# cqlsh: klient wiersza poleceń

**Poziom: podstawy** **Dotyczy: Apache Cassandra 3.x / 4.x** **Czas: ~20 min**

`cqlsh` *jest interaktywnym klientem CQL dostarczanym razem z Cassandrą — łączy interpreter zapytań z zestawem własnych poleceń, które nie są CQL-em, tylko sterują samą sesją albo pytają o metadane klastra.*

### Skróty użyte na tej stronie (3)

| Skrót | Co znaczy |
|---|---|
| **CQL** | Cassandra Query Language: język zapytań Cassandry, podobny do SQL |
| **CSV** | w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami |
| **TLS** | Transport Layer Security: szyfrowanie połączeń, np. w HTTPS |

# Fiszka

**TYLKO ODCZYT**

## Wylistowanie wszystkich keyspace: DESCRIBE KEYSPACES

```
DESCRIBE KEYSPACES;
```

Zwracamy nazwy wszystkich keyspace istniejących w klastrze, łącznie z wewnętrznymi keyspace systemowymi Cassandry.

Przykład:

```
system_schema  system            system_distributed  system_auth
aplikacja      reaper_db         system_traces        system_views
```

**TYLKO ODCZYT**

## Wylistowanie tabel bieżącego keyspace: DESCRIBE TABLES

```
DESCRIBE TABLES;
```

Bez wskazania keyspace `DESCRIBE TABLES` pokazuje tabele tego keyspace, do którego jesteśmy podłączeni poleceniem `USE` — jeśli nie wykonaliśmy `USE`, zwraca tabele wszystkich keyspace naraz, pogrupowane nagłówkami.

Przykład:

```
Keyspace aplikacja
------------------
klienci  zamowienia_wg_klienta  zamowienia_wg_statusu  sesje
```

**TYLKO ODCZYT**

## Wyświetlenie definicji tabeli: DESCRIBE TABLE

```
DESCRIBE TABLE NazwaTabeli;
```

> **Pojęcie: definicja obiektu w DESCRIBE.** W przeciwieństwie do `\d` w `psql`, `DESCRIBE TABLE` w Cassandrze zwraca gotową, wykonywalną instrukcję `CREATE TABLE`, którą tabela powstała — nie tabelaryczne zestawienie kolumn, tylko kod do skopiowania i odtworzenia struktury gdzie indziej.

Przykład:

```
CREATE TABLE aplikacja.zamowienia_wg_klienta (
    klient_id uuid,
    data_zamowienia timestamp,
    id_zamowienia uuid,
    wartosc decimal,
    status text,
    PRIMARY KEY ((klient_id), data_zamowienia, id_zamowienia)
) WITH CLUSTERING ORDER BY (data_zamowienia DESC, id_zamowienia ASC)
    AND gc_grace_seconds = 864000;
```

**TYLKO ODCZYT**

## Wyświetlenie topologii i wersji schematu klastra: DESCRIBE CLUSTER

```
DESCRIBE CLUSTER;
```

Zwracamy nazwę klastra, użyty snitch i partycjoner, oraz wersje schematu widziane przez poszczególne węzły — rozjazd wersji schematu między węzłami jest sygnałem, że zmiana struktury (np. `CREATE TABLE`) nie zdążyła się jeszcze propagować przez gossip do całego klastra.

Przykład:

```
Cluster: produkcja
Partitioner: Murmur3Partitioner

Schema versions:
a1b2c3d4-e5f6-3789-a123-456789abcdef: [10.20.10.36, 10.20.10.37, 10.20.10.38]
```

**□□ TYLKO ODCZYT**

## Przełączenie się na keyspace: USE

```
USE NazwaKeyspace;
```

Łączymy bieżącą sesję z konkretnym keyspace — kolejne polecenia bez podanej nazwy keyspace przed nazwą tabeli (`SELECT * FROM zamowienia_wg_klienta` zamiast `SELECT * FROM aplikacja.zamowienia_wg_klienta`) będą odnosić się właśnie do niego.

Przykład:

```
USE aplikacja;
```

```
dba@cqlsh:aplikacja>
```

Prompt sesji zmienia się, dołączając nazwę bieżącego keyspace — to jedyny wizualny sygnał w `cqlsh`, do którego keyspace jesteśmy aktualnie podłączeni.

