<https://gemini.google.com/u/1/app/28b5eb1f3acf0522?is_sa=1&android-min-version=301356232&ios-min-version=322.0&campaign_id=bkws&utm_source=google&utm_medium=cpc&utm_campaign=2024enUS_gemfeb&pt=9008&mt=8&ct=p-growth-sem-bkws>

<https://gemini.google.com/u/1/app/39ab8052602be0ce>

<https://gemini.google.com/u/1/app/8f9ebd64aa0fd57f>

### 3\. Updated Production-Ready Specification: `public.user\_waypoint\_votes`

Version: 2.1.1 (Internal revision post-checklist review)

Date: May 18, 2025

1\. Purpose & Primary Use-Cases

- Mission Statement: To enable authenticated users (pilgrims) to cast a simple "thumbs up" or "thumbs down" vote on specific waypoints, providing a quick community sentiment indicator and supporting features like sorting by popularity.

- Key User-Story Touchpoints:

- Pilgrim (Anna) - Story A8: Directly allows Anna to see "thumbs up/down" counts for waypoints and contribute her own vote, helping her gauge community sentiment.

- Host (Marco): Indirectly allows Marco to gauge pilgrim sentiment towards his accommodation via aggregated vote counts.

- Regional Manager (Sofia): Indirectly allows Sofia to identify well-regarded or potentially problematic waypoints based on vote patterns.

- Admin Team - Story D1 & D2: Allows admins to oversee voting activity and potentially use aggregated vote data to inform content curation.

2\. Schema

| column | data\_type | constraints | description |

| `profile\_id` | `uuid` | `PRIMARY KEY (Part 1/2)`, `NOT NULL`, `FOREIGN KEY to public.profiles(id) ON DELETE CASCADE` | The ID from the `profiles` table of the user casting the vote. If the user's profile is deleted, their vote is also removed. |

| `waypoint\_id` | `integer` | `PRIMARY KEY (Part 2/2)`, `NOT NULL`, `FOREIGN KEY to public.waypoints(id) ON DELETE CASCADE` | The ID from the `waypoints` table that is being voted on. If the waypoint is deleted, corresponding votes are also removed. Crucial: Confirm type matches `waypoints.id`. |

| `vote\_type` | `public.vote\_type\_enum` | `NOT NULL` | The type of vote cast by the user ('up' or 'down'). |

| `vote\_source\_platform` | `text` | `NULLABLE` | Optional: Indicates the platform from which the vote was cast (e.g., 'web\_app\_v1', 'mobile\_ios\_v1'). Useful for analytics. |

| `created\_at` | `timestamp with time zone` | `NOT NULL`, `DEFAULT now()` | Timestamp of when the vote was initially cast. |

| `updated\_at` | `timestamp with time zone` | `NOT NULL`, `DEFAULT now()` | Timestamp of when the vote was last modified (e.g., `vote\_type` changed, or `deleted\_at` status changed). Auto-updated by a database trigger. |

| `deleted\_at` | `timestamp with time zone` | `NULLABLE` | Timestamp for soft deletion. If set, the vote is considered retracted or inactive but preserved for history. Active votes have `NULL`. |

