<https://gemini.google.com/u/1/app/9ed6b32c76b55d3a?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/8d62cfd96b565ac8

<https://gemini.google.com/u/1/app/2d30425fc8a5984a>

\* \* \* \* \*

### Updated Production-Ready Specification

Table Name: `public.route\_segments`

1\. Purpose & Primary Use-Cases

This linking table defines the precise, ordered sequence of segments that constitute a specific route. It is fundamental for constructing navigable paths, displaying coherent itineraries, and enabling the accurate aggregation of route-level statistics like total distance and elevation gain via a critical trigger mechanism.

Key User-Story Touchpoints:

- Pilgrim (Anna) - Story A2 (Daily Stage Understanding): This table directly provides the sequence of segments that the application uses to build and display daily "stages" of a chosen route.

- Pilgrim (Anna) - Story A7 (Using Pre-defined Itineraries): If curated itineraries are built upon standard routes, this table defines the composition of those underlying routes.

- Platform Administrator (Admin Team) - Story D1 & D2: Admins manage the assembly of segments into routes using this table.

- Application Logic - Route Statistics Calculation: 🔴 DML operations (INSERT, UPDATE, DELETE) on this table are critical triggers for updating aggregated statistics on the parent `routes` table.

2\. Schema

| Column Name | Data Type | Constraints | Description (Translatable fields marked) |

| `id` | `bigint` | PRIMARY KEY, GENERATED ALWAYS AS IDENTITY | Unique identifier for this specific route-segment link record. |

| `route\_id` | `bigint` | NOT NULL, REFERENCES `public.routes(id)` ON DELETE CASCADE | The ID of the route this segment sequence belongs to. |

| `segment\_id` | `bigint` | NOT NULL, REFERENCES `public.segments(id)` ON DELETE CASCADE | The ID of the segment that forms part of this route. |

| `order\_in\_route` | `integer` | NOT NULL, CHECK (`order\_in\_route` > 0) | The sequential, 1-based order of this segment within this route. Critical for path reconstruction. |

| `contextual\_notes\_for\_segment\_in\_route` | `text` | | Optional notes specific to this segment when it is part of this particular route (English base). (Translatable via `public.translations`) |

| `created\_at` | `timestamptz` | NOT NULL DEFAULT `now()` | Timestamp indicating when this link record was created. |

| `created\_by\_profile\_id` | `uuid` | REFERENCES `public.profiles(id)` ON DELETE SET NULL | Profile ID of the user who created this link. |

| `updated\_at` | `timestamptz` | NOT NULL DEFAULT `now()` | Timestamp indicating when this link record was last updated (auto-updated by trigger). |

| `updated\_by\_profile\_id` | `uuid` | REFERENCES `public.profiles(id)` ON DELETE SET NULL | Profile ID of the user who last updated this link. |

| | | UNIQUE (`route\_id`, `order\_in\_route`) | Ensures segment order is unique for each route. |

