CATRUSTDB

Guide utilisateur

Version 0.1 · Avril 2026

Clément LECOMTE

Zéro SQL. Zéro injection. Garantie mathématique.

Table des matières

1 Installation **2**
1.1 Depuis les sources . . . 2
1.2 Vérification . . . 2
1.3 Docker . . . 2

2 Concepts essentiels **2**

3 Écrire un schéma CQL **3**
3.1 Syntaxe . . . 3
3.2 Types de base disponibles . . . 3
3.3 Exemple complet . . . 3

4 Écrire une instance CQL **4**
4.1 Syntaxe . . . 4
4.2 Exemple . . . 4

5 Faire des requêtes **5**
5.1 Syntaxe CQL . . . 5
5.2 Exemples . . . 5
5.3 Opérateurs WHERE . . . 5
5.4 Via le serveur . . . 5

6 Migrer des données avec Σ **6**
6.1 1. Définir le schéma cible . . . 6
6.2 2. Définir le Mapping . . . 6
6.3 3. Exécuter via le serveur . . . 7
6.4 Garanties de Σ . . . 7

7 Le serveur TCP NDJSON **7**
7.1 Démarrage . . . 7
7.2 Protocole NDJSON . . . 7
7.3 Client Python . . . 8

8 Persistence WAL **8**
8.1 Sauvegarder . . . 8
8.2 Recharger après un crash . . . 8
8.3 Format du fichier WAL . . . 8
8.4 Avantages . . . 8

9 Les index O(1) **9**
9.1 API Rust . . . 9
9.2 Quand indexer ? . . . 9

10 Intégration Rust **9**
10.1 `Cargo.toml` . . . 10
10.2 Exemple complet . . . 10

11 Référence des commandes serveur **11**
11.1 `ping` . . . 11
11.2 `load` . . . 11
11.3 `from_wal` . . . 11
11.4 `query` . . . 11
11.5 `sigma` . . . 11
11.6 `stats` . . . 12
11.7 `persist` . . . 12

12 Dépannage **12**

1 Installation

1.1 Depuis les sources

```
git clone https://github.com/clementremy/catrust
cd catrust
cargo build --release -p catrustdb-server
# Binaire : ./target/release/catrustdb
```

1.2 Vérification

```
catrustdb --help
# Usage: catrustdb [--host <HOST>] [--port <PORT>]

catrustdb --port 7474 &
echo '{"cmd":"ping"}' | nc localhost 7474
# -> {"ok":true,"msg":"pong"}
```

1.3 Docker

```
docker run -p 7474:7474 catrust/catrustdb:latest
```

2 Concepts essentiels

CATRUSTDB est fondé sur la **théorie des catégories**. Voici la traduction pratique :

Terme mathématique	Ce que c'est en pratique
Schéma (catégorie)	Structure des données : tables, colonnes, FK
Instance (foncteur)	Les données elles-mêmes : les lignes
Mapping (foncteur)	Règle de restructuration schéma → schéma
Δ (delta)	Restriction / filtre (le WHERE de CQL)
Σ (sigma)	Migration : déplacer les données d'un schéma dans un autre

Information

Dans CATRUSTDB, tout cela se passe **en mémoire, en Rust pur, sans SQL**.

Le cycle de vie des données :

`Schéma CQL` → `Instance CQL` → `Requête CQL` → `Résultat`

`Mapping` → Σ `(migration)` → `Nouvelle Instance`

3 Écrire un schéma CQL

Un schéma décrit la **structure** de vos données.

3.1 Syntaxe

```
schema NomDuSchema {
  entities
    NomEntite1
    NomEntite2

  foreign_keys
    nom_fk : EntiteSource -> EntiteCible

  attributes
    nom_attr : EntiteSource -> TypeBase
}
```

3.2 Types de base disponibles

Type CQL	Exemple CQL	Type Rust interne
String	"Alice"	Val::Str(String)
Float	90000.0	Val::Float(f64)
Integer	42	Val::Int(i64)
Boolean	true	Val::Bool(bool)

3.3 Exemple complet

```
schema Company {
  entities
    Employee
    Department

  foreign_keys
    dept : Employee -> Department

  attributes
    emp_name  : Employee   -> String
    salary    : Employee   -> Float
    dept_name : Department -> String
    budget    : Department -> Float
}
```

Listing 1 – Schéma Company

Attention

Les noms d'entités commencent par une **majuscule**. Les entités cibles des FK doivent être déclarées dans le schéma.