**□□ TYLKO ODCZYT**

## Zmiana poziomu spójności dla bieżącej sesji: CONSISTENCY

```
CONSISTENCY LOCAL_QUORUM;
```

Ustawiamy poziom spójności opisany na stronie [Architektura Cassandry: pierścień, węzeł, replikacja] dla wszystkich kolejnych zapytań tej sesji, dopóki nie zmienimy go ponownie albo nie zakończymy sesji. Bez argumentu `CONSISTENCY` samo wypisuje bieżące ustawienie.

Przykład:

```
Consistency level set to LOCAL_QUORUM.
```

**TYLKO ODCZYT**

## Włączenie śledzenia wykonania zapytania: TRACING ON

```
TRACING ON;
```

Po włączeniu każde kolejne zapytanie wypisuje pod wynikiem szczegółowy ślad tego, co się z nim działo — który węzeł był coordinatorem, do których replik poszło zapytanie, ile trwał każdy etap. Pierwsze narzędzie, po które sięgamy przy diagnozowaniu, dlaczego konkretne zapytanie jest wolniejsze niż się spodziewaliśmy.

Przykład:

```
Tracing session: a1b2c3d4-1111-2222-3333-444455556666

                     activity                          |  timestamp   |   source    |
source_elapsed
-----------------------------------------------------+--------------+-------------
+---------------
 Execute CQL3 query                                  | 12:04:01.001 | 10.20.10.36
|              0
 Parsing statement                                    | 12:04:01.001 | 10.20.10.36
|            112
 Sending READ message to /10.20.10.37                 | 12:04:01.002 | 10.20.10.36
|            891
 Executing single-partition query on zamowienia...   | 12:04:01.002 | 10.20.10.37
|             45
 Request complete                                    | 12:04:01.004 | 10.20.10.36
|           2871
```

**TYLKO ODCZYT**

## Przełączenie formatu wyniku na rozszerzony: EXPAND ON

```
EXPAND ON;
```

Zamiast szerokiej tabeli z kolumnami obok siebie, każdy wiersz wyniku wypisywany jest pionowo — odpowiednik `\x` w `psql`, opisanego na stronie psql: klient wiersza poleceń, przydatny przy wierszach z wieloma kolumnami albo z długą treścią w jednej z nich.

Przykład:

```
Expanded display is on.
```

```
@ Row 1
--------------------------
klient_id         | 3c1f6e2a-1111-4b3a-9c2d-000000000001
data_zamowienia   | 2026-03-14 09:12:00.000000+0000
id_zamowienia     | 9f2e1a3b-4444-5555-6666-777788889999
wartosc           | 1250.00
status            | zrealizowane
```

**TYLKO ODCZYT**

## Wykonanie skryptu CQL z pliku: SOURCE

```
SOURCE 'SciezkaDoPliku';
```

Wczytujemy i wykonujemy zawartość wskazanego pliku, tak jakby jego treść została wpisana ręcznie w sesji — odpowiednik `\i` w `psql`, standardowy sposób odtworzenia zrzutu struktury albo wsadowego załadowania danych z pliku `.cql`.

Przykład:

```
SOURCE '/home/nazwa_usera/schema.cql';
```

**CQL** = Cassandra Query Language: język zapytań Cassandry, podobny do SQL

**TYLKO ODCZYT**

## Eksport tabeli do pliku CSV: COPY TO

```
COPY NazwaTabeli TO 'SciezkaDoPliku' WITH HEADER = true;
```

Eksportujemy zawartość tabeli do pliku CSV po stronie klienta — mechanizm sensowny dla niewielkich zbiorów danych (rząd wielkości tysięcy do niskich milionów wierszy); dla większych wolumenów właściwym narzędziem jest `sstableloader`, opisany na stronie Backup i odtwarzanie, bo `COPY` odczytuje dane przez zwykłe zapytania CQL, jeden wiersz na raz, a nie strumieniowo z plików SSTable.