3\. PostgreSQL DDL

SQL

```

-- This DDL assumes that 'public.routes', 'public.segments',

-- and 'public.profiles' tables already exist.

CREATE TABLE public.route\_segments (

id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- [cite: 624]

route\_id BIGINT NOT NULL REFERENCES public.routes(id) ON DELETE CASCADE, -- [cite: 624]

segment\_id BIGINT NOT NULL REFERENCES public.segments(id) ON DELETE CASCADE, -- [cite: 624]

order\_in\_route INTEGER NOT NULL CHECK (order\_in\_route > 0), -- [cite: 624]

contextual\_notes\_for\_segment\_in\_route TEXT, -- [cite: 624]

created\_at TIMESTAMPTZ NOT NULL DEFAULT now(), -- [cite: 624]

created\_by\_profile\_id UUID REFERENCES public.profiles(id) ON DELETE SET NULL, -- [cite: 624]

updated\_at TIMESTAMPTZ NOT NULL DEFAULT now(), -- [cite: 624]

updated\_by\_profile\_id UUID REFERENCES public.profiles(id) ON DELETE SET NULL, -- [cite: 624]

UNIQUE (route\_id, order\_in\_route) -- [cite: 625]

);

-- Indexes

CREATE INDEX IF NOT EXISTS idx\_route\_segments\_route\_id ON public.route\_segments(route\_id); -- [cite: 626]

CREATE INDEX IF NOT EXISTS idx\_route\_segments\_segment\_id ON public.route\_segments(segment\_id); -- [cite: 626]

-- The UNIQUE constraint on (route\_id, order\_in\_route) automatically creates an efficient index. [cite: 627]

-- Trigger for standard audit fields (created\_by, updated\_by, updated\_at)

CREATE OR REPLACE FUNCTION public.set\_route\_segment\_modification\_meta()

RETURNS TRIGGER AS $$

BEGIN

NEW.updated\_at = NOW(); -- [cite: 628]

IF (TG\_OP = 'INSERT') THEN

NEW.created\_by\_profile\_id = auth.uid(); -- [cite: 629]

NEW.updated\_by\_profile\_id = auth.uid(); -- [cite: 629]

ELSIF (TG\_OP = 'UPDATE') THEN

NEW.updated\_by\_profile\_id = auth.uid(); -- [cite: 630]

NEW.created\_at = OLD.created\_at; -- [cite: 630]

NEW.created\_by\_profile\_id = OLD.created\_by\_profile\_id; -- [cite: 630]

END IF;

RETURN NEW; -- [cite: 631]

END;

$$ LANGUAGE plpgsql SECURITY DEFINER; -- [cite: 631]

CREATE TRIGGER trigger\_route\_segments\_modification\_meta

BEFORE INSERT OR UPDATE ON public.route\_segments

FOR EACH ROW

EXECUTE FUNCTION public.set\_route\_segment\_modification\_meta(); -- [cite: 632]

-- 🔴 Crucial: Trigger to update routes.total\_distance\_km and routes.estimated\_total\_elevation\_gain\_meters

CREATE OR REPLACE FUNCTION public.update\_route\_aggregates\_from\_segments()

RETURNS TRIGGER AS $$

DECLARE

v\_route\_id BIGINT; -- [cite: 633]

BEGIN

IF (TG\_OP = 'DELETE') THEN

v\_route\_id := OLD.route\_id; -- [cite: 634]

ELSE -- INSERT or UPDATE

v\_route\_id := NEW.route\_id; -- [cite: 635]

END IF;

-- This assumes segments table's own triggers have fired or data is accurate. [cite: 637]

UPDATE public.routes

SET

total\_distance\_km = (

SELECT COALESCE(SUM(s.distance\_km), 0)

FROM public.route\_segments rs

JOIN public.segments s ON rs.segment\_id = s.id

WHERE rs.route\_id = v\_route\_id

), -- [cite: 638]

estimated\_total\_elevation\_gain\_meters = (

SELECT COALESCE(SUM(s.elevation\_gain\_meters), 0)

FROM public.route\_segments rs

JOIN public.segments s ON rs.segment\_id = s.id

WHERE rs.route\_id = v\_route\_id

) -- [cite: 639]

WHERE id = v\_route\_id; -- [cite: 639]

IF (TG\_OP = 'UPDATE' AND NEW.route\_id IS DISTINCT FROM OLD.route\_id) THEN

UPDATE public.routes

SET

total\_distance\_km = (

SELECT COALESCE(SUM(s.distance\_km), 0)

FROM public.route\_segments rs

JOIN public.segments s ON rs.segment\_id = s.id

WHERE rs.route\_id = OLD.route\_id

), -- [cite: 641]

estimated\_total\_elevation\_gain\_meters = (

SELECT COALESCE(SUM(s.elevation\_gain\_meters), 0)

FROM public.route\_segments rs

JOIN public.segments s ON rs.segment\_id = s.id

WHERE rs.route\_id = OLD.route\_id

) -- [cite: 642]

WHERE id = OLD.route\_id; -- [cite: 643]

END IF;

RETURN NULL; -- Result is ignored since this is an AFTER trigger [cite: 644]

END;

$$ LANGUAGE plpgsql SECURITY DEFINER; -- [cite: 644]

CREATE TRIGGER trigger\_route\_segments\_after\_change\_update\_route\_stats

AFTER INSERT OR UPDATE OR DELETE ON public.route\_segments

FOR EACH ROW

EXECUTE FUNCTION public.update\_route\_aggregates\_from\_segments(); -- [cite: 645]

-- Comments

COMMENT ON TABLE public.route\_segments IS 'Defines the ordered sequence of segments that constitute a specific route. This table is critical for path construction and route statistic aggregation. Version: V2.'; -- [cite: 646]

COMMENT ON COLUMN public.route\_segments.id IS 'Unique identifier for this route-segment link record. Version: V2.'; -- [cite: 647]

COMMENT ON COLUMN public.route\_segments.route\_id IS 'Foreign key to the parent public.routes. Version: V2.'; -- [cite: 648]

COMMENT ON COLUMN public.route\_segments.segment\_id IS 'Foreign key to the constituent public.segments. Version: V2.'; -- [cite: 649]

COMMENT ON COLUMN public.route\_segments.order\_in\_route IS 'The sequential, 1-based order of this segment within this route. Critical for path reconstruction and defining direction. Version: V2.'; -- [cite: 651]

COMMENT ON COLUMN public.route\_segments.contextual\_notes\_for\_segment\_in\_route IS 'Optional notes specific to this segment when it is part of this particular route (English base, translatable via public.translations table using route\_segments.id). Version: V2.'; -- [cite: 652]

COMMENT ON COLUMN public.route\_segments.created\_at IS 'Timestamp of when this route-segment link was created. Version: V2.'; -- [cite: 653]

COMMENT ON COLUMN public.route\_segments.created\_by\_profile\_id IS 'Profile ID (public.profiles.id) of the user who created this link. Version: V2.'; -- [cite: 654]

COMMENT ON COLUMN public.route\_segments.updated\_at IS 'Timestamp of when this link record was last updated. Auto-updated by trigger. Version: V2.'; -- [cite: 655]

COMMENT ON COLUMN public.route\_segments.updated\_by\_profile\_id IS 'Profile ID (public.profiles.id) of the user who last updated this link. Version: V2.'; -- [cite: 656]

