# Upgrade to Prisma ORM 7

Prisma ORM v7 introduces **breaking changes** when you upgrade from an earlier Prisma ORM version. This guide explains how this upgrade might affect your application and gives instructions on how to handle any changes.

**Questions answered in this page**

For developers using AI Agents, we have a migration prompt that you can add to your project for automatic migrations.

> **INFO**
>
> If you are using MongoDB, please note that Prisma ORM v7 does not yet support MongoDB. You should continue using Prisma ORM v6 for now. Support for MongoDB is coming soon in v7.

## Upgrade the `prisma` and `@prisma/client` packages to v7

To upgrade to Prisma ORM v7 from an earlier version, you need to update both the `prisma` and `@prisma/client` packages:

npm yarn pnpm bun

```
$ npm install @prisma/client@7
$ npm install -D prisma@7
```

**DANGER**

Before you upgrade, check each breaking change below to see how the upgrade might affect your application.

# Breaking changes

This section gives an overview of breaking changes in Prisma ORM v7.

## Minimum supported Node.js & TypeScript versions

| | Minimum Version | Recommended |
|---|---|---|
| Node | 20.19.0 | 22.x |

| | Minimum Version | Recommended |
|---|---|---|
| TypeScript | 5.4.0 | 5.9.x |

## ESM support

Prisma ORM now ships as an ES module, the module format supported in Bun, Deno, and Node. Set the `type` field in your `package.json` to `module`

```
{
  "type": "module",
  "scripts": {...},
}
```

If you are using TypeScript, you need to configure your `tsconfig.json` to be able to consume ES modules

```
{
  "compilerOptions": {
    "module": "ESNext",
    "moduleResolution": "node",
    "target": "ES2023",
    "strict": true,
    "esModuleInterop": true
  }
}
```

## Prisma schema changes

The older `prisma-client-js` provider will be removed in future releases of Prisma ORM. Upgrade to the new `prisma-client` provider which uses the new Rust-free client. This will give you faster queries, smaller bundle size, and require less system resources when deployed to your server.

Additionally, the `output` field is now **required** in the generator block. Prisma Client will no longer be generated in `node_modules` by default. You must specify a custom output path.

### Before

```
generator client {
  provider = "prisma-client-js"
  engineType = "binary"
}
```

### After

```
generator client {
  provider = "prisma-client"
  output   = "./generated/prisma"
}
```

After running `npx prisma generate`, you'll need to update your imports to use the new generated path:

```
// Before
import { PrismaClient } from '@prisma/client'

// After
import { PrismaClient } from './generated/prisma/client'
```

> **NOTE**
>
> The import path depends on where you place your generated client. Adjust the path based on your `output` configuration and the location of the file you're importing from.

Additionally other fields such as `url`, `directUrl`, and `shadowDatabaseUrl` in the `datasource` block are deprecated. You can configure them in the Prisma Config.

If you were previously using `directUrl` to run migrations then you need to pass the `directUrl` value in the `url` field of `prisma.config.ts` instead as the connection string defined in `url` is used by Prisma CLI for migrations.

```
import 'dotenv/config'
import { defineConfig, env } from 'prisma/config'

export default defineConfig({
  datasource: {
    url: env('DATABASE_URL'),
    shadowDatabaseUrl: env('SHADOW_DATABASE_URL')
  },
})
```

## Driver adapters and client instantiation

The way to create a new Prisma Client has changed to require a driver adapter for all databases. This change aligns with the move to make the main Prisma Client as lean and open as possible. For instance, if you are using Prisma Postgres, you now need the `@prisma/adapter-pg` adapter. This also means the signature for creating a new Prisma Client has changed slightly:

> ⚠ **CONNECTION POOL DEFAULTS HAVE CHANGED**
>
> Driver adapters use the connection pool settings from the underlying Node.js database driver, which may differ significantly from Prisma ORM v6 defaults. For example, the `pg` driver has no connection timeout by default ( `0` ), while Prisma ORM v6 used a 5-second timeout.
>
> **If you experience timeout issues after upgrading**, configure your driver adapter to match v6 behavior. See the

connection pool guide for detailed configuration examples for each database.

### Before

```
import { PrismaClient } from '@prisma/client';
const prisma = new PrismaClient({
  datasources: {
    db: { url: process.env.DATABASE_URL },
  },
  datasourceUrl: process.env.DATABASE_URL,
});
```

### After

```
import { PrismaClient } from './generated/prisma/client';
import { PrismaPg } from '@prisma/adapter-pg';

const adapter = new PrismaPg({
  connectionString: process.env.DATABASE_URL
});
const prisma = new PrismaClient({ adapter });
```

If you are using SQLite, you can use the `@prisma/adapter-better-sqlite3`:

```
import { PrismaClient } from './generated/prisma/client';
import { PrismaBetterSqlite3 } from '@prisma/adapter-better-

const adapter = new PrismaBetterSqlite3({
  url: process.env.DATABASE_URL || 'file:./dev.db'
})

export const prisma = new PrismaClient({ adapter })
```

## Prisma Accelerate users upgrading from v6

If you used Prisma Accelerate (including Prisma Postgres' `prisma+postgres://` URLs) in v6, keep using the Accelerate URL with the Accelerate extension. Do **not** pass the Accelerate URL to a driver adapter— `PrismaPg` expects a direct database connection string and will fail with `prisma://` or `prisma+postgres://`.

