



Les 8

Supabase CRUD

INSERT, UPDATE, DELETE + server actions + voting



Terugblik

Les 7 - Supabase setup

Account + project

Region NL, free tier

Tabellen

polls + options met FK

RLS intro

Open policy voor anon

Client + SELECT

Server Component leest DB

Vandaag: schrijven naar de DB.

INSERT (create), UPDATE (mutate), DELETE.

Server Actions + RPC voor voting + TanStack Query basics.

Eind van les: QuickPoll werkt end-to-end op echte database.

Planning

Vandaag - 180 minuten

Welkom + terugblik 10 min

INSERT + /create pagina 20 min

Server actions met Supabase 20 min

LIVE DEMO 1 - /create end-to-end 20 min

Pauze 15 min

UPDATE + DELETE + RPC voor voting 25 min

Types generation + TanStack Query 20 min

Zelfstandig: QuickPoll afmaken 40 min

Huiswerk + afsluiting 10 min

Theorie 1

INSERT - data toevoegen

```
const { data, error } = await supabase
  .from("polls")
  .insert({ title: "Nieuwe poll" })
  .select()
  .single();

if (error) return { error: error.message };
```

.select().single() = krijg ingevoegde row terug.

Meerdere tegelijk:

```
await supabase.from("options").insert([
  { poll_id: 1, text: "Optie A" },
  { poll_id: 1, text: "Optie B" },
  { poll_id: 1, text: "Optie C" },
]);
```

Tip: .select('id, title, created_at') om te bepalen welke kolommen je terugkrijgt.

Pattern

/create pagina + Server Action

```
// app/create/page.tsx
import { createPoll } from "../actions";

export default function CreatePage() {
  return (
    <form action={createPoll}>
      <input name="title" required />
      <input name="options" placeholder="comma-separated" />
      <button>Aanmaken</button>
    </form>
  );
}
```

```
// app/create/actions.ts
"use server";
import { supabase } from "@lib/supabase";
import { redirect } from "next/navigation";
import { revalidatePath } from "next/cache";

export async function createPoll(formData: FormData) {
  const title = formData.get("title") as string;
  // ... insert + revalidate + redirect
}
```

LIVE DEMO 1

/create end-to-end - ~20 min

- 1 app/create/page.tsx - form met name='title' en 'options'
- 2 app/create/actions.ts met 'use server'
- 3 Parse FormData -> validatie
- 4 Insert poll -> krijg id
- 5 Insert opties met poll_id
- 6 revalidatePath('/')
- 7 redirect(`/poll/\${id}`)

Server Action pattern - kort:

use server -> validate -> Supabase call -> revalidate/redirect.



Pauze

15 minuten

Theorie 2

UPDATE + DELETE

UPDATE:

```
await supabase
  .from("polls")
  .update({ title: "Bijgewerkt" })
  .eq("id", 1);
```

DELETE:

```
await supabase.from("polls").delete().eq("id", 1);
```

GEVAAR:

```
// ZONDER eq -> hele tabel leeg!  
await supabase.from("polls").delete(); // BAD  
await supabase.from("polls").delete().eq("id", 1); // OK
```


Theorie 3

RPC - voting via stored procedure

Increment is geen native Supabase call -> stored procedure (RPC):

```
-- In Supabase SQL editor:  
create function increment_vote(option_id_input bigint)  
returns void as $$  
  update options set votes = votes + 1  
  where id = option_id_input;  
$$ language sql;
```

In Next.js code:

```
await supabase.rpc("increment_vote", {  
  option_id_input: 5  
});
```

Waarom RPC? Atomic increment - geen race conditions tussen votes.

Postgres doet 't atomically. Alternatieven: optimistic locking.

