S1 Review

# SQL

### 1/ Intro

**Interesting stuff :**

* SQL centers around limited programming language ⇒ succinct programming but query optimization problems
* Almost behind everything we do on the web
* Unique concurrency-control problems

**DBMS + :**

* Large data sets
* Complexe structure
* Many users access ⇒ access control, DBMS manages privileges and roles + restricts views

**Data model :**

* Describes complex data structure & relationships
* Abstract view of the data
* **Data manipulation Language (DML)** → express complex queries & updates over data
* **Data definition Language (DLL)** → describes data & integrity constraints.
* Enforcing integrity constraints each update ⇒ database consistency

**Crash recovery** ⇒ DB protecting users from effects of sys failure

**History :**

* **First =** rudimentary
* **Early 60 :** network data model ( data = graphs + CODASYL)
* **Late 60 :** hierarchical data model ( data = trees )

(trees/graphs also called semistructured models)

* **70 :** definition of relational data model
  + ⇒ logical relations = tables = sets
  + ⇒ tables linked by keys
* **Today :** XML & object-based

### 2/ relational model

**Relational algebra =** queries translated to operations on sets

**Data model =** mathematical representation of data + operations on data + constraitns

**Database =** collection of relations

**Database schema =** set of all relation schemas in the database

**Database instance =** set of all relation instances in the table (le contenu de toutes les tables à un instant t)

**Relation =** table

* A column header/name = attribute
* A row = tuple

**Relations + :**

* Simple model
* Matches our data conception
* Abstract model good for SQL

**Relation schema =** relation name & attributes list

* Attribute types are optional in this one
* Eg : Beers (name, manf)
* You have to underline the primary key
* # in front of foreign key

**Relation instance =** set of tuples that the relation currently holds. (en gros ce qu’il y a dans la table à un instant t).

**SQL =** query language, get information from a database. DML (get the data from DB) + DLL (describe the db schemas)

**CHAR(n) =** fixed lenght string of n characters

=/= **VARCHAR(n) =** variable length string up to n characters

* Don’t forget the single quotes for strings (don’t need any for int & float)
* ‘Joe’’sbar’ ⇒ ‘’ = 1 quote

**DATE =** ‘yyyy-mm-dd’

**TIME** = ‘hh:mm:ss’

**DATETIME =** DATE + TIME

**BLOB (Binary Large Object)=** content of some file (song, pic, movie…)

**CLOB (Character LOB) =** character file (letter, contract….)

**NULL =** Missing values or inapplicable (ex : spouse for an unmarried person)

⇒ pas confondre avec 0 ou “” qui sont des values :)

**Constraint =** relationship among data elements that DBMS has to enforce. If violated, DBMS has to reject it

* **Keys =** set of attributes of a relation so that no two tuples may agree in all the attributes of the set. There can be several keys, and we pick **one** as **Primary key.** 
  + - Primary key may **not be null**.
* **Foreign keys =** referential integrity. Set of attributes of one relation whose values must appear together in certain attributes of another relation.
  + - Referenced attributes must be declared **primary key** or **unique**
* Value-based, tuple based
* Assertions

**Modification on db** = don’t return a result but changes database (opposé de query)

### 3bis/ E/R Model

* Sktech db schema design ⇒ includes constraints but not operations

**Entity relationship diagrams** = designs / pictures

**Entity** = thing/object of the real world

**Entity set =** collection of similar entities (rectangle)

**Attribute =**  property of the entities of an entity set (oval)

⇒ simple & not compound or multivalued !!!