4 Écrire une instance CQL

Une instance contient les **données** conformes à un schéma.

4.1 Syntaxe

```
instance NomInstance : NomSchema {
  NomEntite {
    symbole  attr1="valeur"  attr2=42.0  fk=symbole_cible
  }
}
```

Les *symboles* (`d1`, `e1`…) sont des étiquettes locales pour les références croisées. Ils n'apparaissent pas dans les données finales.

Attention

Les entités cibles (ex. : `Department`) doivent être déclarées **avant** celles qui les référencent (`Employee`).

4.2 Exemple

```
instance SampleData : Company {
  Department {
    d1  dept_name="Engineering"  budget=150000.0
    d2  dept_name="Marketing"    budget=80000.0
    d3  dept_name="Research"     budget=200000.0
  }

  Employee {
    e1  dept=d1  emp_name="Alice"    salary=90000.0
    e2  dept=d1  emp_name="Bob"      salary=75000.0
    e3  dept=d2  emp_name="Charlie"  salary=70000.0
    e4  dept=d2  emp_name="Diana"    salary=82000.0
    e5  dept=d3  emp_name="Eve"      salary=95000.0
  }
}
```

Listing 2 – Données Company

5 Faire des requêtes

5.1 Syntaxe CQL

```
query NomRequete : NomSchema {
  from    alias : Entite
  where   alias.attribut > valeur
  select  alias.attr1  alias.attr2  alias.fk.attr_distant
}
```

5.2 Exemples

```
query AllEmployees : Company {
  from    e : Employee
  select  e.emp_name  e.salary  e.dept.dept_name
}
```

Listing 3 – Tous les employés

```
query HighEarners : Company {
  from    e : Employee
  where   e.salary > 80000
  select  e.emp_name  e.salary  e.dept.dept_name
}
```

Listing 4 – Employés bien payés (traversal FK inclus)

5.3 Opérateurs WHERE

Opérateur	Signification
`=`	Égalité exacte
`!=`	Différent
`>`	Strictement supérieur
`<`	Strictement inférieur
`>=`	Supérieur ou égal
`<=`	Inférieur ou égal

Conseil

Coercions numériques : `Float` et `Integer` se comparent automatiquement.

5.4 Via le serveur

```
echo '{
  "cmd": "query",
  "cql_query": "query Q : Company { from e : Employee where e.
     salary > 80000 select e.emp_name e.salary }"
```

```
}' | nc localhost 7474
```

Réponse :

```
{
  "ok": true,
  "name": "Q",
  "columns": ["emp_name", "salary"],
  "rows": [["Alice","90000.0"], ["Diana","82000.0"], ["Eve","
     95000.0"]],
  "row_count": 3
}
```

6 Migrer des données avec Σ

Σ (sigma) est l'opération de **migration catégorique**. Elle prend des données dans un schéma source et les transforme dans un schéma cible selon un Mapping.

6.1 1. Définir le schéma cible

```
schema Org {
  entities
    Staff
    Team

  foreign_keys
    member_of : Staff -> Team

  attributes
    full_name : Staff -> String
    pay       : Staff -> Float
    team_name : Team  -> String
}
```

Listing 5 – Schéma cible Org

6.2 2. Définir le Mapping

```
mapping Rename : Company -> Org {
  entities
    Employee   -> Staff
    Department -> Team

  foreign_keys
    dept -> member_of

  attributes
```

```
    emp_name  -> full_name
    salary    -> pay
    dept_name -> team_name
}
```

Listing 6 – Mapping Company -> Org

6.3 3. Exécuter via le serveur

```
# Charger la source
echo '{"cmd":"load","schema_cql":"...Company CQL...","instance_cql
    ":"..."}' \
  | nc localhost 7474

# Migrer vers Org
echo '{"cmd":"sigma","target_cql":"...Org CQL...","mapping_cql
    ":"...Rename CQL..."}' \
  | nc localhost 7474
```

6.4 Garanties de Σ

— **Toutes les FK sont remappées** : aucune clé orpheline possible

— **L'ordre est préservé** : les lignes arrivent dans le même ordre relatif

— **Les attributs non mappés sont `NULL`** : comportement explicite, pas silencieux

