Database Documentation

Source of truth: `schema.sql`

This document describes the Postgres database schema defined in `schema.sql`, with emphasis on the **application schema (`public`)**. It also includes brief notes on Supabase-managed schemas (`auth`, `storage`, `realtime`, `extensions`) that appear in the dump. For an ER diagram, use Supabase Schema Visualizer (auto layout).

Contents

High-level overview

This database backs a marketplace-style application with:

- **Users** (`public.users`) synchronized to Supabase Auth (`auth.users`).
- **Vendors** and vendor metadata (`public.vendors`) owned by users with the `supplier` role.
- **Promotions** offered by vendors (`public.promos`).
- **Categories**, **regions**, and **affiliations** used for discovery and filtering.
- **Vendor registrations** and **verification documents** supporting onboarding/verification.
- **Inquiries** and **reviews** supporting lead-gen and reputation.
- **Soon-to-wed profiles** and **albums/photos** supporting user-generated content.

The schema uses **Row Level Security (RLS)** across the `public` tables and relies on helper functions:

- `public.is_admin()`
- `public.is_supplier()`
- `public.owns_vendor(vendor_id int)`

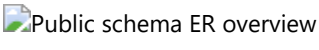
to implement access rules.

At-a-glance: core entities

Area	Primary table(s)	Purpose	Ownership / author	Notes
Auth/profile	`public.users`	App profile + role mapped 1:1 to `auth.users`	User (self) + admin	Synced via `public.handle_new_user()` trigger

Area	Primary table(s)	Purpose	Ownership / author	Notes
Vendor listing	`public.vendors`	Supplier/business listing	Supplier (owner) + admin	Ownership via `vendors.user_id`
Promotions	`public.promos`	Vendor promos/discounts	Vendor owner + admin	`public.active_promos` view filters by validity
Discovery taxonomy	`public.categories`, `public.regions`, `public.affiliations`	Filters/badges	Admin	Publicly viewable
Vendor metadata	`public.vendor_images`, `public.vendor_social_links`	Gallery + social links	Vendor owner + admin	Vendor images are publicly viewable
Many-to-many	`public.vendor_categories`, `public.vendor_affiliations`	Vendor ↔ category/affiliation links	Vendor owner (categories) / admin (affiliations)	See RLS summaries below
Leads	`public.inquiries`	Messages/leads to vendors	Anyone can submit; vendors/admin can view/manage	`user_id` may be null
Reviews	`public.reviews`	Vendor ratings/reviews	Authenticated users	Public can see `published`
Subscription	`public.plans`, `public.subscriptions`	Plans + vendor subscription state	Admin (writes); vendor reads own	Plans public read
Analytics	`public.vendor_analytics`	Daily rollups	System insert; vendor reads own	Insert policy allows any insert (review usage carefully)
Onboarding	`public.vendor_registrations`, `public.verification_documents`	Vendor application + documents	User submits; admin reviews	Docs tied to vendor or registration
Events	`public.bridal_fairs`, `public.fair_registrations`	Fair listing + registrations	Admin manages fairs; anyone can register	Registrations visible to self/admin
Soon-to-wed UGC	`public.soon_to_wed_profiles`, `public.soon_to_wed_albums`, `public.album_photos`	Profiles + albums/photos	User (self) + admin	Visibility-controlled
Bookmarks	`public.saved_vendors`, `public.saved_promos`	User saves	User (self) + admin	Private per-user
Staging/import	`public."staging table"`	Legacy import/staging	None (RLS enabled, no policies)	Deny-by-default under RLS

Diagram: Public schema ER overview


Public schema ER overview

Source file for edits: `docs/diagrams/public-erd.dbml`

Key relationships (FKs)

Child table.column	References	On delete	Meaning
`public.users.id`	`auth.users.id`	CASCADE	App profile follows auth identity
`public.vendors.user_id`	`public.users.id`	(nullable)	Vendor listing owner
`public.promos.vendor_id`	`public.vendors.id`	CASCADE	Promo belongs to vendor
`public.inquiries.vendor_id`	`public.vendors.id`	CASCADE	Inquiry targets vendor
`public.inquiries.user_id`	`public.users.id`	SET NULL	Inquiry may be guest submission
`public.reviews.vendor_id`	`public.vendors.id`	CASCADE	Review targets vendor
`public.reviews.user_id`	`public.users.id`	CASCADE	Review author
`public.vendor_categories.vendor_id`	`public.vendors.id`	CASCADE	Vendor-category link
`public.vendor_categories.category_id`	`public.categories.id`	CASCADE	Vendor-category link
`public.vendor_affiliations.vendor_id`	`public.vendors.id`	CASCADE	Vendor-affiliation link
`public.vendor_affiliations.affiliation_id`	`public.affiliations.id`	CASCADE	Vendor-affiliation link
`public.vendor_images.vendor_id`	`public.vendors.id`	CASCADE	Vendor image belongs to vendor
`public.vendor_social_links.vendor_id`	`public.vendors.id`	CASCADE	Vendor social link belongs to vendor
`public.subscriptions.vendor_id`	`public.vendors.id`	CASCADE	Vendor subscription
`public.subscriptions.plan_id`	`public.plans.id`	RESTRICT	Selected plan
`public.vendor_analytics.vendor_id`	`public.vendors.id`	CASCADE	Analytics rollup belongs to vendor
`public.fair_registrations.fair_id`	`public.bridal_fairs.id`	CASCADE	Registration belongs to fair
`public.fair_registrations.user_id`	`public.users.id`	SET NULL	Registration may be guest
`public.soon_to_wed_profiles.user_id`	`public.users.id`	CASCADE	Profile belongs to user
`public.soon_to_wed_albums.user_id`	`public.users.id`	CASCADE	Album belongs to user

Child table.column	References	On delete	Meaning
public.album_photos.album_id	public.soon_to_wed_albums.id	CASCADE	Photo belongs to album

Simplified access matrix (RLS summary)

This table is a **human summary** of the policies shown later in this document. For authoritative rules, see the policy SQL (from schema.sql).

