Klimadatastyrelsen

Brugervejledning – Transitionsguide for Fleksibel Opslagslogik

Drift og modernisering af Datafordeleren

September 2025

Version 0.5 – 09-09-2025

Indholdsfortegnelse

1 Indledning

Fleksibel opslagslogik er en ny GraphQL-tjeneste på Datafordeleren, der gør det muligt at udstille data gennem brug af joins i GraphQL. Da det er nødvendigt at have en grundlæggende forståelse af GraphQL for at anvende fleksibel opslagslogik, henvises der til siden med GraphQL-dokumentationen, som findes her: **GraphQL på Datafordeleren**.

1.1 Begreber

I nedenstående tabel er der en begrebsliste for guiden.

Begreb	Beskrivelse
Nuværende Datafordeler	
REST-tjenester	REST-tjenester, som omtales her, dækker specifikt over de nuværende REST-tjenester på Datafordeleren. Disse gør det muligt for anvendere at hente data via en URL. Bemærk, at betegnelsen ikke omfatter nyere REST-tjenester som eksempelvis dem, der findes til moderne fildownloads. REST-tjenester er beskrevet her: [REST-tjenester]
Moderniseret Datafordeler	
Entitetsbaserede GraphQL-tjenester	GraphQL-tjenester, der henter specifikke entiteter og ikke understøtter join-operationer mellem entiteter, idet der opretholdes strenge grænser mellem de forskellige entiteter.
Fleksibel opslagslogik	En viderebygning af de entitetsbaserede GraphQL-tjenester, der muliggør joins inden for og mellem registre, hvilket giver mulighed for mere komplekse datarelationer og forespørgsler.
Overentitet	En overentitet er den entitet, som en GraphQL query starter i.
Underentitet	En underentitet er den eller de entiteter som joines på en entitet. Underentiteter kan også have flere underentiteter.
API Client tool	Et program der bruges til at sende endepunkts-forespørgsler med.
GraphQL-klient	Et program der er bygget til at præsentere GraphQL-skemaer på en mere overskuelig måde og lave GraphQL requests.
flexible-tjeneste	Tjeneste for fleksibel opslagslogik der muliggør joins og bitemporal filtrering.
flexibleCurrent-tjeneste	Tjeneste for fleksibel opslagslogik der muliggør joins hvor registreringstid altid er sat som dags dato.
GraphQL-skema	En fil med en oversigt over alle mulige operationer for en entitet.

Tabel 1.1: Begrebsliste

2 GraphQL på Datafordeleren

Som tidligere nævnt, er du nødsaget til at have en forståelse for GraphQL, for at arbejde med fleksibel opslagslogik, henvises der til siden med GraphQL-dokumentationen. Dokumentationen kan findes her **GraphQL på Datafordeleren**.

