Indices in MongoDB Nested Objects

When we introduce the composite types, first for MongoDB, it opens up some extra work when we think about indices and how we'd use them with the nested type definitions.

A composite type is a nested type, not defined in the database schema and in case of MongoDB, is just a nested JSON definition on collection bar:

```
{
  foo: { bar: "lol" }, lol: 2
}
```

And this translates to Prisma as

```
model bar {
  foo BarFoo
  lol Int
}

type BarFoo {
  bar String
}
```

A MongoDB index can be defined to any key in the collection. Typically you'd define an index to a field as `lol` with the following definition:

```
{ "lol": 1 }
```

Additionally, an index can be created to the nested object:

```
{ "foo.bar": 1 }
```

We do not yet have any way to define this kind of indices in the PSL. There are a few different ways to do it.

Solution #1: Index Definition in the Nested Type

Let's imagine we can define the index in the type definition:

```
model bar {
  foo BarFoo
  lol Int
}

type BarFoo {
  bar String

  @@index([bar])
}
```

Pushing this would add an index to `bar` due to it using the `BarFoo` type and the type defining an index. It gets a bit more tricky if we use this type in another collection.

```
model bar {
  foo BarFoo
  lol Int
}

model lol {
  foo BarFoo
}

type BarFoo {
  bar String

  @@index([bar])
}
```

Do we now expect to have an index in `bar` for `{ "foo.bar": 1 }` and, again, in `lol` for `{ "lol.foo": 1 }`?

Solution #2: Index Definition Always in the Model

In this version we define the index from the model.

```
model bar {
  foo BarFoo
  lol Int

  @@index([foo.bar])
}

model lol {
  foo BarFoo
}

type BarFoo {
  bar String
}
```

Here the user decides in which collection the index is created. If they need it also in the model `lol`, they'd need to add another `@@index` attribute in there.