Table	SELECT	INSERT	UPDATE	DELETE
public.vendors	Public if active; owner/admin can see inactive	Supplier owner	Owner (and admin full access policy)	Owner/admin
public.promos	Public if active; owner/admin otherwise	Owner/admin	Owner/admin	Owner/admin
public.inquiries	Vendor owner OR inquiry author OR admin	Anyone (guest or authed)	Vendor owner	Inquiry author
public.reviews	Public if published; author/admin otherwise	Authenticated user	Author	Author/admin
public.users	Self/admin	Self	Self	(no explicit delete policy documented)
public.vendor_images	Public	Owner/admin	Owner/admin	Owner/admin
public.vendor_social_links	Public	Owner	Owner	Owner (via manage policy)
public.vendor_categories	Public	Owner/admin	Owner/admin	Owner/admin
public.vendor_affiliations	Public	Admin	Admin	Admin
public.plans	Public	Admin	Admin	Admin
public.subscriptions	Vendor owner/admin	Admin	Admin	Admin
public.saved_vendors / public.saved_promos	Self/admin	Self	Self	Self
public.soon_to_wed_profiles	Public if profile is public; self/admin otherwise	(managed via upsert/update)	Self/admin	(no explicit delete policy documented)
public.soon_to_wed_albums / public.album_photos	Public if album/public; self/admin otherwise	Self	Self	Self
public.vendor_registrations	Self/admin	Self	Self while submitted	(no explicit delete policy documented)

Table	SELECT	INSERT	UPDATE	DELETE
public.verification_documents	Owner via vendor or registration; admin	Owner via vendor or registration	Admin	Owner via vendor or registration
public."staging table"	None (RLS enabled; no policies)	None	None	None

Storage buckets/prefix conventions (from schema.sql policies)

Bucket (storage.objects.bucket_id)	Object name prefix convention	Notes
vendor-images	<vendor_id>/...	Public SELECT is allowed for this bucket. INSERT requires public.owns_vendor(storage.foldername(name)[1]::int).
verification-docs	<vendor_id>/...	SELECT is restricted to vendor owner or admin. INSERT requires vendor ownership (same folder rule).

Applying / restoring the schema

This repo includes a pg-dump style snapshot (schema.sql) that contains both:

- The application schema (public.*).
- Supabase-managed schemas (auth, storage, realtime, extensions, etc.).

If you need the exact SQL definitions for any object or policy, refer to schema.sql.

Practical restore notes

schema.sql is a pg-dump style snapshot. When restoring/applying it, prefer restoring it into an **empty database** (or a fresh Supabase project) to avoid conflicts on schemas, extensions, and objects.

If you are applying incremental changes over time, treat schema.sql as the authoritative snapshot and maintain changes via your preferred migration process, then re-generate/update schema.sql from the database.

Application access & required environment variables

The app uses Supabase clients defined in:

- src/lib/supabaseServer.ts (anon client)
- src/lib/supabaseAdmin.ts (service role client)

Required environment variables:

- NEXT_PUBLIC_SUPABASE_URL
- NEXT_PUBLIC_SUPABASE_ANON_KEY
- SUPABASE_SERVICE_ROLE_KEY

Important notes:

- Requests made using the anon key are restricted by Postgres **GRANTs** and **RLS policies**.
- Requests made using the service role key run with elevated privileges and can bypass RLS.

Operational quickstart (database + access)

The details below describe how this schema is *intended* to be used by the app. The authoritative definitions still live in schema.sql.

- **[Restore/apply]** Apply `schema.sql` to a fresh database (or fresh Supabase project) to avoid conflicts with existing schemas/objects.
- **[Anon vs authenticated behavior]** The RLS policies frequently reference `auth.uid()` and `public.user_role()`/`public.is_admin()`/`public.is_supplier()`.
  - `anon` requests will typically have `auth.uid() IS NULL`.
  - `authenticated` requests will have `auth.uid()` set.
- **[Service role]** Any code path that uses the service role key (`SUPABASE_SERVICE_ROLE_KEY`) should be treated as privileged and reviewed carefully because it can bypass RLS.
- **[Storage object paths]** When uploading vendor-owned objects to Storage, ensure the object name begins with the numeric vendor id as the first folder segment (see the `storage.objects` policy notes in this document).

Role model used by RLS helper functions

The helper functions used in policies (`public.is_admin()`, `public.is_supplier()`, `public.user_role()`) rely on `public.users.role`.

Expected roles in `schema.sql`:

- `soon_to_wed`
- `supplier`
- `admin`

Schemas

- **`public`**: application tables, views, functions, triggers, policies.
- **`auth`**: Supabase Auth tables/functions.
- **`storage`**: Supabase Storage tables/policies.
- **`realtime`**: Supabase Realtime replication support.
- **`extensions`**: extensions/event triggers used by Supabase.
- **`graphql`**: `pg_graphql` extension schema.
- **`graphql_public`**: GraphQL helper schema used by Supabase.
- **`pgbouncer`**: pgbouncer helper (`pgbouncer.get_auth`).
- **`vault`**: Supabase Vault schema.