3 Fleksibel opslagslogik

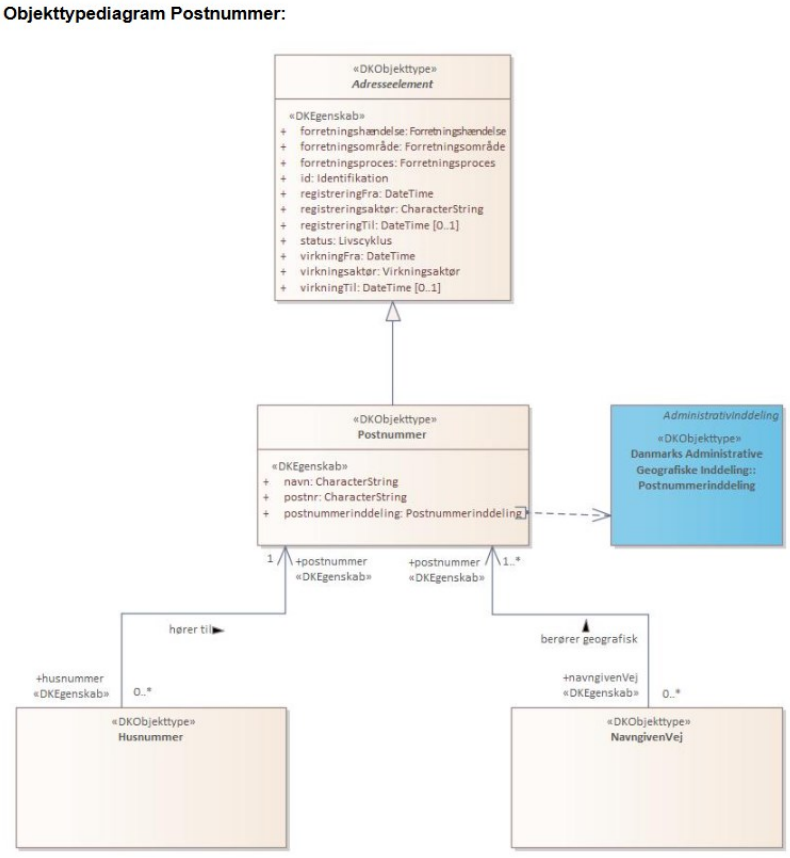
3.1 Hvad er fleksibel opslagslogik

Fleksibel opslagslogik er en viderebygning af de entitetsbaserede GraphQL-tjenester, der muliggør joins inden for og mellem registre, hvilket giver mulighed for mere komplekse datarelationer og forespørgsler. I fleksibel opslagslogik indarbejdes alle relationer, der er defineret i Grunddatamodellen. Grunddatamodellen giver et samlet overblik over samtlige registre og deres indbyrdes relationer, på tværs af alle tabeller. Det er vigtigt at nævne, at Grunddatamodel-dokumentionen, som er tilgængelig på Grunddatamodellen, viser den logiske datamodel hvori mod Datafordelerens tjenester udstiller den fysiske datamodel, som registrene har indsendt. Dette kan betyde, at der kan forekomme forskelle og dermed flere eller færre relationer i fleksibel opslagslogik. Med fleksibel opslagslogik bliver det muligt at slå op på alle disse relationer.

3.1.1 Forskellen mellem entitetsbaserede entiteter og fleksibel opslagslogik

Med de entitetsbaserede GraphQL-tjenester er det kun muligt at hente en entitet fra Grunddatamodellen ad gangen. Dette gør at du skal lave flere kald, hvis du har brug for data fra flere forskellige entiteter for så selv at sammensætte entiteterne. I stedet for at udføre flere kald for at få den nødvendige data, gør fleksibel opslagslogik det muligt at hente fra flere entiteter ad gangen, alt imens de bliver sammensat.

Med udgangspunkt i diagrammet på figur [3.1] vil det blive belyst, hvordan du kan bruge diagrammerne fra Grunddatamodellen til at finde disse relationer.



Figur 3.1: Grunddatamodel for DAR_Postnummer

I diagrammet er der en relation fra DAR_Postnummer til DAGI_Postnummerinddeling. I de entitetsbaserede GraphQL-tjenester ville du skulle udføre to separate forespørgsler for denne relation. Først ville du lave en forespørgsel (Se kode [3.2]) for at hente DAR_Postnummer, og derefter en ny forespørgsel (Se kode [3.3]) for at

hente DAGI_Postnummerinddelingen. Herefter skal resultaterne sammensættes for at skabe relationen mellem de to datasæt.

```
{
  DAR_Postnummer(
    first: 1
    registreringstid: "2025-11-14T11:28:27.184158Z"
    virkningstid: "2025-11-14T11:28:27.184158Z"
    where: { postnr: { eq: "0101" } }
  ) {
    pageInfo {
      hasNextPage
      endCursor
    }
    nodes {
      datafordelerOpdateringstid
      datafordelerRegisterImportSequenceNumber
      datafordelerRowId
      datafordelerRowVersion
      forretningshaendelse
      forretningsomraade
      forretningsproces
      id_lokalId
      id_namespace
      navn
      postnr
      postnummerinddeling
      registreringFra
      registreringsaktoer
      registreringTil
      status
      virkningFra
      virkningsaktoer
      virkningTil
    }
  }
}
```

Kode 3.2: DAR_Postnummer

```
{
  DAGI_Postnummerinddeling(
    first: 1
    registreringstid: "2025-11-14T11:28:27.184158Z"
    virkningstid: "2025-11-14T11:28:27.184158Z"
    where: { postnummer: { eq: "0101" } }
  ) {
    pageInfo {
      hasNextPage
      endCursor
    }
    nodes {
      datafordelerOpdateringstid
      datafordelerRegisterImportSequenceNumber
      datafordelerRowId
      datafordelerRowVersion
      erGadepostnummer
      forretningshaendelse
      forretningsproces
      id_lokalId
      id_namespace
      navn
      objectid
      postnummer
      registreringFra
      registreringsaktoer
      registreringTil
      virkningFra
      virkningsaktoer
      virkningTil
    }
  }
}
```