7 Le serveur TCP NDJSON

7.1 Démarrage

```
catrustdb --host 127.0.0.1 --port 7474
# [catrustdb] tcp://127.0.0.1:7474 -- pret
```

Options disponibles :

Option	Défaut	Description
`--host`	`127.0.0.1`	Adresse d'écoute
`--port`	`7474`	Port TCP

7.2 Protocole NDJSON

Le protocole est **NDJSON** (Newline-Delimited JSON) :

— Chaque **requête** = une ligne JSON terminée par `\n`

— Chaque **réponse** = une ligne JSON terminée par `\n`

— La connexion reste ouverte — envoi de plusieurs commandes possible

7.3 Client Python

```python
import socket, json

def send(s, cmd):
    s.sendall((json.dumps(cmd) + "\n").encode())
    return json.loads(s.makefile().readline())

s = socket.socket()
s.connect(("127.0.0.1", 7474))
print(send(s, {"cmd": "ping"}))
# -> {'ok': True, 'msg': 'pong'}
```

8 Persistence WAL

CATRUSTDB utilise un **Write-Ahead Log** (WAL) au format NDJSON.

8.1 Sauvegarder

```
echo '{"cmd":"persist","path":"/var/data/mydb.wal"}' | nc localhost
    7474
# -> {"ok":true,"path":"/var/data/mydb.wal"}
```

8.2 Recharger après un crash

```
echo '{"cmd":"from_wal","path":"/var/data/mydb.wal"}' | nc
   localhost 7474
# -> {"ok":true,"entities":2,"rows":8}
```

8.3 Format du fichier WAL

```
{"Schema":{"cql":"schema␣Company␣{␣...␣}"}}
{"Insert":{"entity":"Department","attrs":[["dept_name","Engineering
   "],
  ["budget",150000.0]],"fks":[]}}
{"Insert":{"entity":"Employee","attrs":[["emp_name","Alice"],
  ["salary",90000.0]],"fks":[["dept",0]]}}
{"Checkpoint":null}
```

8.4 Avantages

— **Lisible par un humain** — inspecter avec `cat` ou `less`

— **Diffable** — `git diff db_v1.wal db_v2.wal` montre exactement ce qui a changé

— **Append-only** — écriture séquentielle, pas de réécriture de pages

— **Rejouable** — en cas de corruption partielle, tout ce qui précède la coupure est récupérable

9 Les index O(1)

Par défaut, CATRUSTDB fait des scans O(n) sur les colonnes. Les **index d'égalité** permettent de passer en O(1) pour les requêtes `WHERE attr = val`.

9.1 API Rust

```
let store = db.entity_mut("Employee").unwrap();

// Construire l'index sur salary
store.build_index("salary");

// Lookup O(1) - retourne les RowIds directement
let rows = store.lookup_eq("salary", &Val::Float(90000.0));
// -> &[0]  (Alice est a la ligne 0)

// Supprimer l'index
store.drop_index("salary");
```

Information

Une fois `build_index("attr")` appelé, **chaque `insert_row` maintient l'index automatiquement**. Il n'y a pas de risque d'index périmé.

La fonction `filter_entity` détecte automatiquement la présence d'un index :

— Si l'index existe → O(1) lookup via HashMap

— Sinon → O(n) scan séquentiel

9.2 Quand indexer ?

Situation	Conseil
Requêtes répétées sur la même colonne	Indexez
Colonne de haute cardinalité (IDs, emails)	Indexez
Colonne booléenne peu sélective	Inutile
Données insérées une fois, jamais requêtées	Inutile

10 Intégration Rust

CATRUSTDB est une bibliothèque Rust en premier lieu. Le serveur TCP n'est qu'un wrapper.

