**W3 Suite – Analisi entità & relazioni**

**Obiettivo:** definire in modo rigoroso le entità core dell’organizzazione e le relazioni **piramidali** tra **Tenant (Organizzazione) ▸ Ragioni Sociali (RS) ▸ Punti Vendita (Store)**, incluse **Brand**, **Canali**, **Driver di business** e **modello di abilitazioni** (RBAC + Scope a cascata). Il tutto pensato per **PostgreSQL + Drizzle** con **RLS**.

**1) Modello organizzativo (piramide)**

Struttura gerarchica e ambiti di visibilità/azione:

Tenant (Organizzazione)

├─ RS (Ragioni Sociali) [1..N]

│ └─ Store (Punti Vendita) [1..N]

│ ├─ Brand [N..N]

│ ├─ Canale (Franchising | Top Dealer | Dealer) [1]

│ └─ Driver Potentials (Fisso | Mobile | Energia | Protezione | Assicurazione) [N]

└─ Users (Risorse) ↔ Roles ↔ Permissions

↳ Assignments a \*\*Scope\*\*: TENANT | RS | STORE (multiselezione)

**Note chiave:**

* Ogni **Store appartiene ad UNA sola RS**.
* Uno Store può essere **associato a più Brand** (es. *WindTre*, *Very Mobile*) ⇒ relazione N–N.
* Ogni Store appartiene a **un solo Canale** (dimensione), con valori: *Franchising*, *Top Dealer*, *Dealer*.
* I **Driver di business** rappresentano gli asset su cui misuriamo potenziale e performance: *Fisso*, *Mobile*, *Energia*, *Protezione*, *Assicurazione*.
* Le **Risorse (utenti)** hanno ruoli e **scope**: possono operare su **tutto il Tenant**, su **una o più RS**, o su **uno o più Store**.

**2) Entità principali & attributi (sintesi)**

**2.1 Tenant**

* id (uuid)
* name, slug
* status, created\_at, archived\_at?

**2.2 Ragione Sociale (RS)**

* id (uuid), tenant\_id (uuid)
* name, vat (P.IVA), billing\_profile\_id (profilo di fatturazione RS)
* status, created\_at, archived\_at?

**2.3 Store (Punto Vendita)**

* id (uuid), tenant\_id, legal\_entity\_id (RS)
* code (univoco per tenant), name
* address, city, province, region, geo?
* channel\_id (Franchising/Top Dealer/Dealer)
* status (active|suspended|closed), opened\_at, closed\_at?
* **Brand association**: via tabella N–N (vedi 2.5)
* **Driver Potentials/KPI**: via tabella N (vedi 2.6)
* billing\_override\_id? (overrides di billing rispetto alla RS)

**2.4 User (Risorsa)**

* id (uuid), tenant\_id
* email, name
* status, mfa\_enabled, last\_login\_at

**2.5 Brand**

* id (uuid), code (es. W3, VERY), name (WindTre, Very Mobile)
* status (active|hidden)

**StoreBrand (N–N)**

* store\_id, brand\_id (unique pair)
* primary? (per eventuale priorità comunicativa)

**2.6 Canale (dimensione)**

* id (uuid), code (FRAN, TOPD, DEAL), name

**2.7 Driver (asset di business)**

* id (uuid), code (FISSO, MOBILE, ENERGIA, PROTEZIONE, ASSICURAZIONE), name
* active (bool)

**StoreDriverPotential**

* store\_id, driver\_id, potential\_score (0..100)
* cluster\_label (es. A/B/C o High/Med/Low)
* kpis (jsonb) (metriche di dettaglio: volumi, conversioni, ARPU, ecc.)
* updated\_at

**2.8 Ruoli & Permessi (RBAC)**

* roles(id, tenant\_id, name)
* role\_perms(role\_id, perm) con perm nello **spazio dei permessi** (es. crm.read, cassa.refund, settings.manage\_users, ...)
* **UserAssignments**: user\_id, role\_id, scope\_type (TENANT|RS|STORE), scope\_id (nullable per TENANT), expires\_at?
* **UserExtraPerms**: user\_id, perm, mode (GRANT|REVOKE) ⇒ overrides puntuali sulle capabilities.

**2.9 Brand Control (Brand Interface)**

* **Artifacts** (creati da *Super Admin brand*):
  + price\_lists, campaigns, templates (WABA/Email/SMS), drivers (nuovi asset di business)
* **Deployments**: pubblicazione verso target:
  + target\_type (ALL|TENANT|RS|STORE) + target\_id? + version, status, audited\_by

**3) Relazioni (cardinalità)**

Tenant 1 ── N RS 1 ── N Store

Store N ── N Brand (via StoreBrand)

Store 1 ── 1 Canale (dimensione)

Store 1 ── N StoreDriverPotential (uno per Driver attivo)

User N ── M Role (via UserAssignments con scope)

Role 1 ── N RolePerms

User 1 ── N UserExtraPerms (grant/revoke puntuali)

BrandArtifacts (price\_lists, campaigns, templates, drivers)

└─ N Deployments ─→ target (ALL|TENANT|RS|STORE)

**4) Modello di abilitazioni (RBAC + Scope a cascata)**

**4.1 Scope effettivo**

* **TENANT**: l’utente vede/agisce su **tutte le RS e Store** del tenant.
* **RS**: l’utente vede/agisce su **tutti gli Store** di quella RS.
* **STORE**: l’utente opera **solo** su quello Store.
* Gli scope assegnati sono **unione** (es. 1 RS + 3 Store di un’altra RS).

