ClickHouse Cheat Sheet

Quick Start Commands

Connection

```
# Connect via clickhouse-client
clickhouse-client --host localhost --port 9000 --user default

# Connect via HTTP
curl 'http://localhost:8123/' --data-binary "SELECT version()"

# Connect with password
clickhouse-client --host localhost --port 9000 --user myuser --password
mypass
```

Basic Database Operations

```
-- Show databases
SHOW DATABASES;

-- Create database
CREATE DATABASE my_db;

-- Use database
USE my_db;

-- Drop database
DROP DATABASE my_db;
```

Essential Configuration

Docker Compose Setup

```
version: '3.8'
services:
  clickhouse:
    image: clickhouse/clickhouse-server:latest
    container_name: clickhouse-server
    ports:
      - "8123:8123"  # HTTP
      - "9000:9000"  # Native TCP
    volumes:
      - ./data:/var/lib/clickhouse
      - ./logs:/var/log/clickhouse-server
```

```
      - ./config:/etc/clickhouse-server/config.d
    environment:
      - CLICKHOUSE_USER=default
      - CLICKHOUSE_PASSWORD=
      - CLICKHOUSE_DEFAULT_ACCESS_MANAGEMENT=1
    ulimits:
      nofile:
        soft: 262144
        hard: 262144
```

Critical Server Settings (`config.xml`)

```
<clickhouse>
    <!-- Memory -->

<max_server_memory_usage_to_ram_ratio>0.9</max_server_memory_usage_to_ram_
ratio>
    <max_memory_usage>10000000000</max_memory_usage>

    <!-- Connections -->
    <max_concurrent_queries>100</max_concurrent_queries>
    <max_connections>4096</max_connections>

    <!-- Background processing -->
    <background_pool_size>16</background_pool_size>

<background_merges_mutations_concurrency_ratio>2</background_merges_mutati
ons_concurrency_ratio>

    <!-- Compression -->
    <compression>
        <case><method>lz4</method></case>
    </compression>
</clickhouse>
```

Performance Settings

```
-- Memory management
SET max_memory_usage = 10000000000;                    -- 10GB per query
SET max_bytes_before_external_group_by = 20000000000;  -- 20GB before
spilling
SET max_bytes_before_external_sort = 20000000000;      -- 20GB before
spilling

-- Query optimization
SET max_threads = 8;                                   -- Parallel threads
SET max_execution_time = 3600;                         -- 1 hour timeout
SET optimize_read_in_order = 1;                        -- Read
optimization
```

```
SET optimize_aggregation_in_order = 1;                -- GROUP BY
optimization

-- Insert optimization
SET max_insert_block_size = 1048576;                  -- 1M rows per
block
SET async_insert = 1;                                 -- Enable async
inserts
SET async_insert_timeout_ms = 200;                    -- 200ms timeout

-- JOIN optimization
SET join_algorithm = 'hash';                          -- Hash join
(default)
SET max_bytes_in_join = 1000000000;                   -- 1GB join limit
SET join_use_nulls = 1;                               -- Use NULL for
non-matches
```

Data Types Quick Reference

Numeric Types

```
-- Integers
Int8, Int16, Int32, Int64, Int128, Int256             -- Signed integers
UInt8, UInt16, UInt32, UInt64, UInt128, UInt256       -- Unsigned
integers

-- Floating point
Float32, Float64                                      -- IEEE 754
floating point

-- Fixed precision
Decimal32(scale), Decimal64(scale), Decimal128(scale)  -- Fixed decimal
precision
```

String & Text Types

```
String                                                -- Variable-length
UTF-8
FixedString(N)                                       -- Fixed-length (N
bytes)
LowCardinality(String)                               -- Dictionary-
encoded strings
UUID                                                 -- 128-bit UUID
```

Date & Time Types

```
Date                                                    -- YYYY-MM-DD
Date32                                                  -- Extended range
date
DateTime                                                -- Unix timestamp
DateTime('timezone')                                    -- Timezone-aware
timestamp
DateTime64(precision)                                   -- High-precision
timestamp
```

Complex Types

```
Array(T)                                                -- Array of type T
Tuple(T1, T2, ...)                                      -- Fixed-size tuple
Map(K, V)                                               -- Key-value map
Nested(col1 T1, col2 T2, ...)                           -- Nested structure
Enum8('val1'=1, 'val2'=2)                               -- 8-bit enum
Nullable(T)                                             -- Nullable type (use
sparingly)
```

