Überblick

Dieses Schema bildet zwei eng gekoppelte Bereiche ab:

1) **Job-Datenmodell** (Jobs + Raw-Payloads) - `public.jobs` = zentrale Stammtabelle der Jobs (Metadaten, Status, Sichtbarkeit über Zeit) - `public.job_payloads` = 1:1-Detail-/Raw-Payload pro Job (JSONB), technisch über denselben `job_pk` verknüpft - `public.jobs_without_payload` (View) = Qualitäts-/Backfill-Liste: Jobs, denen der Payload fehlt

2) **Scraping-Monitoring** (Company-Konfiguration + Run-Tracking + Tagesaggregation + Alerts) - `public.scrape_companies` = Steuerung, welche Firmen aktiv gescraped werden - `public.scrape_runs` = protokolliert jeden Scrape-Run pro Firma (Status, Metriken, Fehler, HTTP-Counts) - `public.daily_summary_v` (View) = zentrale Tagesaggregation (30-Tage Raster) inkl. Baselines/Flags/Status - `public.daily_summary_looker_v` (View) = BI/Looker-friendly Variante (zusätzliche 0/1 Flags) - `public.alert_candidates_v` (View) = Filter auf "auffällige Tage/Firmen", die potenziell Alerts auslösen

---

Kern-Entitäten und Beziehungen

1) Job-Entitäten

`public.jobs`

Rolle: Stammdatensatz pro Job (was/wo/welche Firma/Status/Zeiten).

Primary Key: - `job_pk uuid` (PK, default `uuid_generate_v4()`)

Wichtige Felder (Auszug): - Identität: `source`, `job_id` (fachliche Job-ID je Quelle) - Inhalte/Metadaten: `title`, `location_text`, `country`, `company`, `department`, `team`, `career_level`, `employment_type`, `contract`, `url` - Rohdaten/Features: `created_raw jsonb`, `keywords jsonb` - Zeitverlauf: `first_seen_at`, `last_seen_at` - ETL/DB: `inserted_at`, `updated_at`

Interpretation: - `jobs` enthält die „leichtgewichtige" Job-Sicht (für Listen/Analysen). - `first_seen_at` / `last_seen_at` erlauben später Auswertungen wie „wie lange war ein Job live?", „wann verschwand er?", etc.

`public.job_payloads`

Rolle: Detail-/Raw-Payload pro Job (z. B. komplettes JSON der Stellenanzeige, HTML-Extrakte, zusätzliche Felder).

Primary Key & Relation: - `job_pk uuid` ist **PK** und gleichzeitig **FK** auf `public.jobs(job_pk)`

Das erzwingt faktisch eine **1:1-Beziehung**: - Ein Job kann **maximal einen** Payload-Datensatz haben. - Ein Payload kann **nur** existieren, wenn der Job existiert.

Wichtige Felder: - `payload jsonb` (die eigentlichen Detaildaten) - `ingested_at timestamptz` (Zeitpunkt der Payload-Ingestion, default `now()`) - `payload_hash text` (optional: zur Änderungserkennung/Idempotenz) - zusätzlich: `source`, `job_id` (redundant zur schnellen Diagnose/Traceability)

`public.jobs_without_payload` (View)

Definition: - `jobs j LEFT JOIN job_payloads p ON p.job_pk = j.job_pk` und `p.job_pk IS NULL`

Rolle: - Liefert **alle Jobs, die (noch) keinen Payload haben**.

Typische Use-Cases: - Backfill-/Queue für den Detail-Scraper („scrape detail pages for missing payloads“) - Datenqualitätskontrolle („wie viele Jobs sind nur Metadaten?“)

---

2) Scraping-Entitäten

`public.scrape_companies`

Rolle: Konfiguration/Steuerung, welche Firmen im Scraping aktiv sind.

Primary Key: - `company_key text` (PK)

Wichtige Felder: - `active boolean default true` (Schalter für Monitoring + Einbezug ins Tagesraster) - `company_name text` (optional, lesbarer Name) - `notes text` (optional) - `created_at timestamptz default now()`

Index: - `idx_scrape_companies_active` auf `(active)`

Interpretation: - `company_key` ist das zentrale Join-Merkmal für Monitoring/Reporting. - `active=true` bestimmt, welche Firmen in `daily_summary_v` überhaupt auftauchen.

`public.scrape_runs`

Rolle: Protokolliert jeden Scrape-Durchlauf pro Firma.

