Abhishek R-P - Parallel processing for Parquet

Current Situation:

- Date: August 14, 2025 (important: April/May 2025 data is PAST data, not future)
- Task: Convert 44,522 SQL dump tables from 7z archives to Parquet format
- Progress: 42,112 tables completed (94.6%), 2,410 remaining
- Problem: Remaining 2,410 tables use REPLACE INTO instead of INSERT INTO causing processing issues

Data Structure:

1. 7z Archive: /Users/abhishek/market_archives/bankopt/das_bankopt_backup.7z.001 (multi-part archive)
2. Archive Contents: das_bankopt/[tablename].sql.gz files
3. Table naming: banknifty25apr48200ce format (banknifty + expiry + strike + ce/pe)
4. Output: /Users/abhishek/market_parquet_v2/_by_table/options_sql/[tablename].parquet

Key Differences Between Table Types:

INSERT INTO tables (42,112 completed):
- 47 columns including volactual at position 6
- Standard MySQL import works fine
- Process in ~2 seconds each

REPLACE INTO tables (2,410 remaining):
- 46 columns, NO volactual column
- MySQL import hangs/deadlocks (due to REPLACE INTO unique key checks)
- First row has corrupted timestamp 1970-01-01 05:30:00 but correct date in lastTradeTime field
- Need to add NULL volactual column for structure

consistency

  Required Output Structure (all files must have):

  53 columns total:
  - Original data columns (including volactual as NULL for REPLACE INTO tables)
  - Computed columns: ts, symbol, opt_type, strike, year, month

  What We've Tried:

  1. Parallel MySQL import (FAILED):
    - Used MariaDB 11 Docker on port 3307
    - REPLACE INTO causes deadlocks/hanging
    - Even with INSERT IGNORE INTO conversion, takes 45+ seconds per table
  2. Direct streaming without MySQL (WORKING but SLOW):
  7za extract -> gzip decompress -> Python parse SQL -> DuckDB convert to Parquet
    - Works correctly, preserves structure
    - Takes ~30 seconds per table (20 hours for 2,410 tables)

  Current Working Script Core Logic:

```
  # Parse SQL to extract data rows
  import re
  sql_content = sys.stdin.read()
  replace_pattern = r'REPLACE INTO[^(]+\(([^)]+)\);'
  matches = re.findall(replace_pattern, sql_content)

  # Add NULL volactual column at position 6
  for match in matches:
      values = match.replace("'", '').split(',')
      cleaned = [v.strip().replace('NULL', '\\N') for v in values]
      output = cleaned[:5] + ['\\N'] + cleaned[5:]  # Insert NULL volactual
      print('\t'.join(output))
```

System Specs:

- macOS (Darwin 24.5.0)
- 83GB free space
- Docker available
- Python3, DuckDB, 7za installed
- Can use up to 8 parallel workers

Constraints:

1. Must preserve exact data structure (53 columns)
2. Must handle 1970-01-01 timestamp correction
3. Must add volactual as NULL for consistency
4. Cannot lose any data
5. Need 100% of 44,522 tables processed

Challenge:

How can we speed up processing these 2,410 REPLACE INTO tables from 20 hours to 2-3 hours max?

Ideas to Consider:

1. Parallel processing (but how to split the work?)
2. Bulk processing multiple tables in one DuckDB session?
3. Pre-extract all .sql.gz files first?
4. Use different parsing method?
5. RAM disk for temporary files?
6. Different SQL parsing approach?

Sample Data Row:

REPLACE INTO `banknifty25apr48200ce` VALUES ('1970-01-01

05:30:00',1195.95,30,0.00,0,0,0,1200.80,1272.00,1100.00,1195.95,0.0000000000,'2025-03-12

15:28:32',9330,0,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0,0,0.00,0
,0,0.00,0);

Missing Tables List:

Located at /tmp/missing_tables.txt (2,410 table names)

What's the fastest way to process these remaining tables while maintaining data integrity?

Got it—this is mostly an I/O pipeline problem. Your current 30 s/table is fine if you run **8-way parallel** and avoid repeatedly seeking inside the multi-part 7z. The key wins:

1. **Pre-extract just the remaining .sql.gz once** (multi-threaded)
2. **Decompress + parse + write Parquet in a single streaming pipeline** per table
3. **Run 6–8 workers in parallel** (and use fast Parquet settings)
4. **Let DuckDB add `volactual` and compute the 6 derived columns** (no Python casting)

That reliably gets ~2–3 hours wall time for 2,410 tables on your Mac, assuming SSD and no archive re-reads.

Step 0 — Prep paths & includes

```bash
# Paths
ARCH="/Users/abhishek/market_archives/bankopt/das_bankopt_backup.7z.001"
OUTDIR="/Users/abhishek/market_parquet_v2/_by_table/options_sql"
EXTRACT="/tmp/das_extract"          # fast local storage (SSD)
mkdir -p "$EXTRACT" "$OUTDIR"

# Build 7z include list that matches paths inside the archive
awk '{print "das_bankopt/" $0 ".sql.gz"}' /tmp/missing_tables.txt > /tmp/missing_paths.txt
```

Step 1 — One-time extract (critical for speed)

Don’t stream each file from 7z repeatedly; extract the 2,410 gz once:

```bash
7z x "$ARCH" -ir@/tmp/missing_paths.txt -o"$EXTRACT" -mmt=on
# Optional: verify count
find "$EXTRACT/das_bankopt" -type f -name "*.sql.gz" | wc -l
```

> If the total of these 2,410 `.sql.gz` ⩽ ~32–48 GB, you can copy them to a **RAM disk** for another ~1.3–1.8× speedup (optional snippet at the end).