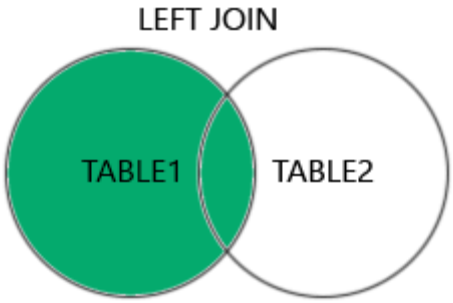
Kode 3.3: DAGI_Postnummerinddeling – Med fleksibel opslagslogik er det muligt at udføre én samlet forespørgsel, hvor både overentiteten og den relaterende underentitet hentes. Dette eliminerer behovet for flere separate queries og efterfølgende sammensætning. (Se eksempel på kode [3.4] nedenfor.)

```
{
  DAR_Postnummer(
    first: 1
    registreringstid: "2025-11-14T11:28:27.184158Z"
    virkningstid: "2025-11-14T11:28:27.184158Z"
    where: { postnr: { eq: "0101" } }
  ) {
    pageInfo {
      hasNextPage
      endCursor
    }
    nodes {
      datafordelerOpdateringstid
      datafordelerRegisterImportSequenceNumber
      datafordelerRowId
      datafordelerRowVersion
      forretningshaendelse
      forretningsomraade
      forretningsproces
      id_lokalId
      id_namespace
      navn
      postnr
      postnummerinddeling
      registreringFra
      registreringsaktoer
      registreringTil
      status
      virkningFra
      virkningsaktoer
      virkningTil
      PostnummerharPostnummerinddeling {
        datafordelerOpdateringstid
        datafordelerRegisterImportSequenceNumber
        datafordelerRowId
        datafordelerRowVersion
        erGadepostnummer
        forretningshaendelse
        forretningsproces
        id_lokalId
        id_namespace
        navn
        objectid
        postnummer
        registreringFra
        registreringsaktoer
        registreringTil
        virkningFra
        virkningsaktoer
        virkningTil
      }
    }
  }
}
```

Kode 3.4: DAR_Postnummer og DAGI_Postnummerinddeling relation

3.1.2 Left Joins i GraphQL

I fleksibel opslagslogik sammensættes forespørgslerne baseret på relationer mellem entiteter. Alle registre har indsendt deres relationsbilag, som definerer, hvilke relationer det er muligt at foretage i fleksibel opslagslogik. Disse relationer danner grundlaget for, hvordan queries konstrueres – nemlig ved hjælp af 'left joins'. Left joins fungerer på samme måde som i https://aws.amazon.com/what-is/sql/, dog med den forskel, at de værdier, der joines på, allerede er foruddefinerede i datamodellen. Det eneste, man som anvender af left joins bør være opmærksom på, er, at du maksimalt må joine op til 20 entiteter i alt og 8 entiteter i dybden (joins inde i joins) ad gangen.



Figur 3.5: Illustration af Left Joins. Kilde: https://www.w3schools.com/sql/sql_join_left.asp

På figur [3.5] vises det, hvordan left joins fungerer i SQL: Her sammensætter du to tabeller, så alle rækker fra den første tabel vises, og kun de matchende rækker fra den anden tabel inkluderes. I GraphQL fungerer det på samme måde, dog med den forskel, at den værdi, der joines på, som nævnt, er forudbestemt i GraphQL-skemaet og ikke angives direkte i forespørgslen.

Alle relevante join-værdier kan findes i GraphQL-skemaerne for fleksibel opslagslogik, som er tilgængelige på:

- ⇨ https://graphql.datafordeler.dk/<tjeneste>/v<version>/schema.

Et eksempel på en udfyldt URL kunne være:

- ⇨ https://graphql.datafordeler.dk/flexible/v1/schema

Her refererer ”flexible" til tjenesten med bitemporal filtrering, og "v1" angiver, at systemet på tidspunktet var i version 1.

Når du har downloadet det relevante GraphQL-skema, kan man finde relationerne på to måder. Én mulighed er at åbne GraphQL-skemafilen i et vilkårligt kode- eller tekstredigeringsprogram og manuelt gennemgå indholdet for at identificere de relevante relationer. Da anvenderen typisk ikke kender relationsnavnet på forhånd, kan det være nødvendigt at læse sig frem til den ønskede relation. Relationerne kan have sigende navne, men de kan også være autogenererede og derfor sværere at identificere. Det anbefales derfor at bruge et GraphQL-klient, som gør det nemmere at undersøge, hvilke relationer der findes, og hvor de peger hen. Det kan f.eks. være et værktøj som Altair eller Nitro. Et tredje alternativ vil være at downloade enten mermaid-filen eller visualiseringen af mermaid-filerne i den konverterede HTML-fil. I mermaid-filen kan man for hver entitet se deres relationer. I den konverterede HTML-fil for mermaid-fil kan man se et diagram over alle entiteter og deres relationer. Man skal zoome langt ind, før det er stort nok til, at man kan se indholdet.

