C2. DataQuality — постановка задачи (детализация)

Версия: 1.0
Дата: 2025-10-15
Назначение: автоматические **проверки целостности** и «санитайз» OHLCV-баров для базового ТФ **1m** и производных ТФ **5m/15m/1h**. Обеспечивает детерминированный выход без дубликатов/аномалий, журнал проблем и метрики качества. Все времена — **UTC**.

> **Входы:** сырые минуты (C1 Fetcher) и/или ресемплированные бары (C1 Resampler).
> **Выходы:** те же Parquet-файлы (исправленные/помеченные) + журнал аномалий `issues` и сводный отчёт качества.
> **Интерфейс:** `validate(df) -> (df_clean, issues)` (см. §9).
> **NFR:** пропуски `< 0.01%`, валидные таймстампы, задержка обновления `≤ 1 бар`.
> **DoD:** 100% покрытие правил; CI-фейл при дефектах; метрики качества данных попадают в отчёт (C14).

---

1. Результат

- **Санитайз-каскад** поверх входных OHLCV: выравнивание времени, дедуп, гарантии OHLC-инвариантов, маркировка «дыр» (`is_gap`), флаги аномальных баров (volume/ATR-spike и др.).
- **Журнал аномалий** (тип, серьёзность, что сделано: fix/flag/drop, оригинальные значения).
- **Метрики качества** per `(symbol, tf)` и в агрегате: доля пропусков, частота правил, топ-интервалы проблем, q99.9 по volume/ATR/returns.

---

2. Схемы и расширения

Вход/выход (минимум): `ts:int64, o:float64, h:float64, l:float64, c:float64, v:float64`
Опционально: `t:float64 (turnover)`, `ver:int32`, `is_gap:bool`, `source:str`.

Новые поля (выход): - `dq_flags:int32` — битовая маска правил (см. §4.2).
- `dq_notes:str` (короткий код последнего сработавшего правила).
- (необяз.) `dq_rank:int16` — приоритет важности (CRIT/WARN/INFO→2/1/0).

---

3. Архитектура обработки

Последовательность для каждого `(symbol, tf)`: 1) **Align**: нормализация времени и календаря (UTC, шаг `tf_ms`), удаление будущих баров.
2) **Dedup**: устранение дубликатов по `ts` (см. §4.1).
3) **Sanitize**: гарантии OHLC-инвариантов, отрицательных/NaN/zero-аномалий, мягкие правки (см. §4.3).

4) **Anomaly flagging**: volume/ATR/return-спайки по квантилям/ `z` (см. §5).
5) **Consistency checks**: для derived-ТФ — сверка с 1m (если доступен) (см. §6).
6) **Emit**: запись обратно в Parquet (атомарно) и `issues.parquet` + `quality.json`.

Режимы: `strict` (только флаговать/дропнуть критичное) и `repair_safe` (минимальные безопасные правки + флаг).

---

4. Правила и коды

4.1 Дедуп/время

- **R01_DUP_TS (CRIT):** повторяющийся `ts`. Действие: оставить **последний** (по `ver`/порядку), остальное → `drop`.
- **R02_TS_FUTURE (CRIT):** `ts` в будущем относительно `now_utc - tf_ms`. Действие: `drop`.
- **R03_TS_MISALIGNED (WARN/CRIT):** `ts % tf_ms ≠ 0`. Если `|misalign| ≤ 1s` → округлить к ближайшей границе и `flag`; иначе → `drop`.

4.2 Инварианты и валидность

- **R10_NEG_PRICE (CRIT):** `o|h|l|c ≤ 0`. Действие: `drop` бар или `gapify` (см. §4.3).
- **R11_NEG_VOL (WARN):** `v < 0`. Действие: `set v=0`, `flag`.
- **R12_NAN (CRIT):** NaN/Inf в числовых колонках. Действие: `gapify`.
- **R13_OHLC_ORDER (CRIT):** нарушено: `h ≥ max(o,c)` или `l ≤ min(o,c)` (любое из двух ложно). Действие: `coerce` (см. §4.3) + `flag`.
- **R14_H_LT_L (CRIT):** `h < l`. Действие: `swap_coerce` (см. §4.3) + `flag`.

4.3 Санитайз-действия

- **coerce_OHLC:** `h = max(h, o, c, l)`, `l = min(l, o, c, h)`. Если после правки `h==l` → `gapify`.
- **swap_coerce:** если `h < l`, сначала `h,l = l,h`, затем `coerce_OHLC`.
- **gapify:** заменить бар на «пустой»: `o=h=l=c=prev_close`, `v=0`, `is_gap=true`, `dq_flags|=GAPIFIED`. Применяется при R10/R12 или при некорректных даных без безопасной починки.
- Все правки фиксируются в `issues` с `before/after` значениями (с округлением до 10-4).

---

5. Спайки и статистические флаги

Порог `q=0.999` (99.9-квантиль) по скользящему окну `W` (дефолт 90 дней для 1m, эквивалент в барах для других ТФ). Дополнительно — z-оценка.

- **R20_VOL_SPIKE (WARN):** `v > Q_q(v)` **или** `z_v > z_thr` (дефолт `z_thr=6`).
- **R21_ATR_SPIKE (WARN):** `ATR_n > Q_q(ATR_n)` с `n=14` и/или `z_atr > 6`.
- **R22_RET_SPIKE (WARN):** `|log(c/prev_c)| > Q_q(|logret|)` и/или `z_ret > 6`.
- **R23_ZERO_RUN (INFO/WARN):** серия `v=0` длиной `>Z` (дефолт `Z=5` для 1m) — часто признак простоя/делиста.