**4.2 Calcolo capabilities effettive**

1. Raccogli **permessi dai ruoli** dell’utente (sul relativo scope).
2. Applica **UserExtraPerms** (GRANT/REVOKE) → override puntuale.
3. La richiesta API filtra i dati applicando **RLS** + filtro scope lato applicazione.

**4.3 Esempi**

* *Store Manager* su Store#A: cassa.\*, magazzino.\*, crm.read, **solo** su Store#A.
* *Manager RS* su RS#X: crm.\*, report.\*, settings.manage\_users **limitato** a RS#X.
* *Owner* (scope TENANT): \* su tutto il tenant.

**5) Vincoli & Invarianti (data quality)**

* **Univocità**: stores(code) **unico per tenant**.
* **Integrità**: stores.legal\_entity\_id obbligatorio; store\_brand(store\_id, brand\_id) unico.
* **Driver coverage**: per ciascun store e driver active deve esistere **1 riga** StoreDriverPotential (o policy di default).
* **Soft delete**: archived\_at su RS/Store; impedire cancellazioni fisiche con figli attivi.
* **Audit**: tutte le mutazioni su ruoli, assignments, deployments.

**6) Modellazione PostgreSQL + RLS (estratti)**

**6.1 Tabelle dimensione**

CREATE TABLE brands (

id uuid primary key default gen\_random\_uuid(),

code text unique not null,

name text not null,

status text not null default 'active'

);

CREATE TABLE channels (

id uuid primary key default gen\_random\_uuid(),

code text unique not null,

name text not null

);

CREATE TABLE drivers (

id uuid primary key default gen\_random\_uuid(),

code text unique not null,

name text not null,

active boolean not null default true

);

**6.2 Core multi-tenant**

CREATE TABLE tenants (

id uuid primary key default gen\_random\_uuid(),

name text not null,

slug text unique,

status text not null default 'active',

created\_at timestamptz not null default now()

);

CREATE TABLE legal\_entities (

id uuid primary key default gen\_random\_uuid(),

tenant\_id uuid not null references tenants(id),

name text not null,

vat text,

billing\_profile\_id uuid,

status text not null default 'active',

created\_at timestamptz not null default now(),

archived\_at timestamptz

);

CREATE TABLE stores (

id uuid primary key default gen\_random\_uuid(),

tenant\_id uuid not null references tenants(id),

legal\_entity\_id uuid not null references legal\_entities(id),

code text not null,

name text not null,

channel\_id uuid not null references channels(id),

address text, city text, province text, region text,

status text not null default 'active',

opened\_at date, closed\_at date,

billing\_override\_id uuid,

created\_at timestamptz not null default now(),

archived\_at timestamptz,

unique (tenant\_id, code)

);

CREATE TABLE store\_brands (

store\_id uuid not null references stores(id) on delete cascade,

brand\_id uuid not null references brands(id),

primary key (store\_id, brand\_id)

);

CREATE TABLE store\_driver\_potential (

store\_id uuid not null references stores(id) on delete cascade,

driver\_id uuid not null references drivers(id),

potential\_score int2 not null check (potential\_score between 0 and 100),

cluster\_label text,

kpis jsonb,

updated\_at timestamptz not null default now(),

primary key (store\_id, driver\_id)

);

**6.3 RBAC & Scope**

CREATE TYPE scope\_type AS ENUM ('TENANT','RS','STORE');

CREATE TABLE roles (

id uuid primary key default gen\_random\_uuid(),

tenant\_id uuid not null references tenants(id),

name text not null

);

CREATE TABLE role\_perms (

role\_id uuid not null references roles(id) on delete cascade,

perm text not null,

primary key (role\_id, perm)

);

CREATE TABLE users (

id uuid primary key default gen\_random\_uuid(),

tenant\_id uuid not null references tenants(id),

email text unique not null,

name text,

status text not null default 'active',

mfa\_enabled boolean not null default false,

last\_login\_at timestamptz

);

CREATE TABLE user\_assignments (

user\_id uuid not null references users(id) on delete cascade,

role\_id uuid not null references roles(id) on delete cascade,

scope scope\_type not null,

scope\_id uuid null, -- null solo per TENANT

expires\_at timestamptz,

primary key (user\_id, role\_id, scope, scope\_id)

);

CREATE TABLE user\_extra\_perms (

user\_id uuid not null references users(id) on delete cascade,

perm text not null,

mode text not null check (mode in ('GRANT','REVOKE')),

primary key (user\_id, perm)

);

**6.4 RLS**

Applicare RLS su **tutte** le tabelle con tenant\_id:

ALTER TABLE legal\_entities ENABLE ROW LEVEL SECURITY;

ALTER TABLE stores ENABLE ROW LEVEL SECURITY;

ALTER TABLE users ENABLE ROW LEVEL SECURITY;

ALTER TABLE roles ENABLE ROW LEVEL SECURITY;

-- ...altre

CREATE POLICY rls\_legal\_entities ON legal\_entities

USING (tenant\_id = current\_setting('app.tenant\_id')::uuid);

CREATE POLICY rls\_stores ON stores

USING (tenant\_id = current\_setting('app.tenant\_id')::uuid);

CREATE POLICY rls\_users ON users

USING (tenant\_id = current\_setting('app.tenant\_id')::uuid);

CREATE POLICY rls\_roles ON roles

USING (tenant\_id = current\_setting('app.tenant\_id')::uuid);

Le tabelle **dimensione globali** (brands, channels, drivers) possono stare in uno schema separato senza tenant\_id e senza RLS (read-only), oppure duplicarle per tenant se servono personalizzazioni.

**7) Algoritmo di autorizzazione (richiesta API)**

1. **Resolve tenant** dalla sessione (OIDC) → SET app.tenant\_id = ....
2. **Resolve scope** dell’utente: lista RS e Store consentiti (unione assignments).
3. **Resolve capabilities**: union dei role\_perms + user\_extra\_perms (applicando revoke/grant).
4. **Enforce**:
   * Postgres filtra per RLS (tenant isolation).
   * Service layer aggiunge **WHERE** su RS/Store, quando la tabella non contiene direttamente tenant\_id (es. tabelle join).

**8) Brand Interface – Artefatti & Deploy**

**8.1 Artefatti di Brand**

* **PriceList**: listini e versioni; può essere per Driver/Brand.
* **Campaign**: campagne marketing; target e creatività; template canale (WABA/Email/SMS).
* **Template**: messaggi approvati (WhatsApp), email, SMS.
* **Driver**: possibilità di definire **nuovi asset** (estendibilità).

**8.2 Deployments**

* artifact\_type (PRICE\_LIST|CAMPAIGN|TEMPLATE|DRIVER)
* artifact\_id, version
* target\_type (ALL|TENANT|RS|STORE), target\_id?
* status (pending|in\_progress|completed|failed), logs, created\_by, approved\_by

**Flusso:** il *Super Admin Brand* crea/approva l’artefatto ⇒ crea un Deployment ⇒ il **worker** propaga verso i target (via API interne, con audit e idempotenza).

**9) Query operative (esempi)**

* **Tutti gli Store di un utente** (in base allo scope):
  1. calcola set di RS autorizzate; 2) union con Store assegnati; 3) intersect con stores attivi.
* **Cluster per Driver**: SELECT store\_id, driver\_id, cluster\_label, potential\_score FROM store\_driver\_potential WHERE driver\_id = $drv AND store\_id IN ($stores);
* **Verifica coverage Driver**: left join drivers.active=true con store\_driver\_potential per individuare buchi.

**10) Integrazione moduli applicativi**

* **CRM**: leads, customers hanno riferimenti a store\_id e tenant\_id ⇒ RLS + filtro scope.
* **Cassa**: orders.store\_id, payments.order\_id ⇒ visibilità per store.
* **Magazzino**: stock\_items(store\_id, product\_id) ⇒ RS/Store scope.
* **Report**: aggregazioni per cluster, canale, brand, driver.
* **HR**: sessioni/corsi per store o RS.

**11) Roadmap dati (per questa area)**

1. Migrazioni tabelle: tenants, legal\_entities, stores, brands, channels, drivers, store\_brands, store\_driver\_potential.
2. RBAC base: roles, role\_perms, users, user\_assignments, user\_extra\_perms.
3. RLS e indici (tenant\_id, store\_id, driver\_id, code univoci).
4. Seeder: canali, brand (WindTre, Very Mobile), driver base, cluster iniziali.
5. API CRUD + validazioni; pagina **Settings ▸ Entity Management** per amministrare tutto.

**12) Check-list di conformità**

* RLS attiva su tutte le tabelle multi-tenant.
* Vincoli univocità su stores(code) per tenant.
* Relazioni N–N: store\_brands con PK composta.
* Coverage driver (constraint logico o job giornaliero).
* Audit su ruoli/assignments/deploy.
* Test con due tenant fittizi per validare isolamento & scope.

**13) Estensioni future**

* **Region/Lot**: raggruppamenti geografici per brand analytics e permessi.
* **Store Capability Matrix**: abilità per brand/driver (es. certificazioni, device compatibili).
* **Time-bounded scope**: assegnazioni a scadenza per progetti temporanei.
* **Policy di eredità billing**: RS → Store (override campo per campo con regole).

**Conclusione**

Il modello definito assicura **coerenza gerarchica**, **flessibilità di assegnazione** (scope a cascata), capacità di **segmentazione per Brand/Canale/Driver** e controlli di sicurezza **RLS-first**. È pronto per essere tradotto in **migrazioni Drizzle** e in **API di Settings**.