3\. PostgreSQL DDL

SQL

```

-- ENUM Type Definition (if not already created globally)

DO $$ BEGIN

IF NOT EXISTS (SELECT 1 FROM pg\_type WHERE typname = 'vote\_type\_enum') THEN

CREATE TYPE public.vote\_type\_enum AS ENUM ('up', 'down');

END IF;

END $$;

-- Table Definition for user\_waypoint\_votes

CREATE TABLE public.user\_waypoint\_votes (

profile\_id uuid NOT NULL,

waypoint\_id integer NOT NULL, -- CRITICAL: Confirm this matches the data type of public.waypoints.id (e.g., integer, bigint, uuid)

vote\_type public.vote\_type\_enum NOT NULL,

vote\_source\_platform text NULL,

created\_at timestamptz NOT NULL DEFAULT now(),

updated\_at timestamptz NOT NULL DEFAULT now(),

deleted\_at timestamptz NULL,

CONSTRAINT user\_waypoint\_votes\_pkey PRIMARY KEY (profile\_id, waypoint\_id),

CONSTRAINT user\_waypoint\_votes\_profile\_id\_fkey FOREIGN KEY (profile\_id) REFERENCES public.profiles(id) ON DELETE CASCADE,

CONSTRAINT user\_waypoint\_votes\_waypoint\_id\_fkey FOREIGN KEY (waypoint\_id) REFERENCES public.waypoints(id) ON DELETE CASCADE

);

-- Comments on Table and Columns

COMMENT ON TABLE public.user\_waypoint\_votes IS 'Stores user votes (up/down) for specific waypoints. Supports soft deletion for vote retractions. Version 2.1.1.';

COMMENT ON COLUMN public.user\_waypoint\_votes.profile\_id IS 'FK to profiles.id of the user who cast the vote. Part of PK.';

COMMENT ON COLUMN public.user\_waypoint\_votes.waypoint\_id IS 'FK to waypoints.id of the waypoint being voted on. Part of PK. CRITICAL: Confirm data type matches waypoints.id.';

COMMENT ON COLUMN public.user\_waypoint\_votes.vote\_type IS 'The type of vote: ''up'' or ''down''. Uses public.vote\_type\_enum.';

COMMENT ON COLUMN public.user\_waypoint\_votes.vote\_source\_platform IS 'Optional: Platform from which the vote was cast (e.g., ''web\_app\_v1'', ''mobile\_ios\_v1''). For analytics.';

COMMENT ON COLUMN public.user\_waypoint\_votes.created\_at IS 'Timestamp of when the vote was initially cast.';

COMMENT ON COLUMN public.user\_waypoint\_votes.updated\_at IS 'Timestamp of when the vote record was last updated (vote type changed or soft deleted/restored). Auto-updated by trigger.';

COMMENT ON COLUMN public.user\_waypoint\_votes.deleted\_at IS 'Timestamp for soft deletion. If set, the vote is considered retracted/inactive.';

-- Index for efficient aggregation of active votes (primarily if not solely relying on triggers to waypoints table)

CREATE INDEX IF NOT EXISTS idx\_user\_waypoint\_votes\_active

ON public.user\_waypoint\_votes (waypoint\_id, vote\_type)

WHERE (deleted\_at IS NULL);

-- Generic Trigger Function to update 'updated\_at' timestamp (if not already created globally)

CREATE OR REPLACE FUNCTION public.handle\_updated\_at()

RETURNS TRIGGER AS $$

BEGIN

NEW.updated\_at = NOW();

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Trigger for user\_waypoint\_votes table 'updated\_at'

CREATE TRIGGER on\_user\_waypoint\_votes\_updated\_at

BEFORE UPDATE ON public.user\_waypoint\_votes

FOR EACH ROW

EXECUTE FUNCTION public.handle\_updated\_at();

-- Trigger Function to update aggregated counts on the 'waypoints' table (CRITICAL)

-- (Assumes public.waypoints table has up\_vote\_count INTEGER and down\_vote\_count INTEGER columns)

CREATE OR REPLACE FUNCTION public.update\_waypoint\_vote\_counts()

RETURNS TRIGGER AS $$

BEGIN

-- Handle INSERT operations

IF (TG\_OP = 'INSERT') THEN

IF NEW.deleted\_at IS NULL THEN -- Only count active votes

IF NEW.vote\_type = 'up' THEN

UPDATE public.waypoints SET up\_vote\_count = up\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

ELSE -- 'down'

UPDATE public.waypoints SET down\_vote\_count = down\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

END IF;

END IF;

-- Handle DELETE operations (Hard Deletes)

ELSIF (TG\_OP = 'DELETE') THEN

IF OLD.deleted\_at IS NULL THEN -- Only reverse active votes

IF OLD.vote\_type = 'up' THEN

UPDATE public.waypoints SET up\_vote\_count = GREATEST(0, up\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

ELSE -- 'down'

UPDATE public.waypoints SET down\_vote\_count = GREATEST(0, down\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

END IF;

END IF;

-- Handle UPDATE operations

ELSIF (TG\_OP = 'UPDATE') THEN

-- Case 1: Vote retracted (deleted\_at changed from NULL to NOT NULL)

IF OLD.deleted\_at IS NULL AND NEW.deleted\_at IS NOT NULL THEN

IF OLD.vote\_type = 'up' THEN

UPDATE public.waypoints SET up\_vote\_count = GREATEST(0, up\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

ELSE -- 'down'

UPDATE public.waypoints SET down\_vote\_count = GREATEST(0, down\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

END IF;

-- Case 2: Vote reinstated (deleted\_at changed from NOT NULL to NULL)

ELSIF OLD.deleted\_at IS NOT NULL AND NEW.deleted\_at IS NULL THEN

IF NEW.vote\_type = 'up' THEN

UPDATE public.waypoints SET up\_vote\_count = up\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

ELSE -- 'down'

UPDATE public.waypoints SET down\_vote\_count = down\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

END IF;

-- Case 3: Active vote type changed (deleted\_at was NULL and remains NULL)

ELSIF OLD.deleted\_at IS NULL AND NEW.deleted\_at IS NULL AND OLD.vote\_type <> NEW.vote\_type THEN

IF OLD.vote\_type = 'up' THEN -- Decrement old up-vote

UPDATE public.waypoints SET up\_vote\_count = GREATEST(0, up\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

ELSE -- Decrement old down-vote

UPDATE public.waypoints SET down\_vote\_count = GREATEST(0, down\_vote\_count - 1) WHERE id = OLD.waypoint\_id;

END IF;

IF NEW.vote\_type = 'up' THEN -- Increment new up-vote

UPDATE public.waypoints SET up\_vote\_count = up\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

ELSE -- Increment new down-vote

UPDATE public.waypoints SET down\_vote\_count = down\_vote\_count + 1 WHERE id = NEW.waypoint\_id;

END IF;

END IF;

END IF;

RETURN NULL; -- Result is ignored since this is an AFTER trigger

END;

$$ LANGUAGE plpgsql SECURITY DEFINER; -- SECURITY DEFINER if waypoints table has restrictive RLS for users

-- Apply the vote count trigger

CREATE TRIGGER after\_user\_waypoint\_votes\_change

AFTER INSERT OR UPDATE OR DELETE ON public.user\_waypoint\_votes

FOR EACH ROW EXECUTE FUNCTION public.update\_waypoint\_vote\_counts();

-- RLS Policies

ALTER TABLE public.user\_waypoint\_votes ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Pilgrims can insert their own votes"

ON public.user\_waypoint\_votes FOR INSERT

TO authenticated

WITH CHECK (auth.uid() = profile\_id);

CREATE POLICY "Pilgrims can update their own votes"

ON public.user\_waypoint\_votes FOR UPDATE

TO authenticated

USING (auth.uid() = profile\_id)

WITH CHECK (auth.uid() = profile\_id);

CREATE POLICY "Pilgrims can view their own votes"

ON public.user\_waypoint\_votes FOR SELECT

TO authenticated

USING (auth.uid() = profile\_id);

-- Optional: Grant direct read access to active votes if needed by clients beyond aggregated counts.

-- CREATE POLICY "Authenticated users can view active public votes"

-- ON public.user\_waypoint\_votes FOR SELECT

-- TO authenticated

-- USING (deleted\_at IS NULL);

COMMENT ON POLICY "Pilgrims can insert their own votes" ON public.user\_waypoint\_votes IS 'Allows authenticated users to insert their own votes.';

COMMENT ON POLICY "Pilgrims can update their own votes" ON public.user\_waypoint\_votes IS 'Allows authenticated users to update their own votes (change type, retract, or reinstate).';

COMMENT ON POLICY "Pilgrims can view their own votes" ON public.user\_waypoint\_votes IS 'Allows authenticated users to view their own vote records.';

