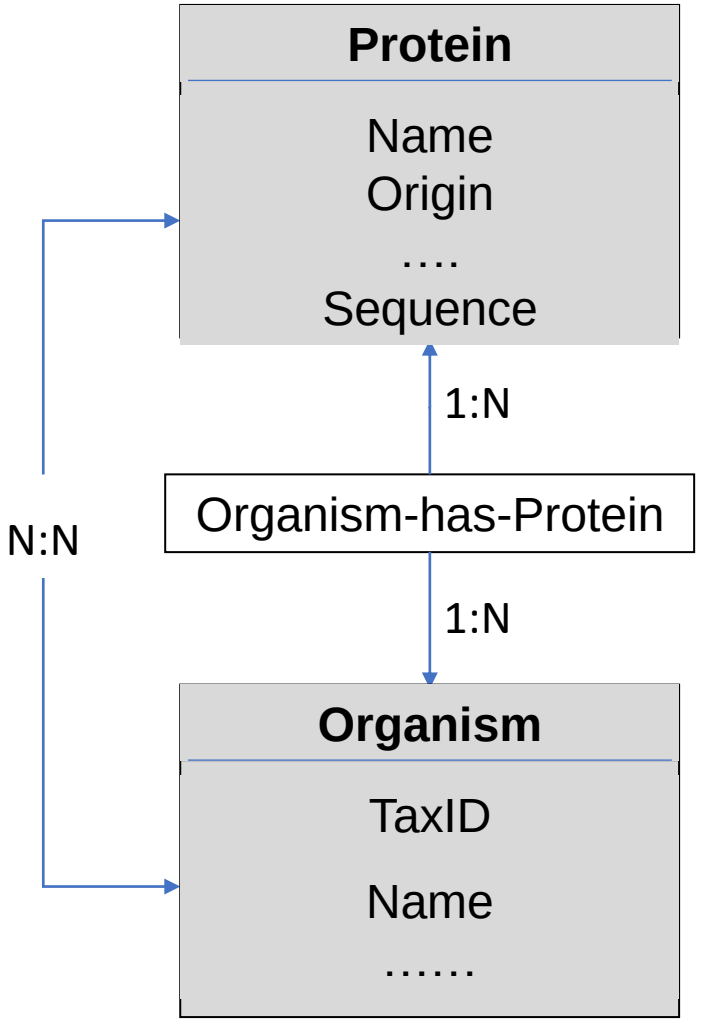
- Identify data entities
  - Data items that “exist” by themselves
- Decide on data attributes
  - Details of every data entity
- Identify data relationships
  - Which attributes relate data entities
- If a Database is involved
  - Define unique identifiers (always useful)
  - Normalize (more later)



Relationships

- Associations between entities
  - Relational DBs include explicit keys
  - O-Oriented DBs and languages often “denormalize” including nested objects

- **1:1** Rare, entities should be merged (common primary key)
  - May be necessary to improve efficiency
- **1:N** most common
  - The “N” classe includes “1” primary key as attribute
- **N:N** A new “hidden” entity exists.
  - The new entity is 1:N to the original entities. Add attributes as necessary.



NoSQL databases do not handle (in general) relationship, but the concept should be considered in the design

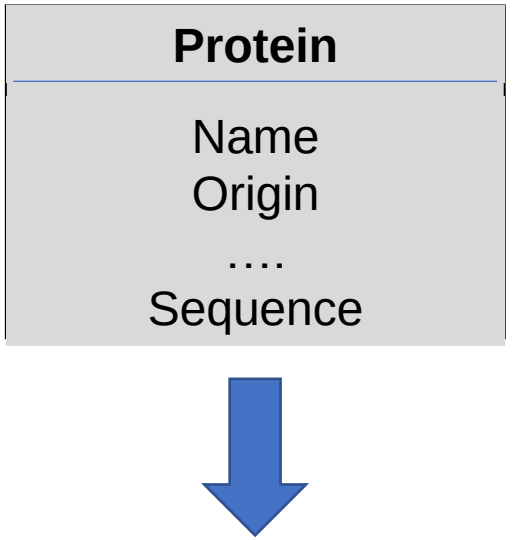
Database (SQL) design phylosophy

- Structure of data should be
  - Compact with minimum redundancies
    - Data stored only once (consistency)
    - Space saving
  - Structure oriented to retrieval
    - Most Bioinformatics DBs are store once, retrieve many
    - Obtaining data quick is required
  - Able to grow
    - Data evolves, structure should be flexible
- Relational DBs requires known and fixed data structures
- For unforeseen data structures, use noSQL approach!!

DB design

- Depends on the language/Database type
- Traditional Relational Databases
  - Saving space and avoiding redundancies is the main issue
- NoSQL databases / O-O Programming
  - Space is not an issue, data can be redundant (but consistent), efficiency in insertion/retrieval is the main issue

DB design



- Entities become classes, tables, collections, …
- Attributes become fields (Columns in tables)
- Unique identifiers become primary keys
  - not NULL, never changes
  - Unique identification of a record
  - Can be a combination of several fields

ID	Name	Origin	Sequence
P9WKE1	Thymidylate kinase	8332	MLIAIEGVDGAG KRTLVE…
P04183	Thymidine kinase, cytosolic	9606	MSCINLPTVLPGS PSK…
…	…	…	…

- In SQL DBs Relationships become “foreign keys”
- Keys are usually integers (often with auto-increment), although can be any field.

