# Bases de datos: Repaso

Relacional: Es una base de datos estructurada, los datos son organizados dentro de tablas.
No relacional: Las bases de datos no relacionales, en cambio, no mantienen relación entre documentos.

# MongoDB: Repaso

MongoDB es uno de los motores más conocidos y más utilizados de base de datos no relacional. Al igual que en el resto de DB no relacionales, la información se guarda en forma de documentos estilo JSON. (En realidad, utiliza BSON que es JSON en formato binario, conteniendo el documento junto con otra información requerida por el motor).

```
// Newly added document
{
    "name" : "Product",
    "description" : "This is a product",
    "price" : 1.45,
    "_id" : ObjectId("63754802232eff6d49d99ccf")
}
```

# In-Depth: GraphQL

GraphQL es excelente para queries. Realizar consultas a la API, especialmente cuando dichas consultas son complejas y tienen múltiples tipos de datos que recibir simultáneamente.

Hasta ahora, vimos que al utilizar APIs tipo REST, para indicar que queremos hacer una operación de escritura o actualización de datos, solamente teníamos que cambiar el tipo de método HTTP. Pero en GraphQL, el tipo de método al igual que la url que consultamos nunca cambian.

## Mutations

En GraphQL, para impactar cambios en nuestra API, se utiliza otro tipo de request GraphQL que se denomina **mutation**.

Las mutations van en el mismo body de la request de GraphQL, pero se aclara que es una mutación al principio de la misma.

Pueden ir junto a otras consultas, o pueden haber múltiples mutations. A diferencia de las queries, las mutations se ejecutan en serie. Mientras que las consultas son en paralelo.

```
mutation {
  changeEmail(email: "something@gmail.com") {
    email,
    age
  }
}
```

Las mutation son muy sencillas de utilizar, son prácticamente iguales que las queries. Reciben sus atributos al nombrar la mutation que queremos realizar, también están vinculadas a un resolver, y debemos seleccionar qué datos queremos obtener de la misma, tal como lo hacemos con las queries normales.

## Variables

GraphQL como dijimos, es una forma de hacer consultas hacia APIs muy potente. Es tan potente, que las variables son solamente una de las tantas cosas que nos permite utilizar dentro del lenguaje de GraphQL.

```
query($productId: String!) {
  product(id: $productId) {
    name,
    price,
    id
  }
}
```

## Fragmentos

Los fragmentos son, como su propio nombre lo describe, trozos de una query.

Supongamos que tenemos múltiples queries hacia una misma entidad y no queremos tener que repetir todo el tiempo las propiedades que estamos necesitando. O bien, tenemos una query que puede devolver dos tipos de datos diferentes y queremos algunas propiedades determinadas según cuál de ellos nos devuelva.

```
fragment productFields on Product {
  name,
  description,
  price
  id
}
```

```
query {
  products {
    ...productFields
  }
}
```

## Unions

Qué pasa si tengo un endpoint que devuelve más de un tipo de datos?

En este caso, puedo optar por unir dos o más tipos, de forma muy sencilla, aunque tiene sus trucos.

```
union Everything = Product | User
```

Si vamos a devolver una unión, es necesario hacer otros pasos más avanzados, ya que GraphQL necesita saber qué dato estamos devolviendo exactamente, tenemos que especificar esto, sumando la propiedad __typename a cada dato que devolvamos.

Esto además, se puede hacer con un resolver. Si bien hasta ahora vimos que podemos hacer resolvers unicamente para query y mutations, también es posible hacerlo para los type. Con la diferencia de que tenemos que devolver el nombre del tipo. Por ejemplo:

```
__resolveType(params) {
    if (params.price) {
        return 'Product';
    }
    if (params.email) {
        return 'User';
    }

    return null;
}
```

# Actividad

- Basado en la actividad de API REST, crear resolvers y mutations para reemplazar la arquitectura inicial por una arquitectura 100% para GraphQL