Extensions

Extensions present in `schema.sql`:

- `pg_graphql` (schema: `graphql`)
- `pg_stat_statements` (schema: `extensions`)
- `pgcrypto` (schema: `extensions`)
- `supabase_vault` (schema: `vault`)
- `uuid-ossp` (schema: `extensions`)

Entity relationship overview (public)

Key relationships (FKs) in `public`:

- `public.users.id` -> `auth.users.id` (CASCADE)
- `public.vendors.user_id` -> `public.users.id` (nullable; ownership)
- `public.vendors.region_id` -> `public.regions.id` (nullable)
- `public.vendors.plan_id` -> `public.plans.id` (nullable)
- `public.promos.vendor_id` -> `public.vendors.id` (CASCADE)
- `public.inquiries.vendor_id` -> `public.vendors.id` (CASCADE)

- public.inquiries.user_id -> public.users.id (SET NULL)
- public.reviews.vendor_id -> public.vendors.id (CASCADE)
- public.reviews.user_id -> public.users.id (CASCADE)
- public.subscriptions.vendor_id -> public.vendors.id (CASCADE)
- public.subscriptions.plan_id -> public.plans.id
- public.vendor_categories.vendor_id -> public.vendors.id (CASCADE)
- public.vendor_categories.category_id -> public.categories.id (CASCADE)
- public.vendor_affiliations.vendor_id -> public.vendors.id (CASCADE)
- public.vendor_affiliations.affiliation_id -> public.affiliations.id (CASCADE)
- public.vendor_images.vendor_id -> public.vendors.id (CASCADE)
- public.vendor_social_links.vendor_id -> public.vendors.id (CASCADE)
- public.vendor_analytics.vendor_id -> public.vendors.id (CASCADE)
- public.bridal_fairs is referenced by public.fair_registrations.fair_id (CASCADE)
- public.fair_registrations.user_id -> public.users.id (SET NULL)
- public.soon_to_wed_albums.user_id -> public.users.id (CASCADE)
- public.album_photos.album_id -> public.soon_to_wed_albums.id (CASCADE)
- public.soon_to_wed_profiles.user_id -> public.users.id (CASCADE)
- public.verification_documents.vendor_id -> public.vendors.id (nullable)
- public.verification_documents.registration_id -> public.vendor_registrations.id (nullable)
- public.regions.parent_id -> public.regions.id (self-referential)

public schema reference

Quick index (public)

Tables:

- public.verification_documents
- public.bridal_fairs
- public.fair_registrations
- public.soon_to_wed_profiles
- public.soon_to_wed_albums
- public.album_photos
- public.saved_vendors
- public.saved_promos
- public."staging table"

Views:

- public.active_promos
- public.vendor_details

Functions:

- public.handle_new_user()
- public.is_admin()
- public.is_supplier()
- public.owns_vendor(vendor_id_input int)
- public.update_updated_at_column()
- public.user_role()

Tables

> Types reflect the definitions in schema.sql (including defaults and check constraints).

public.users

Purpose: application user profile and role table. Linked 1:1 with Supabase Auth (auth.users).

Columns

- id uuid (PK, FK -> auth.users.id)
- email varchar(255) (required)
- role varchar(20) (required; check: soon_to_wed, supplier, admin)
- email_verified boolean (default false)
- is_active boolean (default true)
- created_at timestamp (default CURRENT_TIMESTAMP)
- updated_at timestamp (default CURRENT_TIMESTAMP)
- last_login_at timestamp (nullable)

Indexes

- idx_users_email on (email)
- idx_users_role on (role)

Notes

- A trigger (auth.users -> public.users) inserts a row into this table for every new auth user.

public.vendors

Purpose: supplier/business listing.

Columns (selected)

- id int (PK)

- `user_id uuid` (nullable; owner; used by `public.owns_vendor`)
- `business_name varchar(255)` (required)
- `slug varchar(255)` (required)
- `region_id int` (nullable)
- `plan_id int` (nullable)
- `verified_status varchar(20)` (default `unverified`; check: `unverified`, `pending`, `verified`, `rejected`)
- `is_active boolean` (default `true`)
- `is_featured boolean` (default `false`)
- Metrics: `view_count`, `save_count`, `inquiry_count`, `click_count`, `average_rating`, `review_count`
- Timestamps: `created_at`, `updated_at`

Constraints

- PK: `(id)`

Indexes

- `idx_vendors_active` on `(is_active)`
- `idx_vendors_featured` on `(is_featured)`
- `idx_vendors_rating` on `(average_rating)`
- `idx_vendors_region` on `(region_id)`
- `idx_vendors_slug` on `(slug)`
- `idx_vendors_verified` on `(verified_status)`

`public.promos`

Purpose: promotions/discounts offered by vendors.

Columns (selected)

- `id int` (PK)
- `vendor_id int` (FK -> `public.vendors.id`)
- `title varchar(255)` (required)
- `summary text` (nullable)
- `terms text` (nullable)
- `valid_from date` (nullable)
- `valid_to date` (nullable)
- `is_featured boolean` (default `false`)
- `image_url varchar(500)` (nullable)
- `discount_percentage int` (nullable)
- `save_count int` (default `0`)
- `is_active boolean` (default `true`)
- `created_at`, `updated_at`

Indexes

- `idx_promos_active` on `(is_active)`
- `idx_promos_featured` on `(is_featured)`
- `idx_promos_valid` on `(valid_from, valid_to)`
- `idx_promos_vendor` on `(vendor_id)`

`public.categories`

Purpose: category taxonomy.

Columns

- `id int` (PK)

- `name varchar(100)` (required; unique)
- `slug varchar(100)` (required; unique)
- `description text` (nullable)
- `icon varchar(255)` (nullable)
- `display_order int` (default `0`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Constraints

- Unique: `(name)`, `(slug)`

`public.regions`

Purpose: hierarchical geography.

Columns

- `id int` (PK)
- `name varchar(100)` (required; unique)
- `parent_id int` (nullable; FK -> `public.regions.id`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Notes

- The dump includes two unique constraints on name (`regions_name_key` and `regions_name_unique`). They are redundant; keep in mind for future migrations.

`public.affiliations`

Purpose: vendor affiliations/badges.