Primary Key: - `run_id uuid` (PK)

Wichtige Felder (Auszug): - Bezug: `company_key text` - Zeiten: `started_at`, `finished_at` - Status: `status text` (in den Views werden `success` / `failed` ausgewertet) - Performance: `execution_time_sec`, `cpu_usage_pct` - Job-Metriken: `jobs_fetched`, `jobs_processed`, `new_jobs`, `inactive_jobs`, `skipped_jobs` - Fehlerdiagnose: `error_message`, `stage`, `meta jsonb` - HTTP-Health: `http_requests_total`, `http_requests_failed`, `http_403_count`, `http_429_count`, `http_5xx_count`

Interpretation: - Diese Tabelle ist die Grundlage für alles Monitoring: Tagesstatus, Trends, Block-Erkennung, Performance-Anomalien.

Monitoring-Views im Detail

`public.daily_summary_v` (zentrale Aggregations-View)

Diese View baut ein **tägliches Monitoring-Dataset** für die letzten **30 Tage** pro aktiver Firma.

Schritt 1: Zeit- und Firmenraster („Grid“)

- `today_berlin` wird als Datum in **Europe/Berlin** ermittelt.
- Es wird eine **Serie von 30 Tagen** erzeugt (heute bis heute-29).
- Es werden alle **aktiven Firmen** aus `scrape_companies` genommen.
- Daraus entsteht ein vollständiges Raster: `company_key × run_date`.

Effekt: - Auch wenn es an einem Tag **keinen Run** gab, existiert trotzdem eine Zeile (wichtig für Looker/Charts).

Schritt 2: Runs normalisieren und dem Tag zuordnen

- Quelle: `scrape_runs`
- Es werden nur Runs betrachtet, die:
- `finished_at IS NOT NULL`
- `status IN ('success','failed')`
- Die Tageszuordnung erfolgt über `finished_at AT TIME ZONE 'Europe/Berlin'` ⇒ `run_date_berlin`.

Schritt 3: Aggregation pro Firma/Tag (`agg`)

- `runs_total`, `runs_success`, `runs_failed`
- HTTP-Summen: `http_requests_total_sum`, `http_requests_failed_sum`, `http_403_sum`, `http_429_sum`, `http_5xx_sum`

Schritt 4: “Best Run” und “Last Run”

Best Run (`best_run`): - Nimmt pro Firma/Tag den „besten erfolgreichen Run“ - Sortierlogik: 1) `jobs_processed` DESC (wichtigstes Qualitätskriterium) 2) `jobs_fetched` DESC 3) `finished_at` DESC

Last Run (`last_run`): - Nimmt pro Firma/Tag den zeitlich letzten Run (success oder failed) - Liefert Diagnosefelder: `stage_last`, `error_message_last`, `meta_last`.

Schritt 5: Raten (Health Metrics)

- `http_fail_rate_sum` = `http_requests_failed_sum / http_requests_total_sum` (0 wenn total=0)
- `http_block_rate_sum` = `(http_403_sum + http_429_sum) / http_requests_total_sum` (0 wenn total=0)

Schritt 6: Baselines (rollierend, 14 Tage)

Für jedes `company_key` und `run_date`: - `baseline_jobs_processed_14d` = Median (50%-Perzentil) der `jobs_processed_best` der **vorherigen 14 Tage** (nur >0) - `baseline_exec_time_14d` = Median der `execution_time_sec_best` der **vorherigen 14 Tage**

Wichtig: Es wird **nur rückblickend** gerechnet (bis `run_date - 1`).

Schritt 7: Flags (Anomalie-Signale)

- `flag_zero_jobs`: `COALESCE(jobs_processed_best,0)=0`
- `flag_high_fail_rate`: `http_fail_rate_sum >= 0.30`
- `flag_blocking`: `(http_403_sum + http_429_sum) >= 3`
- `flag_drop_80pct`: `jobs_processed_best < 20% * baseline_jobs_processed_14d` (nur wenn Baseline vorhanden)
- `flag_long_runtime`: `execution_time_sec_best > 2 * baseline_exec_time_14d` (nur wenn Baseline vorhanden)

Schritt 8: `final_status` (Tages-Gesamtbewertung)

Die View klassifiziert jeden Firma/Tag in: - `no_data`: `runs_total = 0` - `success`: mindestens ein success-Run und `jobs_processed_best > 0` **und** keine problematischen Bedingungen - `partial`: success-Run vorhanden, aber z. B.: - hohe Fail-Rate / Blocking / Long Runtime / starker Drop ggü. Baseline - oder success-Run, aber `jobs_processed_best = 0` - `failed`: kein success-Run (bei vorhandenen Runs)

Interpretation: - `success` = „alles gut“ - `partial` = „lief grundsätzlich, aber mit klaren Warnzeichen“ - `failed` = „Runs vorhanden, aber nicht erfolgreich“ - `no_data` = „es gab an dem Tag überhaupt keine verwertbaren Runs“

`public.daily_summary_looker_v`

Diese View ist eine **Looker Studio/BI-kompatible Ableitung** von `daily_summary_v`.