```

4\. JSON Schema Mirror

JSON

```

{

"title": "route\_segment\_link",

"description": "Defines the ordered sequence of segments that make up a specific route, linking a route to its constituent path segments. Also triggers updates to route-level aggregate statistics. Version: V2.",

"type": "object",

"properties": {

"id": {

"type": "integer",

"format": "int64",

"description": "Unique identifier for this route-segment link record. Read-only.", -- [cite: 657]

"readOnly": true

},

"route\_id": {

"type": "integer",

"format": "int64",

"description": "Foreign key referencing the ID of the route (public.routes.id) this segment sequence belongs to." -- [cite: 658]

},

"segment\_id": {

"type": "integer",

"format": "int64",

"description": "Foreign key referencing the ID of the segment (public.segments.id) being linked as part of the route." -- [cite: 659]

},

"order\_in\_route": {

"type": "integer",

"minimum": 1,

"description": "The sequential, 1-based order of this segment within this specific route. Defines the path construction sequence." -- [cite: 660]

},

"contextual\_notes\_for\_segment\_in\_route": {

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

"description": "Optional notes specific to this segment's role or characteristics when it is part of this particular route (English base, translatable using route\_segments.id)." -- [cite: 661]

},

"created\_at": {

"type": "string",

"format": "date-time",

"description": "Timestamp of when this link record was created. Read-only.", --

"readOnly": true

},

"created\_by\_profile\_id": {

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

"format": "uuid",

"description": "Profile ID (public.profiles.id) of the user who created this link. Read-only.", --

"readOnly": true

},

"updated\_at": {

"type": "string",

"format": "date-time",

"description": "Timestamp of when this link record was last updated. Read-only.", -- [cite: 662]

"readOnly": true

},

"updated\_by\_profile\_id": {

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

"format": "uuid",

"description": "Profile ID (public.profiles.id) of the user who last updated this link. Read-only.", --

"readOnly": true

}

},

"required": [

"route\_id",

"segment\_id",

"order\_in\_route", -- [cite: 663]

"created\_at",

"updated\_at"

],

"primary\_key": ["id"], -- [cite: 663]

"unique\_constraints": [

{"columns": ["route\_id", "order\_in\_route"], "name": "route\_segments\_route\_id\_order\_in\_route\_key"} -- [cite: 663]

]

}

```

5\. Relationships & Integrity

- Foreign Key Targets & `ON DELETE` Actions:

- `route\_id` references `public.routes(id)`: `ON DELETE CASCADE`. If a route is deleted, its sequence definition is also deleted.

- `segment\_id` references `public.segments(id)`: `ON DELETE CASCADE`. If a segment is deleted, it's removed from all route sequences.

- `created\_by\_profile\_id`, `updated\_by\_profile\_id` references `public.profiles(id)`: `ON DELETE SET NULL`.

- Primary Key: Surrogate `id` (`BIGINT`) for easy row reference, e.g., by the `translations` table.

- Unique Constraint: `UNIQUE (route\_id, order\_in\_route)` is critical for an unambiguous path for each route.

- Segment Reusability: No `UNIQUE (route\_id, segment\_id)` constraint allows a segment to appear multiple times in the same route if needed (e.g., for loops).

- Mermaid ER Diagram Snippet:

Code snippet

```

erDiagram

routes {

bigint id PK

text name

real total\_distance\_km

integer estimated\_total\_elevation\_gain\_meters

}

segments {

bigint id PK

text name

real distance\_km

integer elevation\_gain\_meters

}

profiles {

uuid id PK

}

route\_segments {

bigint id PK --

bigint route\_id FK --

bigint segment\_id FK --

integer order\_in\_route --

text contextual\_notes\_for\_segment\_in\_route --

timestamptz created\_at --

uuid created\_by\_profile\_id FK --

timestamptz updated\_at --

uuid updated\_by\_profile\_id FK --

-- UNIQUE(route\_id, order\_in\_route)

}

routes ||--|{ route\_segments : "composed\_of (CASCADE)" -- [cite: 680]

segments ||--|{ route\_segments : "is\_part\_of (CASCADE)" -- [cite: 680]

route\_segments }o--|| profiles : "created\_by (SET NULL)" -- [cite: 681]

route\_segments }o--|| profiles : "updated\_by (SET NULL)" -- [cite: 682]

```

6\. Multilingual Strategy

- Translatable Field: `contextual\_notes\_for\_segment\_in\_route`. English base text is stored here.