Columns

- `id int` (PK)
- `name varchar(100)` (unique)
- `slug varchar(100)` (unique)
- `badge_icon varchar(255)`
- `description text`
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

`public.vendor_categories`

Purpose: many-to-many between vendors and categories.

Columns

- `vendor_id int` (PK part, FK -> `public.vendors.id`)
- `category_id int` (PK part, FK -> `public.categories.id`)
- `is_primary boolean` (default `false`)

Indexes

- `idx_vendor_categories_vendor` on `(vendor_id)`
- `idx_vendor_categories_category` on `(category_id)`

`public.vendor_affiliations`

Purpose: many-to-many between vendors and affiliations.

Columns

- `vendor_id int` (PK part, FK -> `public.vendors.id`)
- `affiliation_id int` (PK part, FK -> `public.affiliations.id`)
- `awarded_at timestamp` (default `CURRENT_TIMESTAMP`)

`public.vendor_images`

Purpose: vendor gallery and cover images.

Columns

- `id int` (PK)
- `vendor_id int` (FK -> `public.vendors.id`)
- `image_url varchar(500)`
- `caption text` (nullable)
- `display_order int` (default `0`)
- `is_cover boolean` (default `false`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Indexes

- `idx_vendor_images_vendor` on `(vendor_id)`

`public.vendor_social_links`

Purpose: external links by platform.

Columns

- `id int` (PK)
- `vendor_id int` (FK -> `public.vendors.id`)
- `platform varchar(50)`
- `url varchar(500)`
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Indexes

- `idx_social_links_vendor` on `(vendor_id)`

`public.inquiries`

Purpose: leads/messages sent to vendors.

Columns (selected)

- `id int` (PK)
- `vendor_id int` (FK -> `public.vendors.id`)
- `user_id uuid` (nullable; FK -> `public.users.id`)
- Contact fields: `name`, `email`, `phone`
- `wedding_date date` (nullable)
- `message text` (required)
- `status varchar(20)` (default `new`; check: `new`, `read`, `replied`, `archived`)
- `created_at`, `updated_at`

Indexes

- `idx_inquiries_created` on `(created_at)`
- `idx_inquiries_status` on `(status)`
- `idx_inquiries_user` on `(user_id)`
- `idx_inquiries_vendor` on `(vendor_id)`

public.reviews

Purpose: vendor reviews.

Columns

- id int (PK)
- vendor_id int (FK -> public.vendors.id)
- user_id uuid (FK -> public.users.id)
- rating int (check 1..5)
- review_text text (nullable)
- status varchar(20) (default published; check: published, pending, flagged, removed)
- helpful_count int (default 0)
- created_at, updated_at

Indexes

- idx_reviews_rating on (rating)
- idx_reviews_status on (status)
- idx_reviews_user on (user_id)
- idx_reviews_vendor on (vendor_id)

public.plans

Purpose: subscription plans.

Columns

- id int (PK)
- name varchar(50) (unique)
- price numeric(10,2) (default 0)
- description text (nullable)
- features jsonb (nullable)
- created_at, updated_at

public.subscriptions

Purpose: vendor subscription state.

Columns (selected)

- id int (PK)
- vendor_id int (FK -> public.vendors.id)
- plan_id int (FK -> public.plans.id)
- status varchar(20) (default active; check: trial, active, past_due, canceled, expired)
- start_date date (required)
- end_date date (nullable)
- renewal_date date (nullable)
- Provider fields: provider, provider_customer_id, provider_subscription_id
- created_at, updated_at

Indexes

- idx_subscriptions_status on (status)
- idx_subscriptions_vendor on (vendor_id)

public.vendor_analytics

Purpose: daily rollups for vendor performance.

Columns

- `id int` (PK)
- `vendor_id int` (FK -> `public.vendors.id`)
- `date date` (required)
- Metrics: `views`, `saves`, `inquiries`, `website_clicks`, `phone_clicks`
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Indexes

- `idx_analytics_vendor_date` on `(vendor_id, date)`

`public.vendor_registrations`

Purpose: onboarding flow for vendor signups.

Columns (selected)

- `id int` (PK)
- `submitted_by_user_id uuid` (nullable)
- `business_name varchar(255)`
- `contact_email varchar(255)`
- `contact_phone varchar(50)` (nullable)
- `category_id int` (nullable)
- `location varchar(255)` (nullable)
- `description text` (nullable)
- `website_url varchar(500)` (nullable)
- `plan_id int` (nullable)
- `status varchar(20)` (default `submitted`; check: `submitted`, `in_review`, `approved`, `rejected`)
- `admin_notes text` (nullable)
- `created_at`, `updated_at`
- `reviewed_at timestamp` (nullable)
- `reviewed_by uuid` (nullable)

Indexes

- `idx_registrations_status` on `(status)`

`public.verification_documents`

Purpose: documents for vendor verification.

Columns

- `id int` (PK)
- `vendor_id int` (nullable; FK -> `public.vendors.id`)
- `registration_id int` (nullable; FK -> `public.vendor_registrations.id`)
- `doc_type varchar(50)` (check: `business_permit`, `dti`, `sec`, `bir`, `other`)
- `file_url varchar(500)`
- `file_name varchar(255)` (nullable)
- `status varchar(20)` (default `pending`; check: `pending`, `approved`, `rejected`)
- `uploaded_at timestamp` (default `CURRENT_TIMESTAMP`)
- `reviewed_at timestamp` (nullable)
- `notes text` (nullable)

Indexes

- idx_verification_docs_vendor on (vendor_id)
- idx_verification_docs_registration on (registration_id)

public.bridal_fairs

Purpose: fair/event listing.