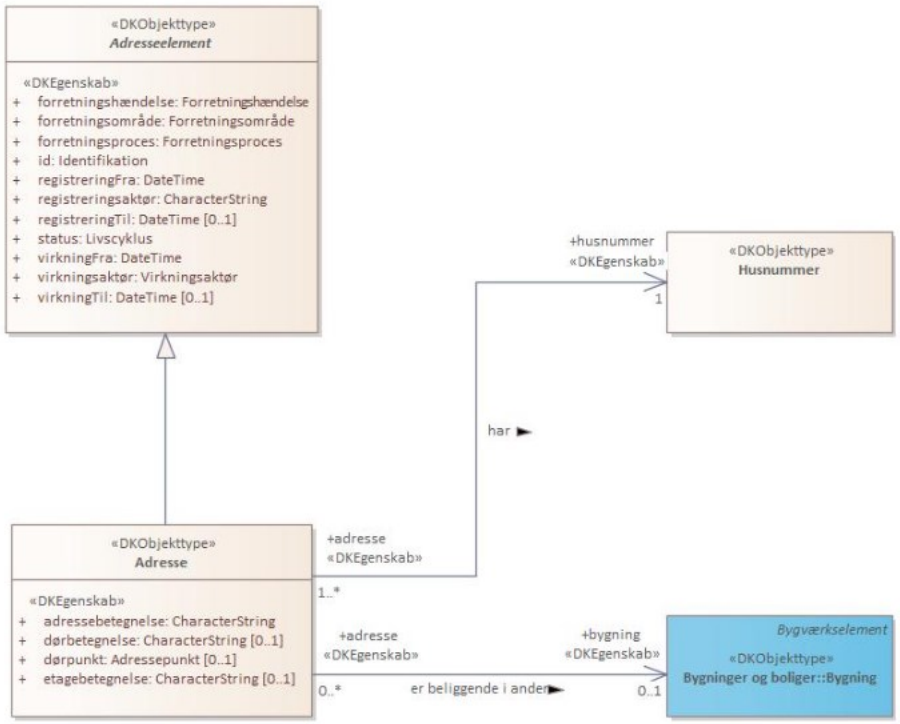
3.1.2.1 Eksempel med left join

I dette afsnit gennemgås et eksempel på, hvordan left joins fungerer i praksis.

3.1.2.1.1 Eksempel: Left join til underentitet fra samme register uden filtrering

I første eksempel er use-casen, at du har en specifik adresse, og du ønsker at finde det tilhørende husnummer.

Objekttypediagram Adresse:



Figur 3.6: Grunddatamodel for DAR_Adresse

I denne query starter du i entiteten DAR_Adresse. Som det fremgår af diagrammet fra Grunddatamodellen, kan man, hvis man ønsker husnummeret til en given adresse, gå direkte fra adressen til husnummeret.

```
{
  DAR_Adresse(
    first: 1
    registreringstid: "2025-01-23T10:39:04.658307Z"
    virkningstid: "2025-01-23T10:39:04.658307Z"
    where: { id_lokalId: { eq: "0000090e-e9f3-4ffe-a0a5-2852666d158c" } }
  ) {
    pageInfo {
      hasNextPage
      endCursor
    }
    nodes {
      adressebetegnelse
      bygning
      datafordelerOpdateringstid
      datafordelerRegisterImportSequenceNumber
      datafordelerRowId
      datafordelerRowVersion
      doerbetegnelse
      doerpunkt
      etagebetegnelse
      forretningshaendelse
      forretningsomraade
      forretningsproces
      husnummer
      id_lokalId
      id_namespace
      registreringFra
      registreringsaktoer
      registreringTil
      status
      virkningFra
      virkningsaktoer
      virkningTil
      adresseHarHusnummer {
        adgangsadressebetegnelse
        adgangspunkt
        adgangTilBygning
        adgangTilTekniskAnlaeg
        afstemningsomraade
        datafordelerOpdateringstid
        datafordelerRegisterImportSequenceNumber
        datafordelerRowId
        datafordelerRowVersion
        forretningshaendelse
        forretningsomraade
        forretningsproces
        geoDanmarkBygning
        husnummertekst
        id_lokalId
        id_namespace
        jordstykke
        kommuneinddeling
        menighedsraadsafstemningsomraade
        navngivenVej
        placeretPaaForeloebigtJordstykke
        postnummer
        registreringFra
        registreringsaktoer
        registreringTil
        sogneinddeling
        status
        supplerendeBynavn
        vejmidte
        vejpunkt
        virkningFra
        virkningsaktoer
        virkningTil
      }
    }
  }
}
```