10.1 Cargo.toml

```
[dependencies]
catrustdb-store = { path = "../catrustdb-store" }
catrustdb-eval  = { path = "../catrustdb-eval" }
catrust-core    = { path = "../catrust-core", features = ["serde"]
    }
catrust-parser  = { path = "../catrust-parser" }
```

10.2 Exemple complet

```
use catrustdb_store::{Database, Val};
use catrustdb_eval::{Pred, filter_entity};
use catrust_core::{schema::Schema, typeside::{BaseType, Value},
    instance::Instance};
use std::collections::HashMap;

fn main() {
    // 1. Creer un schema
    let mut schema = Schema::new("Demo");
    schema.add_node("Employee");
    schema.add_attribute("emp_name", "Employee", BaseType::String);
    schema.add_attribute("salary",   "Employee", BaseType::Float);

    // 2. Creer une instance
    let mut inst = Instance::new("demo", &schema);
    inst.insert("Employee",
        [("emp_name".into(), Value::String("Alice".into())),
         ("salary".into(),   Value::Float(90000.0))].into_iter().
            collect(),
        HashMap::new());

    // 3. Charger dans CATRUSTDB
    let mut db = Database::new(schema);
    db.load_instance(&inst).unwrap();

    // 4. Index O(1) + requete
    db.entity_mut("Employee").unwrap().build_index("salary");
    let rows = filter_entity(&db, "Employee", &Pred::Eq {
        path: vec!["salary".into()],
        val:  Val::Float(90000.0),
    });
    println!("Lignes salary=90000: {:?}", rows); // -> [0]

    // 5. Sauvegarder
    db.schema_cql = "schema Demo { ... }".to_string();
    db.persist(std::path::Path::new("/tmp/demo.wal")).unwrap();

```

```
    // 6. Recharger
    let db2 = Database::from_wal(std::path::Path::new("/tmp/demo.
        wal")).unwrap();
    println!("Lignes rechargees: {}", db2.total_rows()); // -> 1
}
```

Listing 7 – Cycle complet : créer, indexer, persister, recharger

11 Référence des commandes serveur

11.1 ping

```
{"cmd": "ping"}
```

```
{"ok": true, "msg": "pong"}
```

11.2 load

Charge un schéma CQL (et optionnellement une instance) en mémoire.

```
{"cmd": "load", "schema_cql": "<CQL>", "instance_cql": "<CQL
    optionnel>"}
```

```
{"ok": true, "entities": 2, "rows": 8}
```

11.3 from_wal

```
{"cmd": "from_wal", "path": "/chemin/vers/fichier.wal"}
```

```
{"ok": true, "entities": 2, "rows": 8}
```

11.4 query

```
{"cmd": "query", "cql_query": "query Q : Company { from e:Employee
    select e.emp_name }"}
```

```
{"ok": true, "name": "Q", "columns": ["emp_name"],
 "rows": [["Alice"], ["Bob"]], "row_count": 2}
```

11.5 sigma

```
{"cmd": "sigma", "target_cql": "<CQL cible>", "mapping_cql": "<CQL
    mapping>"}
```

```
{"ok": true, "schema": "Org",
 "entities": [{"name": "Staff", "count": 5, "columns": ["full_name"
     ,"pay"], "rows": [...]}]}
```

11.6 stats

```
{"cmd": "stats"}
```

```
{"ok": true, "loaded": true, "schema": "Company",
 "entities": [{"name": "Department","rows": 3}, {"name": "Employee"
    ,"rows": 5}],
 "total_rows": 8}
```

11.7 persist

```
{"cmd": "persist", "path": "/chemin/vers/fichier.wal"}
```

```
{"ok": true, "path": "/chemin/vers/fichier.wal"}
```

12 Dépannage

"aucune base chargée"

Envoyez d'abord `{"cmd":"load",...}` ou `{"cmd":"from_wal",...}` avant toute commande `query`, `sigma`, `stats` ou `persist`.

Erreur de parsing CQL

Vérifiez :

- Les accolades `{ }` sont fermées
- `entities`, `foreign_keys`, `attributes` ont des blocs séparés
- Les noms d'entités commencent par une majuscule

Performance : scans lents sur grande table

Construisez un index sur la colonne filtrée :

```
db.entity_mut("Employee").unwrap().build_index("salary");
```

Le filtre passe de $O(n)$ à $O(1)$ pour les requêtes d'égalité.

Le serveur ne répond plus

```
pkill catrustdb
catrustdb --port 7474 &
echo '{"cmd":"from_wal","path":"/chemin/vers/derniere.wal"}' |
    nc localhost 7474
```

CATRUSTDB — Zéro SQL. Zéro injection. Garantie mathématique.