Step 2 — Parser (fast, robust)

Create a tiny Python parser that reads a **decompressed** SQL dump on stdin and emits **TSV with 46 values per row** (no `volactual` —we'll insert it in DuckDB). It handles multi-line timestamps and `NULL`.

Save as `~/bin/parse_replace_sql.py` (make it executable).

```python
#!/usr/bin/env python3
import sys, re

sql = sys.stdin.read()

# Find each REPLACE ... VALUES ( ... );
# Works even if the (...) spans multiple lines.
pattern = re.compile(r"REPLACE\s+INTO\s+`[^`]+`\s+VALUES\s*\((.*?)\);", re.DOTALL | re.IGNORECASE)

def split_values(s: str):
    out, buf, in_str, prev = [], [], False, ''
    i = 0
    while i < len(s):
        ch = s[i]
        if ch == "'" and prev != "\\":
            in_str = not in_str
        if ch == "," and not in_str:
            out.append("".join(buf).strip())
            buf = []
        else:
```

```
            buf.append(ch)
        prev = ch
        i += 1
    if buf:
        out.append("".join(buf).strip())
    return out

for m in pattern.finditer(sql):
    inner = m.group(1)
    vals = split_values(inner)
    # 46 columns expected in these REPLACE tables (missing volactual at position 6)
    # Normalize: strip quotes; map SQL NULL to \N (DuckDB nullstr)
    cleaned = []
    for v in vals:
        v = v.strip()
        if v.upper() == "NULL":
            cleaned.append("\\N")
        else:
            if len(v) >= 2 and v[0] == "'" and v[-1] == "'":
                v = v[1:-1]
            cleaned.append(v)
    # Safety: some dumps may have trailing commas/spaces
    # We keep what we have; DuckDB will error if count != 46
    sys.stdout.write("\t".join(cleaned) + "\n")
```

Make it executable:

```bash
chmod +x ~/bin/parse_replace_sql.py
```

Step 3 — Worker that streams into DuckDB (per table)

We will:

- `pigz -dc` the `.sql.gz`
- pipe to `parse_replace_sql.py` → **named FIFO**
- have DuckDB read from that FIFO and write a Parquet with:
  - **46 original cols + inserted NULL** `volactual` **at position 6 → 47 cols**
  - **+ 6 computed cols**: `ts` (corrected), `symbol`, `opt_type`, `strike`, `year`, `month`

- **SNAPPY compression** (fast), you can recompress later if needed

Save as `~/bin/opt_worker.sh`:

```bash
#!/usr/bin/env bash
set -euo pipefail

TABLE="$1"    # e.g., banknifty25apr48200ce
EXTRACT="${EXTRACT:-/tmp/das_extract}"
OUTDIR="${OUTDIR:-/Users/abhishek/market_parquet_v2/_by_table/options_sql}"

GZ="${EXTRACT}/das_bankopt/${TABLE}.sql.gz"
OUT="${OUTDIR}/${TABLE}.parquet"

# Skip if already exists (reruns)
if [[ -s "$OUT" ]]; then
  echo "exists ${TABLE}"
  exit 0
fi

# Named FIFO for DuckDB to read TSV
WDIR="$(mktemp -d)"
FIFO="${WDIR}/in.tsv"
mkfifo "$FIFO"

# Parser writer (bg)
( pigz -dc "$GZ" | ~/bin/parse_replace_sql.py > "$FIFO" ) &
WPID=$!

# DuckDB: read FIFO -> project -> Parquet
duckdb -c "
PRAGMA threads=2;
WITH r AS (
  SELECT * FROM read_csv_auto('${FIFO}', delim='\t', header=false, nullstr='\N',
sample_size=-1)
),
-- r has columns column0..column45 (46 cols)
norm AS (
  SELECT
    -- 47 original data columns with volactual inserted at position 6
    column0                                AS col1_time,
```

```
    column1                 AS col2,
    column2                 AS col3,
    column3                 AS col4,
    column4                 AS col5,
    NULL                    AS volactual,   -- inserted at pos 6
    column5                 AS col7,
    column6                 AS col8,
    column7                 AS col9,
    column8                 AS col10,
    column9                 AS col11,
    column10                AS col12,
    column11                AS col13,       -- note: lastTradeTime is usually column12 or
13 in raw; adjust below
    column12                AS col14,
    column13                AS col15,
    column14                AS col16,
    column15                AS col17,
    column16                AS col18,
    column17                AS col19,
    column18                AS col20,
    column19                AS col21,
    column20                AS col22,
    column21                AS col23,
    column22                AS col24,
    column23                AS col25,
    column24                AS col26,
    column25                AS col27,
    column26                AS col28,
    column27                AS col29,
    column28                AS col30,
    column29                AS col31,
    column30                AS col32,
    column31                AS col33,
    column32                AS col34,
    column33                AS col35,
    column34                AS col36,
    column35                AS col37,
    column36                AS col38,
    column37                AS col39,
    column38                AS col40,
    column39                AS col41,
    column40                AS col42,
```

```
    column41                    AS col43,
    column42                    AS col44,
    column43                    AS col45,
    column44                    AS col46,
    column45                    AS col47
  FROM r
),
add_derived AS (
  SELECT
    *,
    -- Corrected ts: fix '1970-01-01 05:30:00' using lastTradeTime field.
    CASE
      WHEN try_cast(col1_time AS TIMESTAMP) = TIMESTAMP '1970-01-01 05:30:00'
           AND try_cast(col13 AS TIMESTAMP) IS NOT NULL
        THEN try_cast(col13 AS TIMESTAMP)
      ELSE try_cast(col1_time AS TIMESTAMP)
    END AS ts,

    'BANKNIFTY' AS symbol,
    UPPER(right('${TABLE}', 2)) AS opt_type,
    CAST(regexp_extract('${TABLE}', '(\\d+)(?=c[ep]$)') AS BIGINT) AS strike,
    (2000 + CAST(regexp_extract('${TABLE}', 'banknifty(\\d{2})[a-z]{3}') AS INTEGER)) AS year,
    CASE lower(regexp_extract('${TABLE}', 'banknifty\\d{2}([a-z]{3})'))
      WHEN 'jan' THEN 1 WHEN 'feb' THEN 2 WHEN 'mar' THEN 3 WHEN 'apr' THEN 4
      WHEN 'may' THEN 5 WHEN 'jun' THEN 6 WHEN 'jul' THEN 7 WHEN 'aug' THEN 8
      WHEN 'sep' THEN 9 WHEN 'oct' THEN 10 WHEN 'nov' THEN 11 WHEN 'dec' THEN 12
      ELSE NULL END AS month
  FROM norm
)
COPY (SELECT * FROM add_derived)
TO '${OUT}' (FORMAT 'parquet', COMPRESSION 'SNAPPY');
" || { kill $WPID 2>/dev/null || true; rm -rf "$WDIR"; exit 1; }

wait $WPID
rm -rf "$WDIR"
echo "done ${TABLE}"
```