Kode 3.7: DAR_Adresse og DAR_Husnummer relation

I GraphQL-skemaet for flexible-tjenesten, som findes på https://graphql.datafordeler.dk/flexible/v1/schema, kan du finde navnet på denne relation. I dette tilfælde hedder relationen adresseHarHusnummer, og derfor bruges det i ovenstående eksempel. Da relationen er defineret sådan, at husnummer fra DAR_Adresse matches med id_lokalId i DAR_Husnummer, returneres alle relevante resultater for hver adresse med matchende id_lokalId. Det er derfor vigtigt at filtrere i overentiteten (DAR_Adresse) for at begrænse resultatsættet – eksempelvis til en enkelt adresse. Du kan også angive adressebetegnelsen i filteret og få alle tilhørende husnumre for den pågældende adresse.

3.2 Begrænsninger for fleksibel opslagslogik

I fleksibel opslagslogik er der indført begrænsninger for at sikre optimal ydeevne og stabilitet af systemet. I denne sektion vil disse blive gennemgået. Disse begrænsninger adskiller blandt andet også GraphQL-queries fra klassiske SQL-queries.

3.2.1.1 Inner joins og unions understøttes ikke

Som den første begrænsning understøtter fleksibel opslagslogik ikke "inner join"-operationer, som ellers er mulige i SQL. Datafordeleren understøtter kun left joins, da det returnerer mere data ad gangen end inner joins, og du skal selv filtrere de returnerede resultater efter behov. I nogle tilfælde kan inner-joins dog håndteres ved at omstrukturere rækkefølgen af joins i en GraphQL-forespørgsel. Hvis der filtreres på en entitet med et inner-join, kan dette join i nogle tilfælde flyttes til toppen af GraphQL-forespørgslen og derved ændre på sammensætningen af joins fra REST-tjenesten. For et eksempel på dette se 3.6.1.1 i Brugervejledning - Udvalgte REST-tjenester omsat til GraphQL-queries.

En anden begrænsning er, at set-operatoren "union" fra SQL heller ikke understøttes i fleksibel opslagslogik. Dette skyldes det modsatte af begrundelsen for ikke at understøtte inner joins: Unions kan resultere i meget store datamængder, hvilket kan belaste serveren væsentligt og dermed påvirke alle anvenderne af endepunktet negativt. I situationer, hvor unions er relevante, anbefales det i stedet at udføre flere enkeltstående queries for hver enkelt entitet som ønskes. På den måde fordeles belastningen mere hensigtsmæssigt.

3.2.1.2 Begrænsning på antal left joins

Selvom left joins er understøttet i Datafordelerens GraphQL, er der en begrænsning på, hvor mange left joins du kan inkludere i en enkelt query. Denne er sat til 20 entiteter i alt og 8 entiteter i dybden (joins inde i joins). Det betyder, at visse REST-tjenester, som tidligere returnerede mange relaterede entiteter i én forespørgsel, ikke altid kan oversættes direkte til én GraphQL-query. I sådanne tilfælde skal forespørgslerne opdeles i flere separate queries.

Det anbefales derfor at overveje, om du reelt har behov for alle de underentiteter, som REST-tjenesten tidligere har returneret. Ofte kan der være data, som ikke er relevante for den konkrete brugssituation, og som med fordel kan frasorteres. Dette kan reducere både datamængde og behovet for efterfølgende filtrering og behandling lokalt.

3.2.1.3 Eksisterende relationer

I fleksibel opslagslogik er det på forhånd defineret, hvilke relationer der er mulige, og det er derfor ikke muligt for anvederen selv at oprette nye relationer. Hvis en ønsket relation ikke findes, kan man i stedet forsøge at opnå den ønskede sammenhæng ved at joine gennem andre entiteter. For eksempel er det ikke muligt at joine direkte fra DAR_Adresse til BBR_Etage; i stedet kan du opbygge en relation fra DAR_Adresse til BBR_Enhed og derfra videre til BBR_Etage.