Флаги **не изменяют** данные; они маркируют строки и попадают в отчёты.

6. Согласованность производных ТФ (опц.)

Если доступны исходные 1m: - **R30_RESAMPLE_O (WARN):** `o_derived ≠ o_first_min`.
- **R31_RESAMPLE_C (WARN):** `c_derived ≠ c_last_min`.
- **R32_RESAMPLE_HL (WARN):** `h_derived < max(h_min)` ***или*** `l_derived > min(l_min)`.
- **R33_RESAMPLE_VOL (WARN):** `|v_derived − sum(v_min)| > ε`, где `ε = max(1e-6, 1e-6·sum)`.

7. Журнал аномалий `issues`

Схема Parquet/CSV:

```
ts, symbol, tf, rule, severity, action, before_json, after_json, note
```

- `rule∈{R01…}`, `severity∈{INFO,WARN,CRIT}`, `action∈{NONE,FLAG,FIX,DROP,GAPIFY}`.
- `before/after_json` содержат изменённые поля.
- Агрегаты: счётчики по правилам/периодам, топ-интервалы аномалий.

8. Конфигурация (пример)

```yaml
mode: "repair_safe"    # strict | repair_safe
quantiles:
  q: 0.999
  window_days: 90
spikes:
  z_thr: 6.0
  zero_run_len_warn: 5
resample_checks:
  enable: true
  epsilon_rel: 1e-6
sanitizers:
  misalign_tolerance_s: 1
  allow_gapify: true
outputs:
  issues_dir: "/data_qa/issues/"
  quality_report: "/data_qa/quality_{symbol}_{tf}.json"
logging:
  level: INFO
```

9. Интерфейсы

9.1 Python

```
def validate(df: pd.DataFrame, tf: str, cfg: dict) -> tuple[pd.DataFrame,
pd.DataFrame]:
    """Возвращает (df_clean, issues). df_clean сохраняет исходную схему +
dq_* поля."""
```

Вспомогательные:

```
run_quality(symbols: list[str], tfs: list[str], cfg: dict) -> QualitySummary
compute_quantiles(df: pd.DataFrame, cols: list[str], cfg) -> dict
```

9.2 CLI

```
dataquality validate --symbols BTCUSDT,ETHUSDT --tfs 1m,5m,15m,1h
  --mode repair_safe --out /data --issues /data_qa/issues

dataquality report --symbols ALL --tfs 1m,5m,15m,1h --out /reports/
data_quality/
```

---

10. Метрики качества (для C13/C14)

- `gap_pct`, `dup_pct`, `nan_cnt`, `neg_vol_cnt`, `ohcl_fix_cnt`, `gapified_cnt`.
- `vol_spike_rate`, `atr_spike_rate`, `ret_spike_rate`, `zero_run_events`.
- Карты «проблемных» часов/дней (`DOW×HOD`).

---

11. Производительность

- Векторные операции `pandas/pyarrow`; расчёт квантилей — `tdigest`/approx (или «ленивый» пересчёт 1 раз/день).
- Цель: проверка 1 года 1m по 1 тикеру $\leq$ 3 мин на 1 ядре; пиковая память $\leq$ 1 ГБ.

---

12. Тесты и DoD

Юнит-набор (100% покрытие правил): - R01/R02/R03: дубликаты/будущее/мисалайны.
- R10/R11/R12/R13/R14: валидность и OHLC-инварианты + санитайз-действия.
- R20/R21/R22/R23: спайки и нулевые серии.
- R30…R33: согласованность ресемплинга.

Интеграция: - Прогон на $\geq$ 5 тикерах × $\geq$ 1 год; формирование `issues.parquet` и `quality.json`.
- CI-гейт: если `CRIT>0` или `gap_pct>0.01%` — job **FAIL**.
- Сводные метрики попадают в C14-отчёт.

DoD: - Все правила покрыты тестами; CI падает при дефектах; отчёт качества сформирован; доля пропусков `< 0.01%`.

---

13. Ошибки и коды

- `E_SCHEMA` — отсутствуют колонки или неверные типы.
- `E_TIME` — несогласованность календаря после санитайза.
- `E_IO` — ошибка записи `issues`/выходных файлов.
- `E_RUNTIME` — прочие исключения.

---

14. Псевдокод

```
def validate(df, tf, cfg):
  issues = []
  df = align_calendar(df, tf, cfg)
  df, dup_issues = dedup(df)
  issues += dup_issues
  df, inv_issues = sanitize_ohlc(df, cfg)
  issues += inv_issues
  df, spike_issues = flag_spikes(df, cfg)
  issues += spike_issues
  if cfg.resample_checks.enable and tf != '1m':
    ref1m = load_1m_window(df.ts.min(), df.ts.max())
    issues += check_resample_consistency(df, ref1m)
  assert_monotonic(df)
  return df, pd.DataFrame(issues)
```

---

15. Интеграция с остальными компонентами

- **C1:** сразу после бэкфилла/ресемплинга — вызов C2; запись обратно поверх тех же файлов (атомарно) и выпуск `issues`.
- **C3/C5:** читают уже санитайз-данные.
- **C13/C14:** потребляют метрики качества и включают их в мониторинг/отчётность.
- **C18:** CI-шаг `dataquality validate` в пайплайне сборки данных.