Columns (selected)

- id int (PK)
- title varchar(255)
- slug varchar(255) (unique)
- description text (nullable)
- start_date date (required)
- end_date date (nullable)
- venue varchar(255)
- venue_address text (nullable)
- venue_map_url varchar(500) (nullable)
- image_url varchar(500) (nullable)
- registration_url varchar(500) (nullable)
- is_featured boolean (default false)
- is_active boolean (default true)
- created_at, updated_at

Indexes

- idx_fairs_active on (is_active)
- idx_fairs_dates on (start_date, end_date)

public.fair_registrations

Purpose: registrations for bridal fairs.

Columns

- id int (PK)
- fair_id int (FK -> public.bridal_fairs.id)
- user_id uuid (nullable; FK -> public.users.id)
- name varchar(255)
- email varchar(255)
- phone varchar(50) (nullable)
- wedding_date date (nullable)
- notes text (nullable)
- created_at timestamp (default CURRENT_TIMESTAMP)

Indexes

- idx_fair_registrations_fair on (fair_id)
- idx_fair_registrations_user on (user_id)

public.soon_to_wed_profiles

Purpose: soon-to-wed profile fields.

Columns (selected)

- user_id uuid (PK, FK -> public.users.id)
- bride_nickname varchar(100) (nullable)

- `groom_nickname varchar(100)` (nullable)
- `wedding_date date` (nullable)
- `wedding_date_public boolean` (default `false`)
- `wedding_venue_area varchar(255)` (nullable)
- `wedding_venue_public boolean` (default `false`)
- `location varchar(255)` (nullable)
- `profile_visibility varchar(20)` (default `private`; check: `public`, `private`)
- `budget_range varchar(50)` (nullable)
- `wedding_style varchar(100)` (nullable)
- `notes text` (nullable)
- `profile_photo_url varchar(500)` (nullable)
- `created_at`, `updated_at`

`public.soon_to_wed_albums`

Purpose: albums owned by users.

Columns

- `id int` (PK)
- `user_id uuid` (FK -> `public.users.id`)
- `title varchar(255)` (default `My Wedding Album`)
- `visibility varchar(20)` (default `private`; check `public`, `private`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Indexes

- `idx_albums_user` on `(user_id)`

`public.album_photos`

Purpose: photos within albums.

Columns

- `id int` (PK)
- `album_id int` (FK -> `public.soon_to_wed_albums.id`)
- `photo_url varchar(500)`
- `caption text` (nullable)
- `display_order int` (default `0`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

Indexes

- `idx_album_photos_album` on `(album_id)`

`public.saved_vendors`

Purpose: user saves/bookmarks for vendors.

Columns

- `user_id uuid` (PK part; FK -> `public.users.id`)
- `vendor_id int` (PK part; FK -> `public.vendors.id`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

`public.saved_promos`

Purpose: user saves/bookmarks for promos.

Columns

- `user_id uuid` (PK part; FK -> `public.users.id`)
- `promo_id int` (PK part; FK -> `public.promos.id`)
- `created_at timestamp` (default `CURRENT_TIMESTAMP`)

`public."staging table"`

Purpose: raw imported staging data.

Notes

- Column names include spaces/symbols and should be treated as a temporary/staging structure.
- This table is not expected to be used by application runtime code; treat it as import-only/legacy unless intentionally referenced.

Views

`public.active_promos`

Purpose: convenience view returning promos joined to vendors, filtered to currently-valid and active promos.

- Joins `public.promos` to `public.vendors`.
- Filters:
  - `p.is_active = true`
  - `valid_to is null OR valid_to >= current_date`
  - `valid_from is null OR valid_from <= current_date`

`public.vendor_details`

Purpose: convenience view returning active vendors with plan name and owner email.

- Left joins `public.vendors` to `public.plans` and `public.users`.
- Filters: `v.is_active = true`

Functions

`public.handle_new_user()`

Type: `trigger` function (SECURITY DEFINER)

Purpose: keeps `public.users` in sync with `auth.users`.

Behavior

- On new row in `auth.users`, inserts into `public.users`:
  - `id = NEW.id`
  - `email = NEW.email`
  - `role = 'soon_to_wed'` (default)
  - `email_verified = (NEW.email_confirmed_at IS NOT NULL)`

`public.is_admin()`

Type: sql function (SECURITY DEFINER)

Returns: boolean

Logic: true if the current authenticated user (`auth.uid()`) has a `public.users` row with `role = 'admin'`.

`public.is_supplier()`

Type: sql function (SECURITY DEFINER)

Returns: boolean

Logic: true if the current authenticated user has `role = 'supplier'`.

`public.owns_vendor(vendor_id_input int)`

Type: sql function (SECURITY DEFINER)

Returns: boolean

Logic: true if `public.vendors.id = vendor_id_input` and `public.vendors.user_id = auth.uid()`.

`public.update_updated_at_column()`

Type: trigger function

Purpose: sets `NEW.updated_at = CURRENT_TIMESTAMP` before update.

`public.user_role()`

Type: sql function (SECURITY DEFINER)

Returns: text

Logic: returns the `role` for the current user.

Triggers

- `auth.users`: `on_auth_user_created` (AFTER INSERT) -> `public.handle_new_user()`
- `public.bridal_fairs`: `update_fairs_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.inquiries`: `update_inquiries_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.plans`: `update_plans_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.promos`: `update_promos_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.vendor_registrations`: `update_registrations_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.reviews`: `update_reviews_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.soon_to_wed_profiles`: `update_stw_profiles_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.subscriptions`: `update_subscriptions_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.users`: `update_users_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`
- `public.vendors`: `update_vendors_updated_at` (BEFORE UPDATE) -> `public.update_updated_at_column()`

Row Level Security (RLS) & policies

RLS is enabled for all application tables in `public`.

Policies are grouped below by table. Policy statements are taken directly from `schema.sql`.