3.2.2 Versionering

Det nye i fleksibel opslagslogik er, at alle registre nu er samlet i samme endepunkt. Det betyder, at hvis ét register – for eksempel DAR – får foretaget en ændring, skal alle anvendere, også dem der kun benytter andre registre såsom EJF, skifte til den nye version. Som tidligere gælder det, at der ikke vil blive foretaget breaking changes til en version,

der allerede er i drift. Hvis et register indberetter en manglende relation til Datafordeleren, og denne håndteres, træder ændringen først i kraft, når næste version af den eller de pågældende tjenester idriftsættes. Dette skyldes, at flexible- og flexibleCurrent-tjenesterne er fælles for alle registre, og derfor skal alle ændringer samles og implementeres samtidig i en ny version.

BBR Version	DAGI Version	CVR Version	Tjeneste version
v1	v1	v1	flexible/v1 flexibleCurrent/v1
v2	v1	v1	flexible/v2 flexibleCurrent/v2
v2	**v2**	v1	flexible/v3 flexibleCurrent/v3
v2	**v3**	**v2**	flexible/v4 flexibleCurrent/v4

Tabel 3.8: Eksempel på versioneringsændringer

I ovenstående eksempel starter tjenesten og alle relationsbilag på version v1.

- I række 2 opdaterer BBR til version v2, og hele tjenesten opdateres til v2.
- I række 3 opdaterer DAGI til version v2, og tjenesten opdateres til v3.
- I række 4 opdaterer DAGI til version v3 og CVR til version v2 samtidig, og systemet opdateres derfor kun én gang og forbliver på v4.

Til sidst vil systemet således have version v4. Det er op til KDS om hvorvidt tjenesterne opdateres ved ændret relationsbilag. Det er derfor ikke altid, at når der er ændringer, at de træder i kraft med det samme.

4 Opbygning af forespørgsler til fleksibel opslagslogik

4.1 URL-opbygning

I fleksibel opslagslogik er URL'er opbygget på to måder: Enten som en flexible-tjeneste, hvor der kan filtreres på bitemporalitet og alle registre kan tilgås, undtagen CVR, EJFCustom_PersonSimpelAlt og EJFCustom_PersonSimpelBegrænset, eller som en flexibleCurrent-tjeneste, hvor registreringstid er sat til dags dato.

Det anbefales, at du altid inkluderer en Accept-header med værdien application/graphql-response+json i deres forespørgsler. På den måde tilvælges moderne HTTP-statuskoder, så man får korrekte statuskoder tilbage fra serveren. Uden denne header returneres der altid statuskode 200, også hvis der opstår fejl.

4.1.1 Opsætning af endepunkter

For entitetsbaseret GraphQL er det nødvendigt at hente GraphQL-skemaer for hvert enkelt register for at få en forståelse for, hvordan registrenes data er opbygget. I fleksibel opslagslogik kan man hente et samlet GraphQL-skema, som giver et overblik over alle registres relationer – også på tværs af hinanden. I nedenstående eksempel kan man se, hvordan dette GraphQL-skema hentes ned via en GET request:

⇨ https://graphql.datafordeler.dk/{tjeneste}/{version}/schema?apikey={apikey}

Hvor {tjeneste} erstattes med enten flexibleCurrent eller flexible, {version} erstattes med den nuværende version, som i skrivende stund er v1 og {apikey} erstattes med egen API-nøgle. URL'en kan indtastes i en vilkårlig browser og filen vil automatisk blive downloadet. Den korrekte fil vil hedde "FLEX_CURRENT_V001.schema.graphql" eller "FLEX_V001.schema.graphql" alt afhængigt af version og tjeneste.

For så at lave specifikke GraphQL-kald vil URL'en næsten se ud på samme måde. Det eneste, der ikke medtages er "schema" parameteren.

⇨ https://graphql.datafordeler.dk/{tjeneste}/{version}?apikey={apikey}

Hvor parameterne igen udskiftes med den korrekte tjeneste, version og apikey. Alternativt kan autentificering opsættes ved hjælp af andre metoder, som forklares i i sektion [5]. Det er ikke et krav at anvende en API-nøgle. Dette gør sig gældende både for GraphQL-skema hentning og -forespørgsler.

4.2 GraphQL-skemaer for fleksibel opslagslogik

Tidligere afsnit har både forklaret GraphQL-skemaer og hvordan de hentes ned. Dette afsnit vil afklare hvordan disse bør læses og bruges.