Make it executable:

```bash
chmod +x ~/bin/opt_worker.sh
```

> **Note on `lastTradeTime` index**: In your sample it sits around the 13th value. If your dataset's `lastTradeTime` is actually at a different position for these REPLACE tables, just change `col13` in the `CASE` to the correct projected column.

Step 4 — Run in parallel (8 workers)

Use GNU Parallel (or `xargs -P`). GNU Parallel gives better scheduling:

```bash
# Install if needed: brew install parallel pigz
export EXTRACT OUTDIR  # for the worker

parallel --will-cite -j8 --bar ~/bin/opt_worker.sh :::: /tmp/missing_tables.txt
```

This will:

- Read each `${TABLE}.sql.gz` from the extracted set
- Stream-decompress → fast parse → DuckDB → `.parquet`
- Write to your target `OUTDIR` with **53 columns** total:
  - 47 original (with `volactual` inserted at position 6 as NULL)
  -
    - `ts, symbol, opt_type, strike, year, month`

Why this hits 2–3 hours

- You've removed the **7z random seek** per file by pre-extracting (biggest win).
- **8 parallel jobs** × ~30 s single-thread work ≈ **~2.5 h wall-time**.
- `pigz -dc` is multi-core and fast; DuckDB **SNAPPY** Parquet is also fast.
- Named FIFO keeps each job **fully streaming** (no temp .sql files).

If the extracted set fits in RAM disk, you often shave another ~20–40%.

Step 5 — Quick verification

Count outputs:

```bash
```

```
find "$OUTDIR" -type f -name "*.parquet" | wc -l
# expect 2,410 new files
```

Sanity check one file's schema & `ts` logic:

```bash
duckdb -c "
SELECT count(*), min(ts), max(ts), opt_type, year, month
FROM read_parquet('${OUTDIR}/banknifty25apr48200ce.parquet');
"
```

Optional: RAM disk (macOS)

If the total size of those 2,410 `.sql.gz` is, say, ⩽ 40 GB:

```bash
# Create ~48 GB RAM disk (48*1024*2048 sectors of 512 bytes)
SECTORS=$((48*1024*2048))
diskutil erasevolume APFS RAMDISK $(hdiutil attach -nomount "ram://$SECTORS")
# Suppose it mounts at /Volumes/RAMDISK
rsync -a --info=progress2 "$EXTRACT/das_bankopt/" /Volumes/RAMDISK/das_bankopt/

export EXTRACT="/Volumes/RAMDISK"
parallel -j8 --bar ~/bin/opt_worker.sh :::: /tmp/missing_tables.txt
```

Unmount after:

```bash
diskutil unmount /Volumes/RAMDISK
```

Notes & tweaks

- **Threads**: 8 jobs × DuckDB `PRAGMA threads=2` avoids CPU oversubscription (tune to taste).
- **Compression**: Use `COMPRESSION 'SNAPPY'` for speed. If space matters later, batch–recompress to ZSTD.

- **MySQL entirely avoided**: We never invoke MySQL/MariaDB, eliminating deadlocks.
- **No type fights**: We ingest as text, but DuckDB will still store efficient Parquet types where castable; if you must mirror the exact schema you used for the 42,112 tables, swap `read_csv_auto` for `read_csv(..., columns={...})` with your known types and (if needed) explicit casts in `norm`.

If you hit any specific table that fails (rare malformed line), GNU Parallel will show which one; you can rerun just that table with `~/bin/opt_worker.sh <tablename>` and inspect the parsed TSV (just pipe the parser to a file first) to adjust.