`public.affiliations`

- "Affiliations are viewable by everyone":
  - `CREATE POLICY "Affiliations are viewable by everyone" ON public.affiliations FOR SELECT USING (true);`
- "Affiliations are editable by admins only":
  - `CREATE POLICY "Affiliations are editable by admins only" ON public.affiliations USING (public.is_admin()) WITH CHECK (public.is_admin());`

public.categories

- "Categories are viewable by everyone":
  - `CREATE POLICY "Categories are viewable by everyone" ON public.categories FOR SELECT USING (true);`
- "Categories are editable by admins only":
  - `CREATE POLICY "Categories are editable by admins only" ON public.categories USING (public.is_admin()) WITH CHECK (public.is_admin());`

public.regions

- "Regions are viewable by everyone":
  - `CREATE POLICY "Regions are viewable by everyone" ON public.regions FOR SELECT USING (true);`
- "Regions are editable by admins only":
  - `CREATE POLICY "Regions are editable by admins only" ON public.regions USING (public.is_admin()) WITH CHECK (public.is_admin());`

public.plans

- "Plans are viewable by everyone":
  - `CREATE POLICY "Plans are viewable by everyone" ON public.plans FOR SELECT USING (true);`
- "Plans are editable by admins only":
  - `CREATE POLICY "Plans are editable by admins only" ON public.plans USING (public.is_admin()) WITH CHECK (public.is_admin());`

public.bridal_fairs

- "Active fairs are viewable by everyone":
  - `CREATE POLICY "Active fairs are viewable by everyone" ON public.bridal_fairs FOR SELECT USING (((is_active = true) OR public.is_admin()));`
- "Only admins can manage fairs":
  - `CREATE POLICY "Only admins can manage fairs" ON public.bridal_fairs USING (public.is_admin()) WITH CHECK (public.is_admin());`

public.fair_registrations

- "Anyone can create fair registrations":
  - `CREATE POLICY "Anyone can create fair registrations" ON public.fair_registrations FOR INSERT WITH CHECK (true);`
- "Admins can view all fair registrations":
  - `CREATE POLICY "Admins can view all fair registrations" ON public.fair_registrations FOR SELECT USING (public.is_admin());`
- "Users can view their own fair registrations":
  - `CREATE POLICY "Users can view their own fair registrations" ON public.fair_registrations FOR SELECT USING (((user_id = auth.uid()) OR public.is_admin()));`

public.vendors

- "Active vendors are viewable by everyone":

    - CREATE POLICY "Active vendors are viewable by everyone" ON public.vendors FOR SELECT USING (((is_active = true) OR (user_id = auth.uid()) OR public.is_admin()));
- "Suppliers can create their own vendor listing":
    - CREATE POLICY "Suppliers can create their own vendor listing" ON public.vendors FOR INSERT WITH CHECK (((user_id = auth.uid()) AND public.is_supplier()));
- "Suppliers can update their own vendor":
    - CREATE POLICY "Suppliers can update their own vendor" ON public.vendors FOR UPDATE USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));
- "Suppliers can delete their own vendor":
    - CREATE POLICY "Suppliers can delete their own vendor" ON public.vendors FOR DELETE USING (((user_id = auth.uid()) OR public.is_admin()));
- "Admins have full access to vendors":
    - CREATE POLICY "Admins have full access to vendors" ON public.vendors USING (public.is_admin()) WITH CHECK (public.is_admin());

public.promos

- "Active promos are viewable by everyone":
    - CREATE POLICY "Active promos are viewable by everyone" ON public.promos FOR SELECT USING (((is_active = true) OR public.owns_vendor(vendor_id) OR public.is_admin()));
- "Vendor owners can manage their promos":
    - CREATE POLICY "Vendor owners can manage their promos" ON public.promos USING (public.owns_vendor(vendor_id)) WITH CHECK (public.owns_vendor(vendor_id));
- "Admins can manage all promos":
    - CREATE POLICY "Admins can manage all promos" ON public.promos USING (public.is_admin()) WITH CHECK (public.is_admin());

public.vendor_categories

- "Vendor categories are viewable by everyone":
    - CREATE POLICY "Vendor categories are viewable by everyone" ON public.vendor_categories FOR SELECT USING (true);
- "Vendor owners can manage their categories":
    - CREATE POLICY "Vendor owners can manage their categories" ON public.vendor_categories USING (public.owns_vendor(vendor_id)) WITH CHECK (public.owns_vendor(vendor_id));
- "Admins can manage all vendor categories":
    - CREATE POLICY "Admins can manage all vendor categories" ON public.vendor_categories USING (public.is_admin()) WITH CHECK (public.is_admin());

public.vendor_images

- "Vendor images are viewable by everyone":
    - CREATE POLICY "Vendor images are viewable by everyone" ON public.vendor_images FOR SELECT USING (true);
- "Vendor owners can manage their images":
    - CREATE POLICY "Vendor owners can manage their images" ON public.vendor_images USING (public.owns_vendor(vendor_id)) WITH CHECK (public.owns_vendor(vendor_id));
- "Admins can manage all vendor images":
    - CREATE POLICY "Admins can manage all vendor images" ON public.vendor_images USING (public.is_admin()) WITH CHECK (public.is_admin());

public.vendor_social_links

- "Social links are viewable by everyone":