1. Keep your Accelerate URL in `.env` (for example `DATABASE_URL="prisma://..."` or `prisma+postgres://...`).
2. You can point `prisma.config.ts` directly to that same Accelerate URL for CLI operations:



**prisma.config.ts**

```
import 'dotenv/config'
import { defineConfig, env } from 'prisma/config'

export default defineConfig({
  schema: 'prisma/schema.prisma',
  datasource: {
    url: env('DATABASE_URL'),
  },
})
```

- If you prefer a separate direct URL for migrations, you can still use `DIRECT_DATABASE_URL` as above—but it's optional for Accelerate users.

3. Instantiate Prisma Client with the Accelerate URL and extension (no adapter):

```
import { PrismaClient } from './generated/prisma/client'
import { withAccelerate } from '@prisma/extension-accelerate

export const prisma = new PrismaClient({
  accelerateUrl: process.env.DATABASE_URL,
}).$extends(withAccelerate())
```

If you later switch away from Accelerate to direct TCP, provide the direct URL to the appropriate driver adapter (for example `PrismaPg`) instead of `accelerateUrl`.

## Explicit loading of environment variables

In Prisma ORM 7.0.0, environment variables are not loaded by default. Instead developers need to explicitly load the variables when calling the `prisma` CLI. Libraries like `dotenv` can be used to manage loading environment variables by reading the appropriate `.env` file.

npm yarn pnpm

```
$ npm install dotenv
```

For bun users, no action is required as bun will automatically load `.env` files.

## Prisma config is now used to configure the Prisma CLI

Prisma Config is now the default place for configuring how the Prisma CLI interacts with your database. You now configure your database URL, schema location, migration output, and custom seed scripts.

> **INFO**
>
> The `prisma.config.ts` file should be placed at the **root of your project** (where your `package.json` is located).

```
import 'dotenv/config'
import { defineConfig, env } from 'prisma/config'

export default defineConfig({
  // the main entry for your schema
  schema: 'prisma/schema.prisma',
  // where migrations should be generated
  // what script to run for "prisma db seed"
  migrations: {
    path: 'prisma/migrations',
    seed: 'tsx prisma/seed.ts',
  },
  // The database URL
  datasource: {
    // Type Safe env() helper
    // Does not replace the need for dotenv
    url: env('DATABASE_URL'),
  },
})
```

## Metrics has been removed from the Client extensions

The Metrics preview feature was deprecated in Prisma ORM 6.14.0 and has been removed for Prisma ORM 7.0.0. If you need this feature, you can use the underlying driver adapter for your database, or Client Extensions to make this information available.

For example, a basic `totalQueries` counter:

```
const total = 0
const prisma = new PrismaClient().$extends({
  client: {
    $log: (s: string) => console.log(s),
    async $totalQueries() { return total; },
  },
  query: {
      $allModels: {
        async $allOperations({ query, args }) {
          total += 1;
          return query(args);
        },
      },
    },
})

async function main() {
  prisma.$log('Hello world')
  const totalQueries = await prisma.$totalQueries()
  console.log(totalQueries)
}
```

## Mapped enum values in generated TypeScript

> **⚠ WARNING**
>
> There is currently a known bug where using mapped enum values with Prisma Client operations (like `create`, `update`, etc.) causes runtime errors. The generated TypeScript types

> expect the mapped values, but the Prisma Client engine expects the schema names. This issue is being tracked in GitHub issue #28591. Until this is fixed, you need to use workarounds described below.

In Prisma ORM v7, the generated TypeScript enum values now use the `@map` values instead of the schema names. This is a breaking change from v6.

### Before Prisma ORM v6

Given this Prisma schema:

```
enum SuggestionStatus {
  PENDING  @map("pending")
  ACCEPTED @map("accepted")
  REJECTED @map("rejected")
}
```

In v6, the generated TypeScript enum would be:

```
export const SuggestionStatus = {
  PENDING: 'PENDING',
  ACCEPTED: 'ACCEPTED',
  REJECTED: 'REJECTED'
} as const
```

### After Prisma ORM v7

In v7, the same schema generates:

```
export const SuggestionStatus = {
  PENDING: 'pending',
  ACCEPTED: 'accepted',
  REJECTED: 'rejected'
} as const
```

This means that `SuggestionStatus.PENDING` now evaluates to `"pending"` instead of `"PENDING"`.