Przykład:

```
COPY aplikacja.klienci TO '/home/nazwa_usera/klienci.csv' WITH HEADER = true;
```

```
Using 7 child processes


Starting copy of aplikacja.klienci with columns [id, nazwa, email, miasto].
Processed: 48213 rows; Rate: 12842 rows/s; Avg. rate: 11207 rows/s
48213 rows exported to 1 files in 4.302 seconds.
```

**CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami ·**CQL** = Cassandra Query Language: język zapytań Cassandry, podobny do SQL

**TYLKO ODCZYT**

## Import danych z pliku CSV: COPY FROM

```
COPY NazwaTabeli FROM 'SciezkaDoPliku' WITH HEADER = true;
```

Wczytujemy dane z pliku CSV, wiersz po wierszu, jako zwykłe instrukcje `INSERT` wykonywane przez klienta — z tego samego powodu co przy eksporcie, sensowne przy mniejszych zbiorach; duży import wsadowy wykonujemy raczej `sstableloader` albo generując pliki SSTable bezpośrednio.

Przykład:

```
COPY aplikacja.klienci FROM '/home/nazwa_usera/klienci.csv' WITH HEADER = true;
```

```
Using 7 child processes
```

```
Starting copy of aplikacja.klienci with columns [id, nazwa, email, miasto].
Processed: 48213 rows; Rate: 9821 rows/s; Avg. rate: 8734 rows/s
48213 rows imported from 1 files in 5.519 seconds (0 skipped).
```

□□ **CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami

**□□ TYLKO ODCZYT**

## Ograniczenie liczby wierszy pobieranych na jedną stronę: PAGING

```
PAGING 100;
```

Ustawiamy rozmiar strony wyniku — `cqlsh` domyślnie pobiera wyniki porcjami po 100 wierszy i pyta, czy pobrać kolejną porcję, zamiast ściągać cały wynik naraz do pamięci klienta. `PAGING OFF` wyłącza to zachowanie i pobiera cały wynik jednym ciągiem, co przy dużych zapytaniach bywa nieostrożne.

Przykład:

```
Page size: 100
```

**□□ TYLKO ODCZYT**

## Wyświetlenie pomocy: HELP

```
HELP;
```

Zwracamy listę wszystkich poleceń rozpoznawanych przez `cqlsh` — zarówno CQL, jak i poleceń klienta opisanych na tej stronie. `HELP NazwaPolecenia` (na przykład `HELP COPY`) pokazuje szczegółowy opis jednego polecenia.

Przykład:

```
Documented shell commands:
===========================
CAPTURE  CLS       COPY  DESCRIBE  EXPAND  LOGIN  SERIAL  SOURCE  UNICODE
CLEAR    CONSISTENCY  DESC  EXIT  HELP    PAGING  SHOW    TRACING

CQL help topics:
```

```
================
ALTER          CREATE_INDEX      DROP_MATERIALIZED_VIEW  SELECT
ALTER_KEYSPACE CREATE_KEYSPACE   DROP_ROLE                UPDATE
...
```

**CQL** = Cassandra Query Language: język zapytań Cassandry, podobny do SQL

# Z życia wzięte

**Z ŻYCIA WZIĘTE**

Tomek dostaje zadanie sprawdzenia, jakie keyspace w ogóle istnieją na klastrze cass01 do cass03, zanim dołoży tam kolejną tabelę pod nowy typ skanera. DESCRIBE KEYSPACES pokazuje oprócz aplikacji telemetrycznej też keyspace reaper_db, o którym Tomek nigdy wcześniej nie słyszał. Dopiero DESCRIBE TABLE na jednej z tabel telemetrii pokazuje mu gotową definicję CREATE TABLE, którą może skopiować i dostosować pod nowy typ danych, zamiast pisać ją od zera.

# Czym jest cqlsh

**Pojęcie: cqlsh.** Interaktywny klient wiersza poleceń napisany w Pythonie, łączący się z klastrem przez natywny protokół Cassandry na porcie `9042`. Podobnie jak `psql` w PostgreSQL, `cqlsh` rozróżnia zwykłe polecenia CQL (`SELECT`, `INSERT`, `CREATE TABLE`) od własnych poleceń klienta, które CQL-em nie są — `DESCRIBE`, `CONSISTENCY`, `PAGING` — i które klient obsługuje sam, bez wysyłania ich do serwera jako zapytanie.