    - CREATE POLICY "Social links are viewable by everyone" ON public.vendor_social_links FOR SELECT USING (true);
- "Vendor owners can manage their social links":
    - CREATE POLICY "Vendor owners can manage their social links" ON public.vendor_social_links USING (public.owns_vendor(vendor_id)) WITH CHECK (public.owns_vendor(vendor_id));

public.vendor_analytics

- "System can insert analytics":
    - CREATE POLICY "System can insert analytics" ON public.vendor_analytics FOR INSERT WITH CHECK (true);
- "Vendors can view their own analytics":
    - CREATE POLICY "Vendors can view their own analytics" ON public.vendor_analytics FOR SELECT USING ((public.owns_vendor(vendor_id) OR public.is_admin()));
- "Admins have full access to analytics":
    - CREATE POLICY "Admins have full access to analytics" ON public.vendor_analytics USING (public.is_admin()) WITH CHECK (public.is_admin());

public.inquiries

- "Authenticated users can create inquiries":
    - CREATE POLICY "Authenticated users can create inquiries" ON public.inquiries FOR INSERT WITH CHECK (((user_id = auth.uid()) OR (auth.uid() IS NULL)));
- "Vendors can view inquiries sent to them":
    - CREATE POLICY "Vendors can view inquiries sent to them" ON public.inquiries FOR SELECT USING ((public.owns_vendor(vendor_id) OR (user_id = auth.uid()) OR public.is_admin()));
- "Vendors can update inquiry status":
    - CREATE POLICY "Vendors can update inquiry status" ON public.inquiries FOR UPDATE USING (public.owns_vendor(vendor_id)) WITH CHECK (public.owns_vendor(vendor_id));
- "Users can delete their own inquiries":
    - CREATE POLICY "Users can delete their own inquiries" ON public.inquiries FOR DELETE USING ((user_id = auth.uid()));
- "Admins have full access to inquiries":
    - CREATE POLICY "Admins have full access to inquiries" ON public.inquiries USING (public.is_admin()) WITH CHECK (public.is_admin());

public.reviews

- "Authenticated users can create reviews":
    - CREATE POLICY "Authenticated users can create reviews" ON public.reviews FOR INSERT WITH CHECK (((user_id = auth.uid()) AND (auth.uid() IS NOT NULL)));
- "Published reviews are viewable by everyone":
    - CREATE POLICY "Published reviews are viewable by everyone" ON public.reviews FOR SELECT USING ((((status)::text = 'published'::text) OR (user_id = auth.uid()) OR public.is_admin()));
- "Users can update their own reviews":
    - CREATE POLICY "Users can update their own reviews" ON public.reviews FOR UPDATE USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));
- "Users can delete their own reviews":
    - CREATE POLICY "Users can delete their own reviews" ON public.reviews FOR DELETE USING (((user_id = auth.uid()) OR public.is_admin()));
- "Admins can manage all reviews":
    - CREATE POLICY "Admins can manage all reviews" ON public.reviews USING (public.is_admin()) WITH CHECK (public.is_admin());

public.users

- "New users can insert their own record":
  - CREATE POLICY "New users can insert their own record" ON public.users FOR INSERT WITH CHECK ((id = auth.uid()));
- "Users can view their own record":
  - CREATE POLICY "Users can view their own record" ON public.users FOR SELECT USING (((id = auth.uid()) OR public.is_admin()));
- "Users can update their own record":
  - CREATE POLICY "Users can update their own record" ON public.users FOR UPDATE USING ((id = auth.uid())) WITH CHECK ((id = auth.uid()));
- "Admins have full access to users":
  - CREATE POLICY "Admins have full access to users" ON public.users USING (public.is_admin()) WITH CHECK (public.is_admin());

public.soon_to_wed_albums

- "Users can view public albums or their own":
  - CREATE POLICY "Users can view public albums or their own" ON public.soon_to_wed_albums FOR SELECT USING ((((visibility)::text = 'public'::text) OR (user_id = auth.uid()) OR public.is_admin()));
- "Users can manage their own albums":
  - CREATE POLICY "Users can manage their own albums" ON public.soon_to_wed_albums USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));

public.album_photos

- "Photos visible based on album visibility":
  - CREATE POLICY "Photos visible based on album visibility" ON public.album_photos FOR SELECT USING ((EXISTS (SELECT 1 FROM public.soon_to_wed_albums a WHERE ((a.id = album_photos.album_id) AND (((a.visibility)::text = 'public'::text) OR (a.user_id = auth.uid()) OR public.is_admin())))));
- "Users can manage photos in their own albums":
  - CREATE POLICY "Users can manage photos in their own albums" ON public.album_photos USING ((EXISTS (SELECT 1 FROM public.soon_to_wed_albums a WHERE ((a.id = album_photos.album_id) AND (a.user_id = auth.uid())))) WITH CHECK ((EXISTS (SELECT 1 FROM public.soon_to_wed_albums a WHERE ((a.id = album_photos.album_id) AND (a.user_id = auth.uid()))))));

public.soon_to_wed_profiles

- "Users can view public profiles":
  - CREATE POLICY "Users can view public profiles" ON public.soon_to_wed_profiles FOR SELECT USING ((((profile_visibility)::text = 'public'::text) OR (user_id = auth.uid()) OR public.is_admin()));
- "Users can manage their own profile":
  - CREATE POLICY "Users can manage their own profile" ON public.soon_to_wed_profiles USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));
- "Admins have full access to profiles":
  - CREATE POLICY "Admins have full access to profiles" ON public.soon_to_wed_profiles USING (public.is_admin()) WITH CHECK (public.is_admin());

public.saved_vendors / public.saved_promos

- "Users can view their own saved vendors":