- Linking Mechanism: Translations via `public.translations` table, using `table\_name='route\_segments'`, `column\_name='contextual\_notes\_for\_segment\_in\_route'`, and `row\_id=route\_segments.id`.

7\. Role-Based Workflow & RLS Notes

- Workflow Fields: Audit columns (`created\_at`, `created\_by\_profile\_id`, `updated\_at`, `updated\_by\_profile\_id`) track link creation/modification.

- Note: The RLS policies outlined above rely on the existence and correct implementation of global RLS helper functions (e.g., public.has\_role(TEXT), public.is\_platform\_admin(), specific regional/trail management checks) that authenticate users and verify their roles stored in the public.profiles table." This reinforces that the table-specific RLS is part of a larger auth system.

- RLS Policy Stubs (Conceptual):

- Public Read Access: If parent route and trail are published.

SQL

```

CREATE POLICY "Allow public read access to route\_segments"

ON public.route\_segments FOR SELECT

USING (

EXISTS (

SELECT 1 FROM public.routes r

JOIN public.trails t ON r.trail\_id = t.id

WHERE r.id = route\_segments.route\_id

AND r.content\_visibility\_status = 'published' AND r.deleted\_at IS NULL

AND t.content\_visibility\_status = 'published' AND t.deleted\_at IS NULL

)

); -- [cite: 689]

```

- Admin Full Access: Full CRUD via `public.is\_platform\_admin()`.

- Regional Content Manager Access: Conditional access via `public.is\_regional\_manager\_for\_route(BIGINT)`.

- Audit trigger `set\_route\_segment\_modification\_meta` is `SECURITY DEFINER`.

8\. ENUM vs. Lookup Discussion

Not applicable directly; this is a junction table.

9\. UI/UX Enablement

- Core Data for Route Construction: `route\_id`, `segment\_id`, `order\_in\_route` are fundamental for application logic.

- Contextual Information: `contextual\_notes\_for\_segment\_in\_route` for route-specific segment insights.

- 🔴 Trigger for Route Statistics: DML operations here trigger updates to `routes.total\_distance\_km` and `routes.estimated\_total\_elevation\_gain\_meters` ensuring UI accuracy.

10\. Key Considerations & Definitions

- `order\_in\_route` Integrity: Paramount for correct path representation; must be a positive, sequential integer per `route\_id`. Application layer manages this sequence.

- Lifecycle: Dependent on parent `routes` and `segments` due to `ON DELETE CASCADE`.

- Route Directionality: Implicitly defined by `order\_in\_route`. Reverse routes should be separate `routes` records.

- Atomicity of Changes: Route assembly/modification operations should be transactional.

11\. Scalability & Future-Proofing

- `BIGINT` PKs/FKs support growth.

- Audit columns provide traceability.

- No soft deletes needed; lifecycle tied to parents.

- Performance: Indexes on FKs and the `UNIQUE (route\_id, order\_in\_route)` constraint are crucial. The aggregate update trigger's performance for very long routes might be a V2+ consideration.

12\. Next-Action Checklist

1. 🔴 Implement DDL: Create the `public.route\_segments` table with all constraints (PK, FKs with `ON DELETE CASCADE`, `CHECK`, `UNIQUE`), indexes, and the `set\_route\_segment\_modification\_meta` trigger.

2. 🔴 Implement and Test Route Aggregates Trigger: Ensure `public.update\_route\_aggregates\_from\_segments` trigger function is correctly implemented and thoroughly tested for `INSERT`, `UPDATE`, and `DELETE` operations on `public.route\_segments`, verifying accurate updates to `routes` table statistics.

3. 🔴 Implement Orphaned Translation Cleanup Trigger: Add an `AFTER DELETE` trigger on `public.route\_segments` that calls `public.cleanup\_related\_translations('route\_segments', OLD.id)` to remove orphaned translations for `contextual\_notes\_for\_segment\_in\_route`.

4. 🟠 Implement RLS Policies & Helper Functions: Define, implement, and test RLS policies for `public.route\_segments`, including helper functions like `public.is\_platform\_admin()` and `public.is\_regional\_manager\_for\_route(BIGINT)`.

5. 🟢 Review Application Logic: Ensure the application layer correctly manages `order\_in\_route` during route assembly and modification.

6. 🟢 Data Migration (if applicable): Plan and execute scripts to populate `route\_segments`, mapping `route\_id`, `segment\_id`, and `order\_in\_route`. Populate audit columns.

\* \* \* \* \*