4.2.1 Uden GraphQL-klienterer

Hvis du ikke ønsker at anvende GrapQL-værktøjer og i stedet vil udforme dine GraphQL-queries manuelt, kan GraphQL-skemaet åbnes i et vilkårligt tekstredigeringsprogram. Her skal du selv finde de ønskede relationer. I tilfælde af du ikke vælger at bruge et GraphQL-klienter, kan du bruge Grunddatamodellen for visualisering.

```
  "Field representing the result of a to-many left-join from DAR_Adresse.id_lokalId to BBR_Enhed.adresseIdentificerer with a standard point-in-time filter
applied."
  AdresseHarEnhed("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list that come after the specified cursor."
after: String where: BBR_EnhedFilterInput): BBR_EnhedConnection @cost(weight: "30000") @listSize(assumedSize: 1000, slicingArguments: [ "first" ],
sizedFields: [ "nodes", "edges" ], requireOneSlicingArgument: false) @entityJoin(joinKind: StandardBiTemporalLeftJoin)
```

Figur 4.1: AdresseHarEnhed join i GraphQL-skemaet

Ovenfor ses et eksempel på, hvordan et join vil fremgå i et GraphQL-skema. Her joines fra DAR_Adresse til BBR_Enhed, hvor relationen sker mellem DAR_Adresse's id_lokalId-værdi og BBR_Enhed's adresseIdentificerer-felt. For de fleste mulige joins vil der kunne findes et tilsvarende felt, der angiver join-betingelsen. Da GraphQL-skemaets tekst ofte er tæt og kompakt, kan det være vanskeligt at overskue eller finde de relevante felter. Et godt råd er at benytte 'CTRL + F' til at søge efter fra- eller til-entiteten i dokumentet.

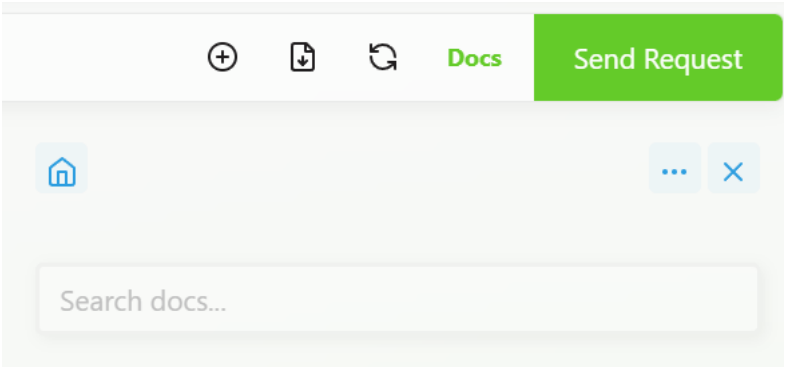
Som nævnt i afsnit [3.1.2], så kan du som alternativ hente mermaid-filen ned og den konverterede HTML-fil for mermaid-filen ned. Mermaid-filen giver et let læseligt overblik over hvilke relationer, der er understøttet for hver entitet og den konverterede HTML-fil giver et diagram overblik over alle entiteter og deres tilhørende relationer. Du skal som nævnt zoome langt ind i HTML-filen for at se indholdet, da det er meget småt.

4.2.2 Med GraphQL-klient

Det anbefales at benytte et GraphQL-klient til at læse GraphQL-skemaerne, da det gør arbejdet med GraphQL-tjenesterne enklere at læse. Med et GraphQL-klient som Altair eller Nitro bliver sammensætningen af queries gjort enklere.

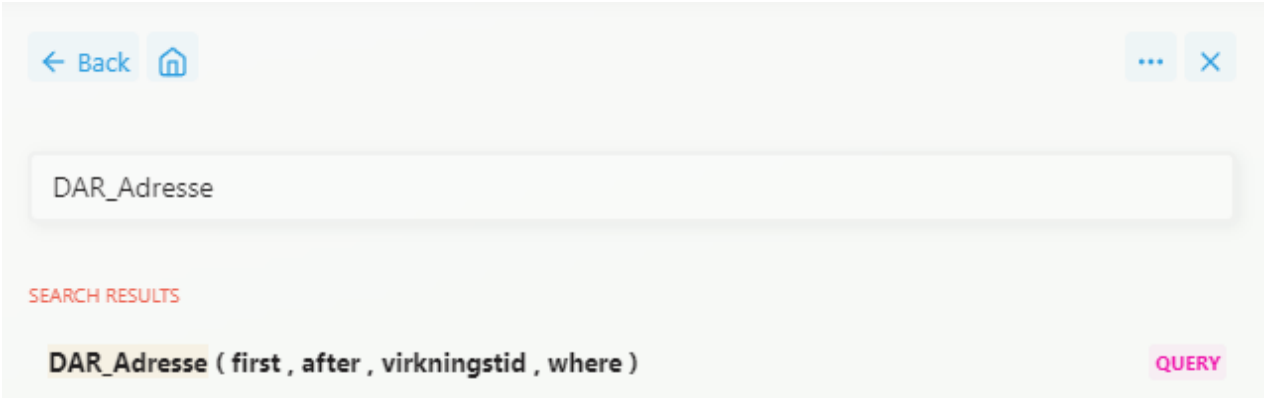
4.2.2.1 Eksempel på visning i GraphQL klient