Normalization of Relational DBs

- Rules to Reduce (eliminate) data redundancies
  - Avoids inconsistencies
  - Allows non-complete insertions or deletions
  - Make easier queries

- 1st Normal Form (1NF)
  - Unique identifiers. Records are independent to each other. All attributes have single values. *Lists of values show hidden entities*

- 2nd Normal Form (2NF)
  - All attributes depend entirely on the entity. *Attribute is misplaced or a new entity*

- 3rd Normal Form (3NF)
  - Data attributes are independent to each other. *Show hidden entitites.*

ETLs

Extract, Transform, & Load

- Software **designed to populate DBs** from the original data sources
- Normally **offline** command-line scripts
- Typically, **scripting languages** (Perl, Python)
- Data is usually obtained from text files or from Web Services

Extract:

- Parsing data input

Transform:

- Do the necessary modifications on the data
- Add new “calculated” fields if necessary

Load

- Insert into the DB

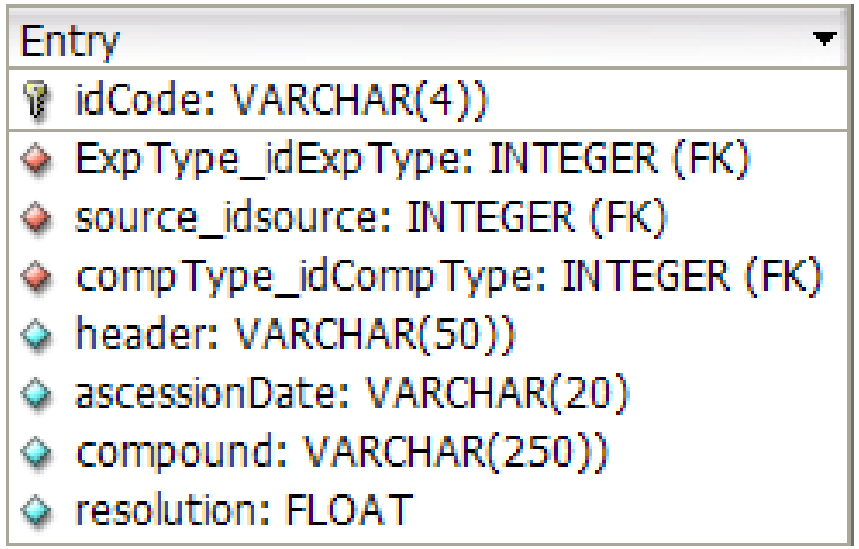
From Web apps

- Server side
  - All Server-side languages include **specific drivers and helpers**
  - The usual ones issues **database commands** (SQL, JSON, …)
    - $result = $db -> mysql_query(“SELECT ^ FROM foo”);
    - $result = $foo_collection->find(array(‘_id’ => ‘any_id’));
  - More elaborated drivers **map DB tables/objects** into program objects
    - Interaction with DB is made in the background
    - Common in pure o-o languages and programming frameworks
  - **DB connections are persistent** .
    - Connection is usually made once at the initialization phase for each script.
- Client side
  - Jquery / AJAX may include direct DB connections (not recommended)
  - Use API's (recommended)

MySQL

- Created in 1979 by Michael Widenius
- MySQL 1.0 in 1995
- Uses SQL as query language
- Used in most bioinformatics applications
  - Free, easy to install
  - Now (v ≥ 5.x) has most features of a commercial DBMS
- MariaDB is an open source replacement (no differences)
- Drivers
  - PHP: mysqli
  - Python: mysql.connector, pymysql, mysqldb, …

Create table example (use helper software)




```
CREATE TABLE Entry (
  idCode VARCHAR(4)) NOT NULL,
  ExpType_idExpType INTEGER UNSIGNED NOT
  NULL,
  source_idsource INTEGER UNSIGNED NOT NULL,
  compType_idCompType INTEGER UNSIGNED NOT
  NULL,
  header VARCHAR(50)) NULL,
  ascessionDate VARCHAR(20) NULL,
  compound VARCHAR(250)) NULL,
  resolution FLOAT NULL,
  PRIMARY KEY(idCode),
  INDEX Entry_FKIndex1(compType_idCompType),
  INDEX Entry_FKIndex3(source_idsource),
  INDEX Entry_FKIndex4(ExpType_idExpType)
);
```

MySQL (usual) data types

- Numeric
  - Integer
    - Used for most keys!!
  - Float (M,D)
- Text
  - **varchar(n)**
  - varbinary(n)
  - **text(n)**
  - blob(n)
  - enum (one of ‘val1’, ‘val2’,…)
  - set (any of ‘val1’, ‘val2’,…)
  - Careful with character sets!!

- Date/time
  - Date yyyy-mm-dd
  - Datetime yyyy-mm-dd hh:mm:ss
  - Timestamp
  - Time hh:mm:ss
  - Year (2|4)

  - Be careful with order, can depend on O.S.!!
  - Safe alternative use strings like
    YYYY-MM-DD:HH-MM

- Data initialization options
  - Auto-increment (automatic key fields)
  - DEFAULT constant (used if no input)
  - NOT NULL (error if empty)

Basic SQL

- Table manipulation
  - CREATE TABLE, ALTER TABLE, DROP TABLE, RENAME TABLE, CREATE INDEX, DROP INDEX
  - Usually done with helper software (Mysql Workbench, PhpMyAdmin)

- Storing data
  - INSERT INTO table (col1, col2,…) VALUES (val1,val2,…)
  - LOAD DATA INFILE ‘file_name’
  - REPLACE
    - Like INSERT but replaces rows with the same primary key
  - UPDATE table SET col1=val1, coln=valn WHERE ‘some_condition’

- Retrieving data
  - SELECT col1, …. FROM table1, table2,… WHERE ‘some condition’ ORDER BY col