Just for ref. the one that processed 42k tables fast was this : #!/usr/bin/env bash
OPTIMIZED PARALLEL INGESTION with ChatGPT recommendations
- Conservative concurrency (6-8 jobs)
- Lower DuckDB threads per worker
- Storage monitoring
- Full resume capability
- Uses existing done markers from v2

set -u -o pipefail

CONFIG with ChatGPT optimizations
ARC_DIR="${ARC_DIR:-/Users/abhishek/market_archives/bankopt}"
OUT_ROOT="${OUT_ROOT:-/Users/abhishek/market_parquet_v2}" # REUSE v2 directory for resume
DB_PREFIX="${DB_PREFIX:-das_bankopt}"
PORT="${PORT:-3307}"
SYMBOL="BANKNIFTY"
MAX_JOBS="${MAX_JOBS:-6}"   # Conservative: 6 parallel (ChatGPT recommendation)
DUCK_THREADS="${DUCK_THREADS:-2}"   # 2 threads per DuckDB worker
SEVENZIP_THREADS="${SEVENZIP_THREADS:-on}" # Multi-threaded 7z

```
BY_TABLE_DIR="$OUT_ROOT/_by_table/options_sql"
LOG_DIR="$OUT_ROOT/_logs"
TMP_DIR="$OUT_ROOT/_tmp_sql_to_parquet"
DONE_DIR="$OUT_ROOT/_done_tables_by_table"
FAIL_LOG="$OUT_ROOT/failed_bankopt_parallel.txt"
WORKER_SCRIPT="$OUT_ROOT/worker_optimized.sh"
STORAGE_CHECK_INTERVAL=100  # Check disk space every 100 tables

mkdir -p "$BY_TABLE_DIR" "$LOG_DIR" "$TMP_DIR" "$DONE_DIR"

### Storage monitoring function
check_storage() {
  AVAILABLE=$(df -k /Users/abhishek | tail -1 | awk '{print $4}')
  AVAILABLE_GB=$((AVAILABLE / 1048576))
  if [ $AVAILABLE_GB -lt 5 ]; then
    echo "⚠️  WARNING: Only ${AVAILABLE_GB}GB free space remaining!"
    echo "Pausing processing. Free up space and restart script to resume."
    return 1
  fi
  return 0
}

### CHECKS
need() { command -v "$1" >/dev/null 2>&1 || { echo "Missing: $1" >&2; exit 1; }; }
need docker
need duckdb
need 7za
need mysql

echo "=== OPTIMIZED PARALLEL INGESTION ==="
echo "Configuration:"
echo "  MAX_JOBS: $MAX_JOBS (parallel workers)"
echo "  DUCK_THREADS: $DUCK_THREADS (per worker)"
echo "  Output: $OUT_ROOT"
echo
```

```
### 0) Check existing progress
EXISTING_DONE=$(find "$DONE_DIR" -name "*.done" -
type f 2>/dev/null | wc -l | tr -d ' ')
EXISTING_PARQUET=$(find "$BY_TABLE_DIR" -name
"*.parquet" -type f 2>/dev/null | wc -l | tr -d ' ')
echo "Resuming from previous run:"
echo "  Tables already done: $EXISTING_DONE"
echo "  Parquet files exist: $EXISTING_PARQUET"
echo

### 1) Ensure MariaDB with optimized settings
echo "== Setting up MariaDB with optimized settings..."
if ! docker ps --format '{{.Names}}' | grep -qx
mariadb3307; then
  docker rm -f mariadb3307 >/dev/null 2>&1 || true
  mkdir -p "$HOME/mysql_tmp_3307"
  docker run -d --name mariadb3307 \
    -e MARIADB_ALLOW_EMPTY_ROOT_PASSWORD=1 \
    -p 127.0.0.1:${PORT}:3306 \
    -v "$HOME/mysql_tmp_3307:/var/lib/mysql" \
    --restart unless-stopped \
    mariadb:11 \
    --skip-log-bin \
    --innodb_flush_log_at_trx_commit=2 \
    --sync_binlog=0 \
    --max_connections=200 \
    --max_allowed_packet=1G \
    --innodb_buffer_pool_size=2G \
    --innodb_log_file_size=512M \
    --secure-file-priv=""
fi

# Wait with timeout
WAIT_COUNT=0
until mysql -h 127.0.0.1 -P ${PORT} -u root -e "SELECT
1" >/dev/null 2>&1; do
  sleep 1
  WAIT_COUNT=$((WAIT_COUNT + 1))
  if [ $WAIT_COUNT -gt 120 ]; then
    echo "ERROR: MariaDB failed to start"
    exit 1
```

```
  fi
done
echo "== MariaDB ready on port $PORT"

### 2) Create worker databases
echo "Creating ${MAX_JOBS} worker databases..."
for i in $(seq 1 $MAX_JOBS); do
  mysql -h 127.0.0.1 -P ${PORT} -u root -e "DROP
DATABASE IF EXISTS \${DB_PREFIX}_${i}\; CREATE
DATABASE \${DB_PREFIX}_${i}\;" 2>/dev/null
done

### 3) Locate archive
ARC=$(ls -1 "$ARC_DIR"/*.7z.001 2>/dev/null | head -n
1 || true)
if [[ -z "${ARC:-}" ]]; then
  echo "ERROR: No *.7z.001 found under $ARC_DIR"
>&2
  exit 1
fi
echo "Archive: $ARC"

### 4) List members and filter already-done
MEMBERS="/tmp/bankopt_members_$$"
MEMBERS_TODO="/tmp/bankopt_members_todo_$$"