Zusatzspalten: - `is_success` = 1 wenn `final_status='success'` sonst 0 - `is_problem` = 1 wenn `final_status IN ('failed','no_data','partial')` sonst 0

Zweck: - In Looker/BI lassen sich damit sehr einfach Kennzahlen bilden wie: - Erfolgsquote pro Woche/Monat - Problem-Tage zählen - Anteil „partial“ vs. „failed“

`public.alert_candidates_v`

Diese View ist ein **Filter** auf `daily_summary_v` und liefert nur die **auffälligen Fälle**.

Einschlusskriterien: - `final_status IN ('failed','no_data','partial')` **oder** - eines der Flags ist true: - `flag_zero_jobs` - `flag_high_fail_rate` - `flag_blocking` - `flag_drop_80pct` - `flag_long_runtime`

Zweck: - Datenbasis für Alerts/Benachrichtigungen (z. B. E-Mail/Slack) - „Arbeitsliste“ für Debugging: welche Firmen/Days brauchen Aufmerksamkeit?

Wie alles zusammenspielt (Datenfluss)

A) Scraping/Monitoring-Pipeline

1. Du definierst Firmen in `scrape_companies` (`active=true`).
2. Jeder Scrape-Durchlauf schreibt einen Datensatz in `scrape_runs` (Status, Metriken, HTTP-Health, Fehlerdetails).
3. `daily_summary_v` baut daraus pro Firma/Tag (letzte 30 Tage):
4. Run-Zähler
5. Best-/Last-Run
6. HTTP-Raten
7. 14d-Baselines
8. Flags und `final_status`
9. `daily_summary_looker_v` macht das BI-freundlich.
10. `alert_candidates_v` liefert die „Problemfälle“, die man aktiv prüfen/alerten will.

B) Job/Payload-Pipeline

1. Metadaten zu Jobs werden in `jobs` geschrieben (inkl. first/last seen).
2. Detail-Scraper schreibt zusätzlich (oder später) den `payload` nach `job_payloads` (1:1).
3. `jobs_without_payload` zeigt Backfill-Bedarf: Jobs ohne Detailpayload.

Praktische Interpretations-Tipps

- **Wenn** `final_status = no_data`:
- Es gab keinen verwertbaren Run (kein finished_at oder keine success/failed Runs). Ursache oft Scheduling/Timeout/Job nicht gestartet.
- **Wenn** `final_status = failed`:
- Runs existieren, aber kein success. Diagnose typischerweise über `last_run`-Felder:
  - `stage_last`, `error_message_last`, `meta_last`
- **Wenn** `final_status = partial`:
- Es gab Erfolg, aber die Flags zeigen Risiken:
  - hohe Fail-Rate (Netzwerk/Parsing)
  - Blocking (403/429)
  - Drop vs. Baseline (Website-Änderung, Parser kaputt, Captcha)
  - Long runtime (Hänger, Retries, Performanceproblem)

- **HTTP-Block-Detektion:**

- `flag_blocking` reagiert konkret auf 403+429 Summen ≥ 3. Das ist ein robuster Indikator für Rate-Limits/WAF.

- **Payload-Integrität:**

- `jobs_without_payload` ist der wichtigste Frühindikator, ob Detail-Scraping „hinterherhinkt“.

Kurz-Fazit

- `jobs` + `job_payloads` bilden Jobdaten in zwei Ebenen ab: **Metadaten** vs. **Detailpayload** (1:1 über `job_pk`).
- `scrape_companies` + `scrape_runs` bilden die Grundlage für ein **vollständiges Scraping-Monitoring**.
- `daily_summary_v` ist die zentrale Kennzahlen-/Status-View (30 Tage, Berlin-Zeit, Best/Last Run, Baselines, Flags, final_status).
- `daily_summary_looker_v` macht das direkt für Looker Studio nutzbar.
- `alert_candidates_v` konzentriert sich auf die Fälle, die Aufmerksamkeit brauchen.