```

4\. JSON Schema Mirror

JSON

```

{

"title": "user\_waypoint\_vote",

"description": "Records an authenticated user's 'thumbs up' or 'thumbs down' vote for a specific waypoint. Handles vote changes and retractions via soft delete. Version 2.1.1.",

"type": "object",

"properties": {

"profile\_id": {

"type": "string",

"format": "uuid",

"description": "FK to profiles.id of the user who cast the vote. Part of the composite PK."

},

"waypoint\_id": {

"type": "integer",

"description": "FK to waypoints.id of the waypoint being voted on. Part of the composite PK. CRITICAL: Confirm data type matches waypoints.id."

},

"vote\_type": {

"$ref": "#/definitions/vote\_type\_enum",

"description": "The type of vote cast ('up' or 'down')."

},

"vote\_source\_platform": {

"type": ["string", "null"],

"description": "Optional: Platform from which the vote was cast (e.g., 'web\_app\_v1', 'mobile\_ios\_v1'). For analytics."

},

"created\_at": {

"type": "string",

"format": "date-time",

"description": "Timestamp of when the vote was initially cast.",

"readOnly": true

},

"updated\_at": {

"type": "string",

"format": "date-time",

"description": "Timestamp of when the vote record was last updated (vote type changed or soft deleted/restored). Auto-updated by trigger.",

"readOnly": true

},

"deleted\_at": {

"type": ["string", "null"],

"format": "date-time",

"description": "Timestamp for soft deletion. If set, the vote is considered retracted/inactive.",

"readOnly": true

}

},

"required": [

"profile\_id",

"waypoint\_id",

"vote\_type",

"created\_at",

"updated\_at"

],

"primary\_key": ["profile\_id", "waypoint\_id"],

"definitions": {

"vote\_type\_enum": {

"type": "string",

"enum": ["up", "down"],

"description": "The type of vote: 'up' or 'down'."

}

}

}

```

5\. Relationships & Integrity

- Foreign Keys:

- `profile\_id` references `public.profiles(id)` (`ON DELETE CASCADE`).

- `waypoint\_id` references `public.waypoints(id)` (`ON DELETE CASCADE`).

- Junction Table: Acts as a many-to-many junction between `profiles` and `waypoints` for votes.

- `vote\_type\_enum` remains an ENUM.