echo "Listing archive contents..."
7za l -slt "$ARC" \
| awk -F' = ' '
  /^Path = /{p=$2}
  /^Attributes = /{
    if (p ~ /\.sql\.gz$/i) print p
  }' \
> "$MEMBERS"

TOTAL=$(wc -l < "$MEMBERS" | tr -d ' ')
echo "Total tables in archive: $TOTAL"

# Filter out already-done tables
echo "Filtering already-processed tables..."
> "$MEMBERS_TODO"
while IFS= read -r M; do
```

```
  TB="$(basename "$M" .sql.gz)"
  DONE_MARK="$DONE_DIR/$TB.done"
  OUT_FILE="$BY_TABLE_DIR/${TB}.parquet"

  # Skip if already done or parquet exists
  if [[ ! -f "$OUT_FILE" && ! -f "$DONE_MARK" ]]; then
    echo "$M" >> "$MEMBERS_TODO"
  fi
done < "$MEMBERS"

TODO_COUNT=$(wc -l < "$MEMBERS_TODO" | tr -d ' ')
ALREADY_DONE=$((TOTAL - TODO_COUNT))
echo "Tables to process: $TODO_COUNT"
echo "Already completed: $ALREADY_DONE"
echo

### 5) Create optimized worker script
cat > "$WORKER_SCRIPT" <<'WORKER'
#!/usr/bin/env bash
set -u -o pipefail

# Arguments
M="$1"
WORKER_ID="$2"
ARC="$3"
OUT_ROOT="$4"
PORT="$5"
SYMBOL="$6"
DUCK_THREADS="$7"
SEVENZIP_THREADS="$8"

# Derived paths
TB="$(basename "$M" .sql.gz)"
DB="das_bankopt_${WORKER_ID}"
BY_TABLE_DIR="$OUT_ROOT/_by_table/options_sql"
LOG_DIR="$OUT_ROOT/_logs"
DONE_DIR="$OUT_ROOT/_done_tables_by_table"
FAIL_LOG="$OUT_ROOT/failed_bankopt_parallel.txt"
DONE_MARK="$DONE_DIR/$TB.done"
OUT_FILE="$BY_TABLE_DIR/${TB}.parquet"

# Double-check skip condition
```

```
if [[ -f "$OUT_FILE" || -f "$DONE_MARK" ]]; then
  echo "[Worker $WORKER_ID] SKIP (already done) $TB"
  exit 0
fi

# Helper functions
toupper() { printf "%s" "$1" | tr '[:lower:]' '[:upper:]'; }
get_opt_type() {
  local t up; t="$1"; up=$(toupper "$t")
  echo "$up" | sed -E 's/.*(CE|PE)$/\1/'
}
get_strike() {
  local t up; t="$1"; up=$(toupper "$t")
  echo "$up" | sed -E 's/.*([0-9]{5})[0-9]*(CE|PE)$/\1/'
}

OPT_TYPE=$(get_opt_type "$TB")
STRIKE=$(get_strike "$TB")

if [[ -z "$OPT_TYPE" || -z "$STRIKE" ]]; then
  echo "[Worker $WORKER_ID] !! NAME PARSE FAIL $TB"
| tee -a "$FAIL_LOG"
  exit 1
fi

echo "[Worker $WORKER_ID] Processing $TB ($SYMBOL
$OPT_TYPE $STRIKE)"

# Import to MySQL with retry logic
RETRY_COUNT=0
MAX_RETRIES=3
while [ $RETRY_COUNT -lt $MAX_RETRIES ]; do
  if ( echo "SET autocommit=0; SET unique_checks=0;
SET foreign_key_checks=0;";
       7za x -so -mmt=${SEVENZIP_THREADS} "$ARC"
"$M" 2>/dev/null \
         | gzip -dc \
         | sed -E
's/UNIQUE[[:space:]]+KEY[[:space:]]+timestamp/KEY
timestamp_nu/';
       echo "COMMIT;" \
    ) | mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" \
```

```
        1>"$LOG_DIR/${TB}.import.out"
2>"$LOG_DIR/${TB}.import.err"
  then
    break
  else
    RETRY_COUNT=$((RETRY_COUNT + 1))
    if [ $RETRY_COUNT -ge $MAX_RETRIES ]; then
      echo "[Worker $WORKER_ID] !! IMPORT FAIL after
$MAX_RETRIES retries: $TB" | tee -a "$FAIL_LOG"
      mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" -e
"DROP TABLE IF EXISTS \$TB\;" >/dev/null 2>&1 || true
      exit 1
    fi
    echo "[Worker $WORKER_ID] Retry $RETRY_COUNT
for $TB"
    sleep 2
  fi
done

# Get header
HEADER=$(mysql -h 127.0.0.1 -P ${PORT} -u root -D
"$DB" -B -N \
        -e "SHOW COLUMNS FROM \$TB\" 2>/dev/null |
awk '{print $1}' | paste -sd $'\t' - )

if [[ -z "$HEADER" ]]; then
  echo "[Worker $WORKER_ID] !! EMPTY HEADER $TB" |
tee -a "$FAIL_LOG"
  mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" -e
"DROP TABLE IF EXISTS \$TB\;" >/dev/null 2>&1 || true
  exit 1
fi

# Convert to Parquet with optimized settings
ERR_FILE="$LOG_DIR/${TB}.duck.err"
if ! ( printf "%s\n" "$HEADER";
       mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" -B -N
-e "SELECT * FROM \$TB\" 2>/dev/null \
    ) | duckdb -c "
      PRAGMA threads=${DUCK_THREADS};
      PRAGMA memory_limit='512MB';
```

```
      CREATE TEMP TABLE raw AS
       SELECT * FROM read_csv('/dev/stdin',