⇒ you have to underline the key attribute(s)

**Relationship =** connection between entity sets (diamond)

⇒ relationship cardinality (min, max) : min : the entity participates at least min times; & max: the entity participates at most max times

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Sens opposé par rapport aux class diagrams !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

⇒ you can have attributes attached to relationships

**Relationship set =** value of a relationship, set of tuples with one component for each related entity set

**Binary relationship (1 - 1)** = between 2 entity sets

**Many-many relationship ([0→n] - [0→n])** = an entity of either set can be connected to many entities of the other set

* **Eg:** bar sells many beers, beer sold by many bars

**Many-one relationship (1 - [0→n])** : (double diamond)

⇒ each entity of the first set connected to at most one entity of the second se

⇒ entity of the second set can be connected to zero, one or many entities of the first set

**One-one relationship ( [0→1]-[0→1])**

**Role =** label of an entity set

**Subclass (isa in triangle) =** special case/fewer properties

In E/R each entity has a representative in all subclasses

Weak entity set (double rectangle) = if to identify entities of E uniquement, you need to follow one or more many-one relationships from E

⇒ weak entity has (1,1) cardinality

Relationship ⇒ relation

3 approaches : use nulls, objec oriented, er style

### Some queries :

## creation

CREATE TABLE <name> (

<name> <type> *PRIMARY KEY NOT NULL,*

<list of elements>

<name> <type> DEFAULT <value> //valeur prédéfinie

*PRIMARY KEY (bar, beer)*

*FOREIGN KEY (<list of attributes>) REFERENCES <relation> (<attributes>)*

);

(choisir entre l’une des 2 déclarations de primary key)

(la première ne marche que s’il y a qu’une seule clé primaire il me semble)

⇒ if from relation R to relation S :

* create S > create R

## b) deletion

DROP TABLE <name> *WHERE <condition>*

⇒ if from relation R to relation S :

* Delete R > delete S

## c) Insertion

INSERT INTO <relation> (<attributes>)

VALUES (<list of values>);

⇒ (<attributes>) facultatif , utile :

* si on veut insérer nb values =/= nb columns (ça compensera le reste par NULL)
* Si on oublie l’ordre d’insertion des valeurs

⇒ you can insert the entire result of a query into a relation :

INSERT INTO <relation> (<subquery>)