- Mermaid ER Snippet:

Code snippet

```

erDiagram

profiles { uuid id PK }

waypoints { integer id PK "CRITICAL: Confirm type" }

user\_waypoint\_votes {

uuid profile\_id PK, FK

integer waypoint\_id PK, FK "CRITICAL: Confirm type"

vote\_type\_enum vote\_type

text vote\_source\_platform

timestamptz created\_at

timestamptz updated\_at

timestamptz deleted\_at

}

profiles ||--o{ user\_waypoint\_votes : "casts"

waypoints ||--o{ user\_waypoint\_votes : "receives\_vote\_for"

waypoints {

integer up\_vote\_count "DENORMALIZED"

integer down\_vote\_count "DENORMALIZED"

}

```

6\. Multilingual Strategy

- Not Directly Applicable: This table stores `vote\_type` ('up', 'down') which are internal system values. UI representation is handled by the frontend.

- `vote\_source\_platform` is for internal analytics and not typically translated.

7\. Role-Based Workflow & RLS Notes

- Key Fields for Workflow: `profile\_id`, `created\_at`, `updated\_at`, `deleted\_at`.

- RLS Policies:

- Table RLS is enabled.

- Authenticated users can insert their own votes (`WITH CHECK (auth.uid() = profile\_id)`).

- Authenticated users can update their own votes (`USING (auth.uid() = profile\_id) WITH CHECK (auth.uid() = profile\_id)`).

- Authenticated users can view their own votes (`USING (auth.uid() = profile\_id)`).

- Anonymous access (`anon` role) should generally be denied.

- Admin access is typically via `service\_role` or specific admin policies if needed for direct manipulation (not common for this table).

8\. ENUM vs Lookup Discussion

- `vote\_type\_enum` (`'up'`, `'down'`): Remains an ENUM. This is appropriate as values are simple, binary, and unlikely to need extended metadata or i18n for the ENUM values themselves.

9\. UI/UX Enablement

- Columns Powering UI: `vote\_type` (indirectly via aggregated counts on `waypoints`), `deleted\_at` (for UI state of user's own vote).

- Performance for UX:

- CRITICAL: Denormalized `up\_vote\_count` and `down\_vote\_count` on the `waypoints` table, updated by the `update\_waypoint\_vote\_counts` trigger, are essential for fast display and sorting.

- Composite PK `(profile\_id, waypoint\_id)` is efficient for checking existing votes.

10\. Auditing & Lifecycle Management

- Audit Columns: `created\_at`, `updated\_at`. `profile\_id` serves as the implicit creator/updater ID due to RLS ownership rules. Separate `created\_by\_profile\_id`/`updated\_by\_profile\_id` are deemed redundant here.

- Soft Deletion: `deleted\_at` for vote retractions.

- Data Integrity: `ON DELETE CASCADE` on FKs prevents orphaned votes.

11\. Scalability & Future-Proofing

- Partitioning: Not anticipated as necessary for V1/V2.

- Vote Abuse/Spam: Future consideration (V2+); schema could be extended (e.g., `ip\_address\_hash`).

12\. Seed Data

- Not applicable; this table contains user-generated transactional data.

13\. Next-Action Checklist

1. 🔴 CRITICAL: Confirm `user\_waypoint\_votes.waypoint\_id` Data Type: Rigorously verify and align the `integer` data type for `user\_waypoint\_votes.waypoint\_id` with `public.waypoints.id`. Adjust DDL for `user\_waypoint\_votes` and the `update\_waypoint\_vote\_counts` trigger function if `waypoints.id` is different (e.g., `bigint`, `uuid`).

2. 🔴 CRITICAL: Implement `waypoints` Table Modifications: Ensure `up\_vote\_count INTEGER NOT NULL DEFAULT 0` and `down\_vote\_count INTEGER NOT NULL DEFAULT 0` columns are added to `public.waypoints`.

3. 🔴 CRITICAL: Implement and Test `update\_waypoint\_vote\_counts` Trigger: Create and assign this trigger to `public.user\_waypoint\_votes`. Ensure the trigger function correctly references the `waypoints.id` column (type and name) and consider making it `SECURITY DEFINER` if `waypoints` RLS would prevent direct updates by the user triggering the vote. If `SECURITY DEFINER` is used, ensure `search\_path` is hardened.

4. 🟢 RLS Policies: Implement and test all defined RLS policies.

5. 🟢 `updated\_at` Trigger: Ensure `public.handle\_updated\_at()` and its trigger are correctly implemented.

6. 🟢 Application Logic: Develop robust backend logic for vote submission/UPSERT pattern.

7. 🟢 Backfill Counts: If `waypoints` data exists, run a one-time script to populate denormalized vote counts.

8. 🟢 Documentation: Note the deliberate omission of `created\_by\_profile\_id`/`updated\_by\_profile\_id` due to RLS ownership model.