# Połączenie z klastrem

Do jakiego węzła i z jakimi poświadczeniami łączy się `cqlsh`, opisują parametry przekazane przy starcie:

## Połączenie z klastrem: cqlsh

```
cqlsh AdresWezla 9042 -u NazwaUsera -p HasloUzytkownika
```

Łączymy się z dowolnym węzłem klastra — nie musi to być węzeł, który akurat przechowuje dane, którymi jesteśmy zainteresowani, bo łączymy się z coordinatorem opisanym na stronie Architektura Cassandry: pierścień, węzeł, replikacja, a on sam odnajdzie właściwe repliki. Port `9042` jest portem natywnego protokołu binarnego i jest domyślny, więc pomijamy go, jeśli nie został zmieniony w konfiguracji.

Przykład:

```
Connected to produkcja at 10.20.10.36:9042
[cqlsh 6.1.0 | Cassandra 4.1.3 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
dba@cqlsh>
```

Bez `-u` i `-p` `cqlsh` próbuje połączyć się bez uwierzytelniania — powiedzie się to tylko wtedy, gdy w `cassandra.yaml` uwierzytelnianie jest wyłączone (`authenticator: AllowAllAuthenticator`), co w środowisku produkcyjnym praktycznie się nie zdarza.

# Połączenie z wymuszeniem szyfrowania: --ssl

```
cqlsh AdresWezla 9042 -u NazwaUsera -p HasloUzytkownika --ssl
```

Wymuszamy połączenie przez TLS, gdy klaster ma włączone szyfrowanie klient-węzeł (`client_encryption_options` w `cassandra.yaml`) — bez tej flagi `cqlsh` spróbuje połączyć się zwykłym, nieszyfrowanym protokołem i zostanie odrzucony przez węzeł wymagający szyfrowania.

# Wymuszenie konkretnej wersji CQL: --cqlversion

```
cqlsh AdresWezla 9042 -u NazwaUsera -p HasloUzytkownika --cqlversion=3.4.6
```

Wymuszamy konkretną wersję protokołu CQL zamiast negocjowanej automatycznie — przydatne przy diagnozowaniu niezgodności między wersją `cqlsh` a wersją serwera, rzadziej potrzebne w codziennej pracy.

# Plik .cqlshrc — konfiguracja sesji

> **Pojęcie: .cqlshrc.** Plik w katalogu domowym (`~/.cassandra/cqlshrc`), wczytywany automatycznie przy każdym starcie `cqlsh` — miejsce na dane połączenia i ustawienia, które inaczej trzeba by powtarzać przy każdym wywołaniu z linii poleceń.

```
[connection]
hostname = 10.20.10.36
port = 9042

[authentication]
username = nazwa_usera
password = HasloUzytkownika

[ssl]
certfile = ~/.cassandra/rootca.crt
validate = true
```

Plik z hasłem w treści powinien mieć uprawnienia `0600` — z tego samego powodu, dla którego wymaga tego `.pgpass` w PostgreSQL, opisany na stronie psql: klient wiersza poleceń.

# Częste błędy

**Podawanie hasła w `-p` widocznego w historii powłoki.** Argument `-p` trafia do `~/.bash_history` w postaci jawnej — plik `.cqlshrc` z uprawnieniami `0600` jest tu bezpieczniejszym rozwiązaniem, dokładnie z tego samego powodu, dla którego `psql` odradza `PGPASSWORD`.

**`COPY FROM` na dużym pliku bez podzielenia na partie.** `COPY` importuje dane jako zwykłe zapytania CQL — przy pliku rzędu dziesiątek milionów wierszy potrafi to trwać godzinami i obciążać klaster w sposób nieproporcjonalny do wolumenu danych; dla takich wolumenów sięgamy po `sstableloader`.

**Powiązane:** CQL: zapytania i operacje na danych · Architektura Cassandry: pierścień, węzeł, replikacja · Backup i odtwarzanie · psql: klient wiersza poleceń

**Źródła:** dokumentacja Apache Cassandra, rozdz. „cqlsh" · `cqlsh --help`

---

Revision #43
Created 2026-09-24 23:34:51 UTC by Tomasz Konieczny
Updated 2026-10-26 14:13:07 UTC by Tomasz Konieczny