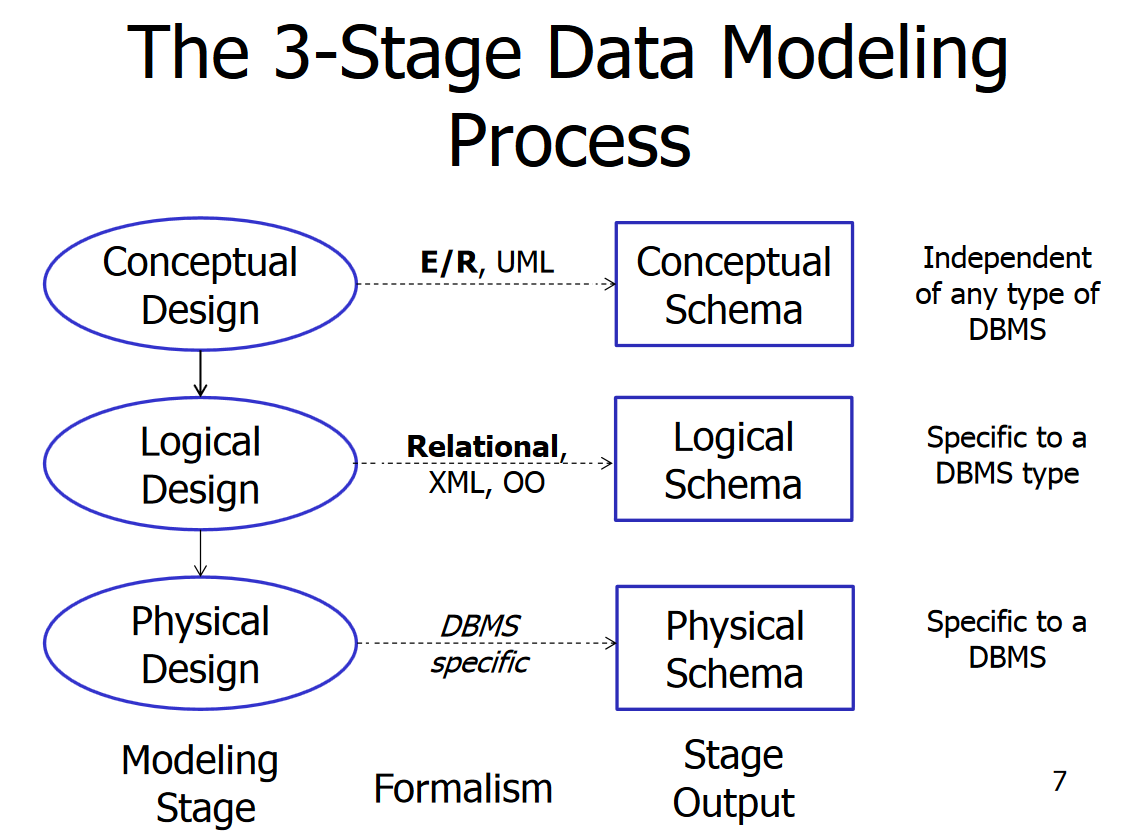
## d) update

UPDATE <relation>

SET <list of attribute assignments>

WHERE <conditions of tuples>;

### 4/ Data modeling



* :
* **Divide and conquer** 
  + ⇒ each step answers specific questions / specific designs.
  + ⇒ Step N changes don’t affect step N-1 schema

### Constraints

Foreign key constraints from relation R to relation S (the primary key is in S and R references it) :

* An insert or update to R introduces values not found in S
* A deletion or update to S causes some tuple of R to “dangle”

At declaration can declare : ON [UPDATE, DELETE][SETNULL|CASCADE]

### Transactions

ACID transactions are:

•Atomic: Whole transaction or none is done.

•Consistent: Database consistency preserved.

•Isolated: It appears to the user as if only one process executes at a time, i.e. serially.

•Durable: Effects of a process survive a crash.

Start a new transaction :

START TRANSACTION

End the transaction :

COMMIT -> complete, modification are now permanent in db

ROLLBACK -> abort, no effects on db

Implicit commit : new START TRANSACTION, statement CREATE TABLE

Implicit Rollback : constraint violation, connection is closed

Isolation levels :

* SERIALIZABLE : ACID, consistency but use lock so no concurrency
* REPEATABLE READ : like read committed but always show at least what was read before
* READ COMMITTED : can only see the committed data, but not necessarily the same data each time
* READ UNCOMMITTED : can see data, even if the data has not been committed

Effects : Read Phenomena

* Dirty reads : reads data that a transaction has modified but not committed yet
* Non-repeatable (fuzzy) reads : reads the same row twice but not with the same value
* Phantom reads : the same query twice but the rows differ from each other

# NoSQL

**Def** : Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge data amount and more.

**BASE** :

***Basic Availability***

* + The database appears to work most of the time. Availability by replication

***Soft-state***

* + Stores don’t have to be write-consistent, nor do different replicas have to be mutually consistent all the time. It is the user’s application’s task to guarantee Consistency

***Eventual consistency***

* + Stores exhibit consistency at some later point (e.g., lazily at read time). Weakly Consistent, the database will be consistent in the long run; ‘stale’ data is OK.

**Characteristics :**

* Weak consistency – stale data OK
* Availability first
* Best effort
* Approximate answers OK
* Aggressive (optimistic)
* Simpler and faster

**CAP theorem** : a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.

MongoDB CP :

* Consistency : All clients always have the same view of the data
* Partition Tolerance : The system works well despite network partitions

MySQL CA :

* Availability : each client can read and write
* Consistency : same as before

## Application

### Create Operations

db.collection.insertOne(

<document>

)

Same thing for insertMany()

In an insert you don’t have to specify the \_id field. It will be automatically added if it is not specified. If specify make sure you never add the same one twice because it will throw an error.

### Read Operations

**Find :**

db.users.find( <- collection

{ age : { $gt : 18 } }, <- query criteria

{ name : 1, address : 1, \_id : 0 } <- projection

). limit(5) <- cursor modifier

**Collection** : the table

**Query criteria** : self-explanatory

**Projection** : what field are going to be displayed

{ field : 1 } display field

{ field : 0 } do not display field

\_id field is always displayed by default

If you want another to be displayed as \_id or if the \_id has been the grouping field you can do { new\_field : “$\_id” }

“$field” works every time you want to reference an index as a value

**Cursor modifier** :

.limit( number\_of\_entries )

.count( )

.sort( field1 : 1 **(ascending)** , field2 : -1 **(descending)** )

.pretty()

.min(field : value, ...)