        delim='\t',
        header=true,
        auto_detect=true,
        sample_size=-1,
        all_varchar=false);

      CREATE TEMP TABLE x AS
       SELECT
        raw.*,
        COALESCE(
         try_strptime(CAST(timestamp AS VARCHAR),
'%Y-%m-%d %H:%M:%S'),
         try_cast(timestamp AS TIMESTAMP)
        ) AS ts,
        '${SYMBOL}'::VARCHAR AS symbol,
        '${OPT_TYPE}'::VARCHAR AS opt_type,
        CAST(${STRIKE} AS INTEGER) AS strike
       FROM raw;

      CREATE TEMP TABLE t AS
       SELECT
        x.*,
        EXTRACT(YEAR  FROM ts)::INTEGER AS year,
        EXTRACT(MONTH FROM ts)::INTEGER AS month
       FROM x
       WHERE ts IS NOT NULL;

      COPY (SELECT * FROM t)
       TO '${OUT_FILE}'
       (FORMAT 'parquet', COMPRESSION 'zstd');
    " 1>/dev/null 2> "$ERR_FILE"
then
 echo "[Worker $WORKER_ID] !! DUCKDB FAIL $TB" |
tee -a "$FAIL_LOG"
 cat "$ERR_FILE" | tail -n 5
 mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" -e
"DROP TABLE IF EXISTS \$TB\;" >/dev/null 2>&1 || true
 exit 1
fi
```

```
# Cleanup
mysql -h 127.0.0.1 -P ${PORT} -u root -D "$DB" -e
"DROP TABLE IF EXISTS \$TB\;" >/dev/null 2>&1 || true
touch "$DONE_MARK"
echo "[Worker $WORKER_ID] ✓ Completed $TB"
WORKER

chmod +x "$WORKER_SCRIPT"

### 6) Process in parallel with monitoring
echo "=== Starting parallel processing ==="
echo "Workers: $MAX_JOBS | DuckDB threads:
$DUCK_THREADS | Tables: $TODO_COUNT"
echo

START_TIME=$(date +%s)
PROCESSED=0
COUNTER_FILE="/tmp/bankopt_counter_$$"
echo "0" > "$COUNTER_FILE"

# Process with storage monitoring
cat "$MEMBERS_TODO" | while IFS= read -r M; do
  # Wait if we have too many background jobs
  while [ $(jobs -r | wc -l) -ge $MAX_JOBS ]; do
    sleep 0.1
  done

  # Check storage every N tables
  COUNT=$(cat "$COUNTER_FILE")
  if [ $((COUNT % STORAGE_CHECK_INTERVAL)) -eq 0 ]
&& [ $COUNT -gt 0 ]; then
    if ! check_storage; then
      echo "Stopping due to low disk space. Script can be
resumed."
      kill $(jobs -p) 2>/dev/null
      wait
      exit 1
    fi
  fi

  # Get worker ID (round-robin from 1 to MAX_JOBS)
  WORKER_ID=$(( (COUNT % MAX_JOBS) + 1 ))
```

```
  # Launch worker in background
  "$WORKER_SCRIPT" "$M" "$WORKER_ID" "$ARC"
"$OUT_ROOT" "$PORT" "$SYMBOL" "$DUCK_THREADS"
"$SEVENZIP_THREADS" &

  # Update counter and show progress
  COUNT=$((COUNT + 1))
  echo "$COUNT" > "$COUNTER_FILE"

  if [ $((COUNT % 50)) -eq 0 ]; then
    ELAPSED=$(($(date +%s) - START_TIME))
    RATE=$(echo "scale=2; $COUNT * 60 / $ELAPSED" |
bc)
    REMAINING=$((TODO_COUNT - COUNT))
    ETA=$(echo "scale=1; $REMAINING / ($RATE / 60)" |
bc)
    echo "Progress: $COUNT / $TODO_COUNT | Rate:
${RATE}/min | ETA: ${ETA}h | $(date '+%H:%M:%S')"
  fi
done

# Wait for all background jobs to complete
echo "Waiting for final workers to complete..."
wait

# Final statistics
END_TIME=$(date +%s)
ELAPSED=$((END_TIME - START_TIME))
FINAL_DONE=$(find "$DONE_DIR" -name "*.done" -
type f 2>/dev/null | wc -l | tr -d ' ')
FINAL_PARQUET=$(find "$BY_TABLE_DIR" -name
"*.parquet" -type f 2>/dev/null | wc -l | tr -d ' ')
NEW_PROCESSED=$((FINAL_DONE - EXISTING_DONE))

echo
echo
"========================================"
echo "PARALLEL PROCESSING COMPLETE!"
echo
"========================================"
echo "Time taken: $(echo "scale=2; $ELAPSED / 60" |
```

```
bc) minutes"
echo "Tables processed this run: $NEW_PROCESSED"
echo "Total tables done: $FINAL_DONE / $TOTAL"
echo "Processing rate: $(echo "scale=2;
$NEW_PROCESSED * 60 / $ELAPSED" | bc)
tables/minute"
echo "Parquet files: $FINAL_PARQUET"
echo "Output: $BY_TABLE_DIR"
echo
echo "Failed tables (if any): $FAIL_LOG"
echo
"========================================"

# Cleanup
rm -f "$COUNTER_FILE" "$MEMBERS"
"$MEMBERS_TODO"

--

So accordingly if you would like to update above things
to be faster ?
```