Pattern

Voting in QuickPoll

```
// components/VoteForm.tsx
<form action={vote}>
  <input type="hidden" name="pollId" value={pollId} />
  {options.map(o => (
    <label key={o.id}>
      <input type="radio" name="optionId" value={o.id} />
      {o.text} ({o.votes} stemmen)
    </label>
  ))}
  <button>Stem</button>
</form>
```

```
// actions.ts
"use server";
export async function vote(formData: FormData) {
  const pollId = Number(formData.get("pollId"));
  const optionId = Number(formData.get("optionId"));
  await supabase.rpc("increment_vote", { option_id_input: optionId });
  revalidatePath(`/poll/${pollId}`);
}
```

Theorie 4

TanStack Query - client-side data

Voor live updates en optimistic UI: `useEffect` + `fetch` is matig.

TanStack Query = standaard:

```
pnpm add @tanstack/react-query
```

```
"use client";
import { useQuery } from "@tanstack/react-query";

export function LivePollResults({ pollId }) {
  const { data, isLoading } = useQuery({
    queryKey: ["poll", pollId],
    queryFn: () => fetch(`/api/polls/${pollId}`).then(r => r.json()),
    refetchInterval: 2000, // poll elke 2 sec
  });
  ...
}
```

Voor mutations: `useMutation`. Cache invalidation: `queryClient.invalidateQueries`.

Theorie 5

Types generation

Supabase genereert TS types uit je schema. Geen handmatig typen meer.

```
pnpm add -D supabase
pnpm dlx supabase login
pnpm dlx supabase link --project-ref <project-id>
pnpm dlx supabase gen types typescript --linked > types/database.ts
```

Gebruik:

```
import type { Database } from "@types/database";

export const supabase = createClient<Database>(...);
const { data } = await supabase.from("polls").select("title");
// data is typed: { title: string }[] | null
```

Herhaal na schema-wijzigingen. Of automatiseer in CI.

Theorie 6

Error handling - productie-pattern

```
async function safeSupabaseCall<T>(  
  fn: () => Promise<{ data: T | null; error: Error | null }>  
) : Promise<{ ok: true; data: T } | { ok: false; error: string }> {  
  const { data, error } = await fn();  
  if (error) return { ok: false, error: error.message };  
  if (data === null) return { ok: false, error: "No data" };  
  return { ok: true, data };  
}
```

```
const result = await safeSupabaseCall(() =>  
  supabase.from("polls").select("*").single()  
);  
if (!result.ok) return <Error message={result.error} />;  
return <Poll poll={result.data} />;
```

Result<T, E> pattern = discriminated union van Les 4. TypeScript shines.

LESOPDRACHT

QuickPoll afmaken (~40 min)

Checklist:

- v /create pagina met form + server action
- v Insert poll + opties in transaction-stijl
- v Voting werkt via RPC (atomic increment)
- v revalidatePath na elke mutation
- v Types gegenereerd uit Supabase
- v Optioneel: TanStack Query op poll detail

Eindstaat - wat draait er:

- Homepage = lijst polls vanuit DB
- /create = nieuwe poll form
- /poll/[id] = detail + voting
- Database = polls + options met RLS open

Huiswerk

Voor Les 9 (Auth start)

1. QuickPoll productie-klaar

Insert/update/delete + voting werkt op Supabase

2. Types genereren in package.json

Add script: 'gen:types' voor herbruikbaar

3. Deploy naar Vercel

Env vars in Vercel zetten - test productie URL

4. Inleveren via Teams

Live Vercel URL + GitHub repo

Tip: Vercel - voeg .env.local vars toe via Vercel Dashboard -> Settings.

Volgende les

Les 9 - Supabase Auth

Iedereen kan nu alles. Tijd om dat te fixen.

- Supabase Auth basics - email + magic link
- @supabase/ssr voor Next.js Server Components
- Session in middleware + cookies
- Login / logout / protected routes
- Voorbereiding op RLS per user (Les 10 + 18)

Voorbereiden:

QuickPoll deployed op Vercel met werkende DB - we bouwen erop verder.

Vragen?

Tot volgende week!