.max(field : value, ...)

.skip(<offset>)

Comparaison query operators :

* By default **and** if many fields in query criteria
* $or [ {field1 : val1}, {field2 : val2} ]
* [$not](https://docs.mongodb.com/manual/reference/operator/query/not/#op._S_not) Inverts the effect of a query expression and returns documents that do *not* match the query expression.
* [$eq](https://docs.mongodb.com/manual/reference/operator/query/eq/#op._S_eq) Matches values that are equal to a specified value.
* [$gt](https://docs.mongodb.com/manual/reference/operator/query/gt/#op._S_gt) Matches values that are greater than a specified value.
* [$gte](https://docs.mongodb.com/manual/reference/operator/query/gte/#op._S_gte) Matches values that are greater than or equal to a specified value.
* [$in](https://docs.mongodb.com/manual/reference/operator/query/in/#op._S_in) Matches any of the values specified in an array.
* [$lt](https://docs.mongodb.com/manual/reference/operator/query/lt/#op._S_lt) Matches values that are less than a specified value.
* [$lte](https://docs.mongodb.com/manual/reference/operator/query/lte/#op._S_lte) Matches values that are less than or equal to a specified value.
* [$ne](https://docs.mongodb.com/manual/reference/operator/query/ne/#op._S_ne) Matches all values that are not equal to a specified value.
* [$nin](https://docs.mongodb.com/manual/reference/operator/query/nin/#op._S_nin) Matches none of the values specified in an array.
* [$exists](https://docs.mongodb.com/manual/reference/operator/query/exists/#op._S_exists) Matches documents that have the specified field.
* [$type](https://docs.mongodb.com/manual/reference/operator/query/type/#op._S_type) Selects documents if a field is of the specified type.
* [$regex](https://docs.mongodb.com/manual/reference/operator/query/regex/#op._S_regex) Selects documents where values match a specified regular expression.
* [$elemMatch](https://docs.mongodb.com/manual/reference/operator/query/elemMatch/#op._S_elemMatch) Selects documents if element in the array field matches all the specified [$elemMatch](https://docs.mongodb.com/manual/reference/operator/query/elemMatch/#op._S_elemMatch) conditions.
* [$size](https://docs.mongodb.com/manual/reference/operator/query/size/#op._S_size) Selects documents if the array field is a specified size.

Examples :

Select all documents :

db.inventory.find( {} )

Select documents where field “status” equals “D” :

db.inventory.find( { status: "D" } )

Select documents where “uom” is nested in “size” and equals “in” :

db.inventory.find( { "size.uom": "in" } )

Select an element in an array

db.inventory.find( { tags: "red" } )

Select an element exactly in array

db.inventory.find( { tags: ["red", "blank"] } )

### Update Operations

db.user.updateMany( <- collection

{ age: {$lt : 18} }, <- update filter

{ $set : { status : “reject” } }, <- update action

{  
 upsert: <boolean>,  
 writeConcern: <document>,  
 collation: <document>,  
 arrayFilters: [ <filterdocument1>, ... ]  
 }

)

updateMany changes all the documents corresponding to the filter

updateOne changes only the first document corresponding to the filter

db.restaurant.replaceOne(  
 { "name" : "Central Perk Cafe" }, <- filter  
 { "name" : "Central Pork Cafe", "Borough" : "Manhattan" } <- replacement  
 );

replaceOne replace the first matching document in filter by the provided one

When doing so it returns a document with :

“acknowledged” : true

“matchedCount”, number of field corresponding to the filter

“modifiedCount”, number of documents updated.

If the option upsert is set to true, then if no document is matched then it inserts a new one.