    - CREATE POLICY "Users can view their own saved vendors" ON public.saved_vendors FOR SELECT USING (((user_id = auth.uid()) OR public.is_admin()));
- "Users can manage their own saved vendors":
    - CREATE POLICY "Users can manage their own saved vendors" ON public.saved_vendors USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));
- "Users can view their own saved promos":
    - CREATE POLICY "Users can view their own saved promos" ON public.saved_promos FOR SELECT USING (((user_id = auth.uid()) OR public.is_admin()));
- "Users can manage their own saved promos":
    - CREATE POLICY "Users can manage their own saved promos" ON public.saved_promos USING ((user_id = auth.uid())) WITH CHECK ((user_id = auth.uid()));

public.vendor_affiliations

- "Vendor affiliations are viewable by everyone":
    - CREATE POLICY "Vendor affiliations are viewable by everyone" ON public.vendor_affiliations FOR SELECT USING (true);
- "Only admins can manage affiliations":
    - CREATE POLICY "Only admins can manage affiliations" ON public.vendor_affiliations USING (public.is_admin()) WITH CHECK (public.is_admin());

public.vendor_registrations

- "Authenticated users can create registrations":
    - CREATE POLICY "Authenticated users can create registrations" ON public.vendor_registrations FOR INSERT WITH CHECK ((submitted_by_user_id = auth.uid()));
- "Users can view their own registrations":
    - CREATE POLICY "Users can view their own registrations" ON public.vendor_registrations FOR SELECT USING (((submitted_by_user_id = auth.uid()) OR public.is_admin()));
- "Users can update their pending registrations":
    - CREATE POLICY "Users can update their pending registrations" ON public.vendor_registrations FOR UPDATE USING (((submitted_by_user_id = auth.uid()) AND ((status)::text = 'submitted'::text))) WITH CHECK (((submitted_by_user_id = auth.uid()) AND ((status)::text = 'submitted'::text)));
- "Admins can manage all registrations":
    - CREATE POLICY "Admins can manage all registrations" ON public.vendor_registrations USING (public.is_admin()) WITH CHECK (public.is_admin());

public.verification_documents

- "Vendors can view their own documents":
    - CREATE POLICY "Vendors can view their own documents" ON public.verification_documents FOR SELECT USING ((public.owns_vendor(vendor_id) OR (EXISTS (SELECT 1 FROM public.vendor_registrations vr WHERE ((vr.id = verification_documents.registration_id) AND (vr.submitted_by_user_id = auth.uid()))))) OR public.is_admin()));
- "Vendors can upload their own documents":
    - CREATE POLICY "Vendors can upload their own documents" ON public.verification_documents FOR INSERT WITH CHECK ((public.owns_vendor(vendor_id) OR (EXISTS (SELECT 1 FROM public.vendor_registrations vr WHERE ((vr.id = verification_documents.registration_id) AND (vr.submitted_by_user_id = auth.uid()))))));
- "Vendors can delete their own documents":
    - CREATE POLICY "Vendors can delete their own documents" ON public.verification_documents FOR DELETE USING ((public.owns_vendor(vendor_id) OR (EXISTS (SELECT 1 FROM public.vendor_registrations vr WHERE ((vr.id = verification_documents.registration_id) AND (vr.submitted_by_user_id = auth.uid()))))));

- `"Admins can manage all verification documents"`:
  - `CREATE POLICY "Admins can manage all verification documents" ON public.verification_documents USING (public.is_admin()) WITH CHECK (public.is_admin());`

`public.subscriptions`

- `"Vendors can view their own subscriptions"`:
  - `CREATE POLICY "Vendors can view their own subscriptions" ON public.subscriptions FOR SELECT USING ((public.owns_vendor(vendor_id) OR public.is_admin()));`
- `"Only admins can manage subscriptions"`:
  - `CREATE POLICY "Only admins can manage subscriptions" ON public.subscriptions USING (public.is_admin()) WITH CHECK (public.is_admin());`

`public."staging table"`

`schema.sql` enables RLS on `public."staging table"`, but does not define any `CREATE POLICY` rules for it.

As a result, access is **denied by default under RLS** for `anon`/`authenticated` (even if table privileges are granted). Treat this table as import-only/legacy unless you intentionally add policies and application usage.

Grants / privileges

The dump grants `ALL` privileges on most `public` relations to roles:

- `anon`
- `authenticated`
- `service_role`
- `prisma`

Important: With Supabase, `GRANT` controls which operations are even possible, but **RLS policies ultimately restrict which rows** are accessible for `anon`/`authenticated`. Keep both aligned.

Supabase-managed schemas (brief)

`auth`

Supabase Auth tables and helper SQL functions exist in `auth.*`. The application relies on:

- `auth.uid()` to identify the current user.
- `auth.users` as the canonical identity store.

`storage`

Supabase Storage tables are present (`storage.buckets`, `storage.objects`, etc.). This dump includes policies that are directly relevant to the app:

Note: this section is a high-level summary. For the authoritative policy definitions, refer to `schema.sql`.

- `storage.objects` SELECT for bucket `vendor-images` is public.
- `storage.objects` INSERT for bucket `vendor-images` and `verification-docs` is restricted to vendor owners (folder name contains vendor id).
- `storage.objects` SELECT for bucket `verification-docs` is restricted to vendor owners or admin.

Bucket/path convention implied by policies (important)

The storage policies in `schema.sql` use:

- `storage.foldername(name)[1]` cast to integer, and then
- `public.owns_vendor(vendor_id)`

This means object names are expected to be prefixed like:

- `vendor-images/<vendor_id>/...`
- `verification-docs/<vendor_id>/...`

where `<vendor_id>` is the **first path segment** of the object name and must parse as an integer.

`realtime`

`realtime.*` tables/functions support Supabase Realtime. No app code should generally depend on these directly.

`extensions`

Contains event triggers/functions used by Supabase for schema change tracking and GraphQL placeholders.

Appendix: seed / dump data

`schema.sql` includes `COPY ... FROM stdin;` sections containing initial data. In the excerpt observed:

- `public.affiliations` includes a row: `HRAP` (`slug = hrap`).
- `public.categories` includes multiple seeded categories (e.g. `Venue`, `Photo & Video`, ...).

If you plan to use `schema.sql` as a repeatable seed, keep in mind it also includes data for Supabase-managed schemas (`auth.*`, etc.).