Table Engines

MergeTree Family

```
-- Basic MergeTree
CREATE TABLE events (
    id UInt64,
    timestamp DateTime,
    user_id UInt32,
    event_type String
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(timestamp)
ORDER BY (user_id, timestamp);

-- ReplacingMergeTree (deduplication)
CREATE TABLE user_profiles (
    user_id UInt32,
    name String,
    updated_at DateTime
) ENGINE = ReplacingMergeTree(updated_at)
ORDER BY user_id;

-- CollapsingMergeTree (updates/deletes)
CREATE TABLE user_actions (
    user_id UInt32,
    action String,
    timestamp DateTime,
    sign Int8
) ENGINE = CollapsingMergeTree(sign)
```

```
ORDER BY (user_id, timestamp);

-- SummingMergeTree (pre-aggregation)
CREATE TABLE daily_stats (
    date Date,
    user_id UInt32,
    page_views UInt64,
    clicks UInt64
) ENGINE = SummingMergeTree((page_views, clicks))
PARTITION BY toYYYYMM(date)
ORDER BY (date, user_id);
```

Distributed Tables

```
-- Create distributed table
CREATE TABLE events_distributed AS events
ENGINE = Distributed(cluster_name, database_name, table_name,
sharding_key);
```

Essential SQL Operations

CREATE TABLE Patterns

```
-- Analytics table with partitioning
CREATE TABLE analytics_events (
    event_id UUID,
    timestamp DateTime,
    user_id UInt32,
    session_id String,
    event_type LowCardinality(String),
    properties Map(String, String),

    -- Indexes
    INDEX event_type_idx event_type TYPE bloom_filter GRANULARITY 1,
    INDEX user_id_idx user_id TYPE minmax GRANULARITY 1
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(timestamp)
ORDER BY (user_id, timestamp, event_id)
SETTINGS
    index_granularity = 8192,
    storage_policy = 'default';
```

INSERT Patterns

```
-- Single insert
INSERT INTO events VALUES (1, now(), 1001, 'click');
```

```sql
-- Batch insert
INSERT INTO events VALUES
    (1, now(), 1001, 'click'),
    (2, now(), 1002, 'view'),
    (3, now(), 1003, 'purchase');

-- Insert from SELECT
INSERT INTO events_backup
SELECT * FROM events
WHERE timestamp >= '2024-01-01';

-- Insert with functions
INSERT INTO events
SELECT
    generateUUIDv4(),
    now() - toIntervalDay(number % 30),
    toUInt32(rand() % 10000),
    'generated_event'
FROM numbers(1000000);
```

SELECT Patterns

```sql
-- Basic filtering
SELECT * FROM events
WHERE event_type = 'click'
  AND timestamp >= '2024-01-01'
LIMIT 100;

-- Aggregation
SELECT
    event_type,
    count() as event_count,
    uniq(user_id) as unique_users,
    avg(session_duration) as avg_duration
FROM events
GROUP BY event_type
ORDER BY event_count DESC;

-- Time-based aggregation
SELECT
    toStartOfDay(timestamp) as day,
    count() as daily_events
FROM events
WHERE timestamp >= now() - INTERVAL 7 DAY
GROUP BY day
ORDER BY day;

-- Window functions
SELECT
    user_id,
```

```
    timestamp,
    event_type,
    row_number() OVER (PARTITION BY user_id ORDER BY timestamp) as
event_sequence
FROM events
ORDER BY user_id, timestamp;
```

JOIN Patterns

```
-- INNER JOIN
SELECT
    e.event_type,
    u.username,
    e.timestamp
FROM events e
INNER JOIN users u ON e.user_id = u.user_id
WHERE e.timestamp >= '2024-01-01';

-- LEFT JOIN with aggregation
SELECT
    u.username,
    count(e.event_id) as event_count
FROM users u
LEFT JOIN events e ON u.user_id = e.user_id
GROUP BY u.username
ORDER BY event_count DESC;
```

Indexing & Optimization

Secondary Indexes

```
-- Bloom filter (for equality checks)
ALTER TABLE events ADD INDEX event_type_bloom event_type TYPE bloom_filter
GRANULARITY 1;

-- Set index (for IN queries)
ALTER TABLE events ADD INDEX status_set status TYPE set(0) GRANULARITY 1;

-- MinMax index (for range queries)
ALTER TABLE events ADD INDEX timestamp_minmax timestamp TYPE minmax
GRANULARITY 1;

-- N-gram index (for text search)
ALTER TABLE events ADD INDEX content_ngram content TYPE ngrambf_v1(3, 256,
2, 0) GRANULARITY 1;
```

Materialized Views

```sql
-- Create materialized view for aggregation
CREATE MATERIALIZED VIEW daily_user_stats
ENGINE = SummingMergeTree()
PARTITION BY toYYYYMM(date)
ORDER BY (date, user_id)
AS SELECT
    toDate(timestamp) as date,
    user_id,
    count() as event_count,
    uniq(session_id) as session_count
FROM events
GROUP BY date, user_id;
```