Operators :

* [$currentDate](https://docs.mongodb.com/manual/reference/operator/update/currentDate/#up._S_currentDate) Sets the value of a field to current date, either as a Date or a Timestamp.
* [$inc](https://docs.mongodb.com/manual/reference/operator/update/inc/#up._S_inc) Increments the value of the field by the specified amount.
* [$min](https://docs.mongodb.com/manual/reference/operator/update/min/#up._S_min) Only updates the field if the specified value is less than the existing field value.
* [$max](https://docs.mongodb.com/manual/reference/operator/update/max/#up._S_max) Only updates the field if the specified value is greater than the existing field value.
* [$mul](https://docs.mongodb.com/manual/reference/operator/update/mul/#up._S_mul) Multiplies the value of the field by the specified amount.
* [$rename](https://docs.mongodb.com/manual/reference/operator/update/rename/#up._S_rename) Renames a field.
* [$set](https://docs.mongodb.com/manual/reference/operator/update/set/#up._S_set) Sets the value of a field in a document.
* [$setOnInsert](https://docs.mongodb.com/manual/reference/operator/update/setOnInsert/#up._S_setOnInsert) Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
* [$unset](https://docs.mongodb.com/manual/reference/operator/update/unset/#up._S_unset) Removes the specified field from a document.

### Delete Operations

db.orders.deleteOne( { "expiryts" : { $lt: ISODate("2015-11-01T12:40:15Z") } } );

deleteOne() delete the first document matching the filter

deleteMany() delete all documents matching the filter

Example :

Document here needs to match both filters

db.orders.deleteMany( { "stock" : "Brent Crude Futures", "limit" : { $gt : 48.88 } } );

### Aggregation

Aggregation is used when want to calculate stuff, group by something , …

Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

Each pipeline work on the preceding output as its input

Stages :

* $match, filter the documents
  + $match: { $or: [ { score: { $gt: 70, $lt: 90 } }, { views: { $gte: 1000 } } ] }
* $group, group on \_id field as a key the input documents and computes in another field
  + $group: { \_id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... }
  + $group: { \_id: null, count: { $sum: 1 } }
  + [$addToSet](https://docs.mongodb.com/manual/reference/operator/aggregation/addToSet/#grp._S_addToSet) Returns an array of *unique* expression values for each group. Order of the array elements is undefined.
  + [$avg](https://docs.mongodb.com/manual/reference/operator/aggregation/avg/#grp._S_avg) Returns an average of numerical values. Ignores non-numeric values.
  + [$first](https://docs.mongodb.com/manual/reference/operator/aggregation/first/#grp._S_first) Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.
  + [$last](https://docs.mongodb.com/manual/reference/operator/aggregation/last/#grp._S_last) Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.
  + [$max](https://docs.mongodb.com/manual/reference/operator/aggregation/max/#grp._S_max) Returns the highest expression value for each group.
  + [$mergeObjects](https://docs.mongodb.com/manual/reference/operator/aggregation/mergeObjects/#exp._S_mergeObjects) Returns a document created by combining the input documents for each group.
  + [$min](https://docs.mongodb.com/manual/reference/operator/aggregation/min/#grp._S_min) Returns the lowest expression value for each group.
  + [$push](https://docs.mongodb.com/manual/reference/operator/aggregation/push/#grp._S_push) Returns an array of expression values for each group.
  + [$stdDevPop](https://docs.mongodb.com/manual/reference/operator/aggregation/stdDevPop/#grp._S_stdDevPop) Returns the population standard deviation of the input values.
  + [$stdDevSamp](https://docs.mongodb.com/manual/reference/operator/aggregation/stdDevSamp/#grp._S_stdDevSamp) Returns the sample standard deviation of the input values.
  + [$sum](https://docs.mongodb.com/manual/reference/operator/aggregation/sum/#grp._S_sum) Returns a sum of numerical values. Ignores non-numeric values.
* $sort, 1 to specify ascending order, -1 to specify descending order.
  + $sort: { <field1>: <sort order>, <field2>: <sort order> ... }
* $limit, $limit: <positive integer>
* $project, format and print the result
  + $project : { \_id: 0, title : 1 , author : 1 }

Examples :

db.orders.aggregate([  
 { $match: { status: "A" } },  
 { $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } },  
 { $sort: { total: -1 } }  
 ])