### Migration steps

If you're using mapped enums, you'll need to update any code that relies on the enum values being the schema names rather than the mapped values.

#### Temporary workaround

Until we resolve the open issue, you need to use the schema name as a string literal instead of the generated enum value:

```
// This may cause a TypeScript error but works at runtime
await prisma.suggestion.create({
  data: {
    status: "PENDING" as any, // Use schema name, not mapped
  },
});
```

Alternatively, you can temporarily remove the `@map` directives from your enum values if you don't strictly need the database values to differ from the schema names:

```
// Before: with @map directives
enum SuggestionStatus {
  PENDING  @map("pending")
  ACCEPTED @map("accepted")
  REJECTED @map("rejected")
}

// After: without @map directives
enum SuggestionStatus {
  PENDING
  ACCEPTED
  REJECTED
}
```

With this change, both the schema names and the database values will be `PENDING`, `ACCEPTED`, and `REJECTED`, and the generated TypeScript enum will work correctly with Prisma Client operations.

## Client middleware has been removed

The client middleware API has been removed. If possible, use Client Extensions.

```
// ❌ Old (removed)
prisma.$use(async (params, next) => {
  // middleware logic
  return next(params)
})
// ✅ New (use extensions)
const prisma = new PrismaClient().$extends({
  query: {
    user: {
      async findMany({ args, query }) {
        // extension logic
        return query(args)
      }
    }
  }
})
```

## Automatic seeding during migrations has been removed

In Prisma ORM v6 and earlier, running `prisma migrate dev` or `prisma migrate reset` would automatically execute your seed script after applying migrations. This automatic seeding behavior has been removed in Prisma ORM v7.

To seed your database in v7, you must explicitly run:

```
$ npx prisma db seed
```

## CLI flags `--skip-generate` and `--skip-seed` removed

The `--skip-generate` flag was removed from `prisma migrate dev` and `prisma db push`. The `--skip-seed` flag was removed from `prisma migrate dev`.

`migrate dev` and `db push` no longer run `prisma generate` automatically. You must run `prisma generate` explicitly to generate Prisma Client.

## `--schema` and `--url` flags removed from `prisma db execute`

The `--schema` and `--url` flags have been removed from the `prisma db execute` command. Previously, you could use `--schema` to specify the path to your Prisma schema file, or `--url` to specify the database URL directly. Now, the database connection must be configured in `prisma.config.ts`.

**Before (v6)**

```
$ # Using --schema
$ prisma db execute --file ./script.sql --schema prisma/sche
$
$ # Using --url
$ prisma db execute --file ./script.sql --url "$DATABASE_URL
```

### After (v7)

Configure your database connection in `prisma.config.ts` instead:

```
import 'dotenv/config'
import { defineConfig, env } from 'prisma/config'

export default defineConfig({
  schema: 'prisma/schema.prisma',
  datasource: {
    url: env('DATABASE_URL'),
  },
})
```

Then run the command without `--schema` or `--url`:

```
$ prisma db execute --file ./script.sql
```

## `prisma migrate diff` CLI options changed

Several options have been removed from `prisma migrate diff` and replaced with new options that use `prisma.config.ts`:

| Removed Option | Replacement |
| --- | --- |
| `--from-url` | `--from-config-datasource` |

| Removed Option | Replacement |
| --- | --- |
| `--to-url` | `--to-config-datasource` |
| `--from-schema-datasource` | `--from-config-datasource` |
| `--to-schema-datasource` | `--to-config-datasource` |
| `--shadow-database-url` | Configure in `prisma.config.ts` |

### Before (v6)

```
$ prisma migrate diff \
$   --from-url "$DATABASE_URL" \
$   --to-schema schema.prisma \
$   --script
```

### After (v7)

Configure your database connection in `prisma.config.ts`, then use `--from-config-datasource` or `--to-config-datasource`:

```
$ prisma migrate diff \
$   --from-config-datasource \
$   --to-schema schema.prisma \
$   --script
```

## Various environment variables have been removed

We've removed a small selection of Prisma-specific environment variables.

- `PRISMA_CLI_QUERY_ENGINE_TYPE`
- `PRISMA_CLIENT_ENGINE_TYPE`
- `PRISMA_QUERY_ENGINE_BINARY`
- `PRISMA_QUERY_ENGINE_LIBRARY`
- `PRISMA_GENERATE_SKIP_AUTOINSTALL`
- `PRISMA_SKIP_POSTINSTALL_GENERATE`
- `PRISMA_GENERATE_IN_POSTINSTALL`
- `PRISMA_GENERATE_DATAPROXY`
- `PRISMA_GENERATE_NO_ENGINE`
- `PRISMA_CLIENT_NO_RETRY`
- `PRISMA_MIGRATE_SKIP_GENERATE`
- `PRISMA_MIGRATE_SKIP_SEED`