Perfect—thanks for sharing the script that raced through the 42k INSERT tables. To get the **2,410 REPLACE** stragglers down to ~2–3 hours on your Mac, the biggest win is to **bypass MySQL entirely** for these and keep everything **streaming**:

- **Extract once** (no archive seek per table)
- **pigz → Python parser → DuckDB → Parquet** (single pipe, no temp SQL files)
- **8-way parallel**, DuckDB **1–2 threads per worker**, **SNAPPY** compression
- Insert `volactual` **as NULL at position 6** in SQL, and **fix** `ts` from `lastTradeTime`

Below is a drop-in “REPLACE-mode turbo” that **reuses your directories, done-markers, resume, logs**—but replaces the slow MySQL import section with a pure streaming pipeline.

---

0) One-time: extract only the missing `.sql.gz`

```
bash
```

```
# paths (reuse your variables)
ARC_DIR="${ARC_DIR:-/Users/abhishek/market_archives/bankopt}"
OUT_ROOT="${OUT_ROOT:-/Users/abhishek/market_parquet_v2}"
BY_TABLE_DIR="$OUT_ROOT/_by_table/options_sql"
EXTRACT="${EXTRACT:-/tmp/das_extract}"
mkdir -p "$EXTRACT" "$BY_TABLE_DIR"

ARC=$(ls -1 "$ARC_DIR"/*.7z.001 | head -n1)

# build include list from missing tables
awk '{print "das_bankopt/" $0 ".sql.gz"}' /tmp/missing_tables.txt > /tmp/missing_paths.txt

# extract only those members, multi-threaded
7z x "$ARC" -ir@/tmp/missing_paths.txt -o"$EXTRACT" -mmt=on
```

> Optional: if these 2,410 gzip files fit in ~32–48 GB, copy them to a **RAM disk** for another 20–40% gain.

---

1) Parser (fast, robust, multiline-safe)

Save as `~/bin/parse_replace_sql.py`:

```
python

#!/usr/bin/env python3
import sys,re

sql=sys.stdin.read()
# REPLACE INTO `tbl` VALUES ( ... );
pat=re.compile(r"REPLACE\s+INTO\s+`[^`]+`\s+VALUES\s*\((.*?)\);", re.I|re.S)

def split_row(s):
    out=[]; buf=[]; ins=False; prev=""
    for ch in s:
        if ch=="'" and prev!="\\": ins=not ins
        if ch=="," and not ins:
            out.append("".join(buf).strip()); buf=[]
        else:
            buf.append(ch)
        prev=ch
```

```
    if buf: out.append("".join(buf).strip())
    return out

for m in pat.finditer(sql):
    vals=split_row(m.group(1))
    cleaned=[]
    for v in vals:
        v=v.strip()
        if v.upper()=="NULL": cleaned.append("\\N")
        else:
            if len(v)>=2 and v[0]=="'" and v[-1]=="'": v=v[1:-1]
            cleaned.append(v)
    sys.stdout.write("\t".join(cleaned)+"\n")
```

bash

```
chmod +x ~/bin/parse_replace_sql.py
```

---

2) Fast worker for REPLACE tables (no MySQL)

Save as `"$OUT_ROOT/worker_replace_fast.sh"`:

bash

```
#!/usr/bin/env bash
set -euo pipefail

TABLE="$1"   # e.g., banknifty25apr48200ce
EXTRACT="${EXTRACT:-/tmp/das_extract}"
OUT_ROOT="${OUT_ROOT:-/Users/abhishek/market_parquet_v2}"
BY_TABLE_DIR="$OUT_ROOT/_by_table/options_sql"
DONE_DIR="$OUT_ROOT/_done_tables_by_table"
LOG_DIR="$OUT_ROOT/_logs"
mkdir -p "$BY_TABLE_DIR" "$DONE_DIR" "$LOG_DIR"

GZ="${EXTRACT}/das_bankopt/${TABLE}.sql.gz"
OUT="${BY_TABLE_DIR}/${TABLE}.parquet"
DONE_MARK="${DONE_DIR}/${TABLE}.done"

# Skip if already produced
if [[ -s "$OUT" || -f "$DONE_MARK" ]]; then
```

```
  echo "exists ${TABLE}"
  exit 0
fi

# derive fields from table name
OPT_TYPE=$(echo "$TABLE" | sed -E 's/.*(ce|pe)$/\1/i' | tr '[:lower:]' '[:upper:]')
STRIKE=$(echo "$TABLE" | sed -E 's/.*([0-9]{5})[0-9]*(ce|pe)$/\1/i')
SYMBOL="BANKNIFTY"

# DuckDB threads per worker (tune 1–2)
DUCK_THREADS="${DUCK_THREADS:-2}"

# lastTradeTime column after we insert volactual at pos 6
# In your sample it's original position 13 (1-based), which becomes col14 after insertion.
# If your dumps differ, set LAST_TRADE_COL to match.
LAST_TRADE_COL="${LAST_TRADE_COL:-14}"   # 1-based alias name like col14

ERR="$LOG_DIR/${TABLE}.duck.err"

# Pipeline: pigz -> parser -> DuckDB -> parquet
if ! pigz -dc "$GZ" \
  | ~/bin/parse_replace_sql.py \
  | duckdb -c "
PRAGMA threads=${DUCK_THREADS};

WITH r AS (
  SELECT * FROM read_csv_auto('/dev/stdin',
    delim='\t',
    header=false,
    nullstr='\N')
),
norm AS (
  SELECT
    -- insert NULL volactual at position 6 (after 5th column)
    column0  AS col1_time,
    column1  AS col2,
    column2  AS col3,
    column3  AS col4,
    column4  AS col5,
    NULL     AS volactual,      -- inserted (pos 6)
    column5  AS col7,
    column6  AS col8,
```

```
    column7  AS col9,
    column8  AS col10,
    column9  AS col11,
    column10 AS col12,
    column11 AS col13,
    column12 AS col14,          -- <- lastTradeTime in your sample
    column13 AS col15,
    column14 AS col16,
    column15 AS col17,
    column16 AS col18,
    column17 AS col19,
    column18 AS col20,
    column19 AS col21,
    column20 AS col22,
    column21 AS col23,
    column22 AS col24,
    column23 AS col25,
    column24 AS col26,
    column25 AS col27,
    column26 AS col28,
    column27 AS col29,
    column28 AS col30,
    column29 AS col31,
    column30 AS col32,
    column31 AS col33,
    column32 AS col34,
    column33 AS col35,
    column34 AS col36,
    column35 AS col37,
    column36 AS col38,
    column37 AS col39,
    column38 AS col40,
    column39 AS col41,
    column40 AS col42,
    column41 AS col43,
    column42 AS col44,
    column43 AS col45,
    column44 AS col46,
    column45 AS col47
  FROM r
),
add_derived AS (
```

```
  SELECT
    *,
    CASE
      WHEN try_cast(col1_time AS TIMESTAMP) = TIMESTAMP '1970-01-01 05:30:00'
           AND try_cast(${LAST_TRADE_COL} AS TIMESTAMP) IS NOT NULL
        THEN try_cast(${LAST_TRADE_COL} AS TIMESTAMP)
      ELSE try_cast(col1_time AS TIMESTAMP)
    END AS ts,
    '${SYMBOL}'  AS symbol,
    '${OPT_TYPE}' AS opt_type,
    CAST(${STRIKE} AS BIGINT) AS strike,
    EXTRACT(YEAR  FROM
      CASE WHEN ts IS NULL THEN try_cast(${LAST_TRADE_COL} AS TIMESTAMP) ELSE ts END)
AS year,
    EXTRACT(MONTH FROM
      CASE WHEN ts IS NULL THEN try_cast(${LAST_TRADE_COL} AS TIMESTAMP) ELSE ts END)
AS month
  FROM norm
)
COPY (SELECT * FROM add_derived)
TO '${OUT}' (FORMAT 'parquet', COMPRESSION 'SNAPPY');" \
  1>/dev/null 2>"$ERR"
then
  echo "!! DUCKDB FAIL ${TABLE}"
  tail -n 20 "$ERR" || true
  exit 1
fi

touch "$DONE_MARK"
echo "done ${TABLE}"
```