Projections

```sql
-- Add projection for different sort order
ALTER TABLE events
ADD PROJECTION timestamp_projection (
    SELECT * ORDER BY timestamp, user_id
);

-- Materialize projection
ALTER TABLE events MATERIALIZE PROJECTION timestamp_projection;
```

Essential Functions

Date/Time Functions

```sql
now()                                    -- Current timestamp
today()                                  -- Current date
yesterday()                              -- Yesterday's date
toYYYYMM(date)                          -- Extract YYYYMM
toStartOfMonth(date)                     -- First day of month
toStartOfWeek(date)                      -- Start of week
toStartOfDay(date)                       -- Start of day
toStartOfHour(date)                      -- Start of hour
dateDiff('day', date1, date2)           -- Date difference
dateAdd('day', 7, date)                 -- Add days to date
formatDateTime(date, '%Y-%m-%d %H:%M:%S') -- Format date
```

String Functions

```sql
length(string)                           -- String length
lower(string), upper(string)             -- Case conversion
substring(string, start, length)         -- Extract substring
```

```
concat(str1, str2, ...)                 -- Concatenate strings
splitByChar(',', string)                 -- Split string
replaceAll(string, 'old', 'new')        -- Replace all occurrences
trim(string)                            -- Remove whitespace
```

Aggregate Functions

```
count()                                  -- Count rows
count(DISTINCT column)                   -- Count unique values
sum(column)                             -- Sum values
avg(column)                             -- Average
min(column), max(column)                -- Min/max values
uniq(column)                            -- Approximate unique count
uniqExact(column)                       -- Exact unique count
quantile(0.5)(column)                   -- Median (50th percentile)
quantiles(0.25, 0.5, 0.75)(column)     -- Multiple quantiles
groupArray(column)                       -- Collect values into array
```

Array Functions

```
arrayJoin(array)                        -- Expand array to rows
arrayMap(x -> x * 2, array)             -- Apply function to elements
arrayFilter(x -> x > 0, array)         -- Filter array elements
arrayReduce('sum', array)               -- Reduce array
arraySort(array)                        -- Sort array
arrayReverse(array)                     -- Reverse array
has(array, element)                     -- Check if element exists
length(array)                           -- Array length
```

Type Conversion Functions

```
toString(value)                         -- Convert to string
toInt32(value)                          -- Convert to integer
toFloat64(value)                        -- Convert to float
toDate(string)                          -- Convert to date
toDateTime(string)                      -- Convert to datetime
toUUID(string)                          -- Convert to UUID
CAST(value AS Type)                     -- Generic type casting
```

Monitoring & Troubleshooting

System Tables

```sql
-- Current processes
SELECT query_id, user, query, elapsed, memory_usage
FROM system.processes
WHERE query != '';

-- Query log (recent queries)
SELECT
    query_start_time,
    query_duration_ms,
    query,
    read_rows,
    memory_usage
FROM system.query_log
WHERE event_time > now() - INTERVAL 1 HOUR
  AND type = 'QueryFinish'
ORDER BY query_duration_ms DESC
LIMIT 10;

-- Table sizes
SELECT
    database,
    table,
    sum(rows) as total_rows,
    formatReadableSize(sum(bytes_on_disk)) as size
FROM system.parts
WHERE active = 1
GROUP BY database, table
ORDER BY sum(bytes_on_disk) DESC;

-- Parts information
SELECT
    table,
    partition,
    name,
    rows,
    formatReadableSize(bytes_on_disk) as size,
    modification_time
FROM system.parts
WHERE active = 1 AND table = 'my_table'
ORDER BY modification_time DESC;
```

Performance Monitoring

```sql
-- Slow queries
SELECT
    query,
    query_duration_ms,
    read_rows,
    read_bytes,
    memory_usage
```

```
FROM system.query_log
WHERE query_duration_ms > 5000
  AND event_time > now() - INTERVAL 1 DAY
ORDER BY query_duration_ms DESC
LIMIT 20;

-- Memory usage
SELECT
    event_time,
    CurrentMetric_MemoryTracking as memory_usage,
    CurrentMetric_BackgroundPoolTask as background_tasks
FROM system.metric_log
WHERE event_time > now() - INTERVAL 1 HOUR
ORDER BY event_time DESC
LIMIT 100;

-- Merge performance
SELECT
    table,
    count() as merge_count,
    avg(duration_ms) as avg_duration,
    sum(bytes_read_uncompressed) as total_bytes
FROM system.part_log
WHERE event_type = 'MergeParts'
  AND event_time > now() - INTERVAL 1 DAY
GROUP BY table
ORDER BY avg_duration DESC;
```

Health Checks

```
-- Cluster status
SELECT
    'Version' as metric, version() as value
UNION ALL
SELECT 'Uptime', toString(uptime())
UNION ALL
SELECT 'Active Parts', toString(count()) FROM system.parts WHERE active =
1;

-- Replication status (for replicated tables)
SELECT
    database,
    table,
    replica_name,
    log_max_index - log_pointer as replication_lag
FROM system.replicas;

-- Disk usage
SELECT
    name,
    formatReadableSize(free_space) as free_space,
```

```
    formatReadableSize(total_space) as total_space,
    round((free_space / total_space) * 100, 2) as free_percent
FROM system.disks;
```

Common Patterns & Best Practices

Partitioning Strategies

```
-- Monthly partitioning (most common)
PARTITION BY toYYYYMM(timestamp)

-- Daily partitioning (high volume)
PARTITION BY toYYYYMMDD(timestamp)

-- Custom partitioning
PARTITION BY (toYYYYMM(timestamp), user_type)
```

ORDER BY Best Practices

```
-- High to low cardinality
ORDER BY (user_id, timestamp, event_id)  -- Good
ORDER BY (timestamp, user_id, event_id)  -- Less optimal for user queries

-- Include frequently filtered columns
ORDER BY (status, created_date, id)      -- If you often filter by status
```

Compression Settings

```
-- Table with compression codecs
CREATE TABLE compressed_logs (
    timestamp DateTime CODEC(Delta, LZ4),
    user_id UInt32 CODEC(LZ4),
    message String CODEC(ZSTD(3)),
    level Enum8('DEBUG'=1, 'INFO'=2, 'ERROR'=3) CODEC(LZ4)
) ENGINE = MergeTree()
ORDER BY (timestamp, user_id);
```

Batch Processing

```
-- Optimal settings for batch inserts
SET max_insert_block_size = 1048576;
SET async_insert = 1;
SET async_insert_timeout_ms = 200;
```

```
-- Generate test data
INSERT INTO events
SELECT
    generateUUIDv4(),
    now() - toIntervalSecond(number),
    toUInt32(rand() % 10000),
    ['click', 'view', 'purchase'][rand() % 3 + 1]
FROM numbers(1000000);
```

Troubleshooting Commands

Query Optimization

```
-- Explain query plan
EXPLAIN SELECT * FROM events WHERE user_id = 1001;

-- Explain with pipeline
EXPLAIN PIPELINE SELECT count() FROM events GROUP BY user_id;

-- Check if indexes are used
SELECT * FROM system.query_log
WHERE query LIKE '%your_query%'
  AND ProfileEvents['SelectedMarks'] > 0;
```

Memory Issues

```
-- Check memory usage by query
SELECT
    query_id,
    memory_usage,
    peak_memory_usage,
    query
FROM system.query_log
WHERE memory_usage > 1000000000  -- > 1GB
ORDER BY memory_usage DESC;

-- Current memory usage
SELECT formatReadableSize(sum(memory_usage)) as total_memory
FROM system.processes;
```

Performance Tuning

```
-- Settings for different workloads

-- OLAP/Analytics workload
SET max_memory_usage = 20000000000;
```

```
SET max_threads = 16;
SET optimize_read_in_order = 1;
SET compile_expressions = 1;

-- High-throughput inserts
SET max_insert_block_size = 1048576;
SET async_insert = 1;
SET max_insert_threads = 8;

-- Memory-constrained environment
SET max_memory_usage = 2000000000;
SET max_bytes_before_external_group_by = 1000000000;
SET join_algorithm = 'partial_merge';
```

Quick Reference Commands

Administrative

```
SHOW TABLES;                              -- List tables
DESCRIBE table_name;                       -- Table schema
SHOW CREATE TABLE table_name;             -- Table DDL
OPTIMIZE TABLE table_name FINAL;          -- Force merge
TRUNCATE TABLE table_name;                -- Delete all data
DROP TABLE table_name;                    -- Delete table
```

Useful Shortcuts

```
-- Current settings
SELECT name, value FROM system.settings WHERE changed = 1;

-- Table statistics
SELECT count(), min(timestamp), max(timestamp) FROM events;

-- Sample data
SELECT * FROM events SAMPLE 0.1 LIMIT 10;  -- 10% sample

-- Check data types
SELECT name, type FROM system.columns WHERE table = 'events';
```

This cheat sheet covers the most commonly used ClickHouse operations and configurations. Keep it handy for quick reference during development and operations.