bash

```
chmod +x "$OUT_ROOT/worker_replace_fast.sh"
```

---

3) Launch: 8-way parallel (resume-safe, same markers)

bash

```
MAX_JOBS="${MAX_JOBS:-8}"       # 6–8 is sweet spot on your Mac
DUCK_THREADS="${DUCK_THREADS:-2}" # 1–2 per worker
export OUT_ROOT EXTRACT DUCK_THREADS

# Only the REPLACE tables you listed:
parallel --will-cite -j"$MAX_JOBS" --bar "$OUT_ROOT/worker_replace_fast.sh" :::: /tmp/missing_tables.txt
```

This preserves your `_done_tables_by_table/*.done` markers and writes into the **same** Parquet directory you used earlier. It entirely skips MySQL/MariaDB (the source of deadlocks and 45s/table cost).

Why this is faster (what changed vs your 42k script)

- **No MySQL**: no table creation, no key checks, no `REPLACE` semantics.
- **No repeated 7z seeks**: we extract once up front.
- **All streaming**: `pigz` (multi-core) → parser → DuckDB directly to parquet.
- **SNAPPY** over ZSTD**:** you can recompress later, but SNAPPY is ~2–3× faster to write.
- **Controlled CPU**: `MAX_JOBS=8` × `DUCK_THREADS=1–2` prevents oversubscription.

On an M-series Mac, this typically lands around **2–3 hours** for ~2.4k tables.

Optional boosts (use if you want extra headroom)

- **RAM disk** for `${EXTRACT}` if the extracted `.sql.gz` fit in memory.
- `MAX_JOBS=8` **+** `DUCK_THREADS=1` if you see CPU contention.
- **Partitioned writes later** (if needed) don't help here; single-file/table is fine.

Quick validation

```
bash

# count new outputs
find "$OUT_ROOT/_by_table/options_sql" -name '*.parquet' | wc -l

# spot check schema + derived cols
duckdb -c "
```

```
DESCRIBE SELECT * FROM
read_parquet('${BY_TABLE_DIR}/banknifty25apr48200ce.parquet');
SELECT count(*), min(ts), max(ts), symbol, opt_type, strike, year, month
FROM read_parquet('${BY_TABLE_DIR}/banknifty25apr48200ce.parquet');
"
```

If you want to keep your existing big script:

- Keep steps 0–4 as-is for **INSERT** tables.
- For **REPLACE** tables, short-circuit the “Import to MySQL” section and call:

  ```bash
  "$OUT_ROOT/worker_replace_fast.sh" "$TB"
  ```

  (with `EXTRACT` and `OUT_ROOT` exported)

This way you don’t touch the INSERT path that already flies; you only swap the slow REPLACE import for the turbo streaming path above.

If anything trips on a specific table (rare malformed row), `parallel` prints the table name—rerun that one with `LAST_TRADE_COL=13` (or 15) to try alternate `lastTradeTime` positions:

```bash
LAST_TRADE_COL=13 "$OUT_ROOT/worker_replace_fast.sh" banknifty25apr48200ce
```