Altair fungerer både som API-klient og GraphQL-klient. Med Altair kan du indlæse det hentede GraphQL-skema for den ønskede tjeneste ved at klikke på “Docs”, derefter på “...”-knappen og vælge den relevante fil. Når GraphQL-skemaet er åbnet, kan man frit søge i dokumentationen på en let læselig måde.



Figur 4.2: Altair dokumentation

I eksemplet på figur [4.1] ønskes det at finde en relation fra DAR_Adresse til BBR_Enhed. For at gøre dette i Altair søger du på "DAR_Adresse" i Docs-vinduet.



Figur 4.3: Altair søgning

Derefter åbner du entiteten med typen QUERY og klikker på nodes for at se alle felter og relationer, som DAR_Adresse indeholder. Her kan du se, at der findes en relation til BBR_Enhed ved navn AdresseHarEnhed.

Figur 4.4: ”ADD QUERY”-knappen i Altair

Alternativt kan du holde musen til højre for entitetsnavnet, hvor en “ADD QUERY”-knap vises. Ved at klikke på denne knap udfyldes alle felter automatisk, og der genereres en gyldig query. Det eneste, der herefter skal tilpasses, er valg af underentiteter, registrerings- og virkningstid samt paging.

4.3 Oversigt over registre

I nedenstående tabel er der en oversigt over hvilke registre der er tilgængelige i de forskellige tjenester samt om de er adgangsbegrænset. Denne oversigt er er god at bruge til beslutningen om hvilken tjeneste, du vil kalde, samt hvilke der kræver særtilladelser.

Register	flexible	flexibleCurrent	Adgangsbegrænset
BBR	**Ja**	**Ja**	**Nej**
CPR	**Ja (uden joins)**	**Ja (uden joins)**	**Ja**
CVR	**Nej**	Ja	**Nej (pånær CVRPerson**
DAGI	Ja	Ja	Nej
DAR	Ja	Ja	Nej
DHM Højdekurver	Ja	Ja	Nej
DHM Oprindelse	Ja	Ja	Nej
Danske Stednavne	Ja	Ja	Nej
EBR	Ja	Ja	Nej
EJF	Ja	Ja	**Ja**
EJFCustom	**Ja (uden PersonSimpel)**	Ja	**Ja**
Fikspunkter	Ja	Ja	Nej
GeoDanmark Vektor	Ja	Ja	Nej
MAT	Ja	Ja	Nej
SVR	Ja	Ja	**Ja**
VUR	Ja	Ja	Nej

Tabel 4.5: Oversigt over registre

5 Autentifikation

Alle der vil bruge fleksibel opslagslogik skal være authenticated. For opsætning af autentifikation til fleksibel opslagslogik se autentifikation afsnittet under **GraphQL på Datafordeleren**.

5.1 Specielt for fleksibel opslagslogik

Da nogle entiteter har begrænset adgang, er det nødvendigt at ansøge om adgang til disse. Du kan både ansøge om adgang og se, hvilke entiteter der er adgangsbegrænsede, under "Dataadgang" i IT-systemet i Datafordeler Administration. Her vælger du det ønskede register samt de specifikke entiteter, du ønsker adgang til. Følgende registre indeholder entiteter med adgangsbegrænsning: CPR, CVR, EJF, EJFCustom og SVR. Dette krav om adgangsgodkendelse gjaldt også i den entitetsbaserede opslagslogik. Forskellen er, at det nu er muligt at joine mellem både ubegrænsede og begrænsede entiteter, så længe du har den nødvendige adgang til samtlige relevante entiteter. Skal du tilgå disse registre, kan dette ikke gøres ved brug af en API-nøgle. Grundet øget sikkerhedskrav, skal der anvendes Shared Secret. Opsætningen af Shared Secret forklares også i **GraphQL på Datafordeleren**.

6 Bitemporalitet i fleksibel opslagslogik

Bitemporalitet i fleksibel opslagslogik fungerer næsten på samme måde som i entitetsbaseret GraphQL. Hvis du ønsker at læse mere om, hvordan bitemporalitet fungerer, henvises der til siden Datafordeleren - introduktion til bitemporalitet. I fleksibel opslagslogik er det ikke er muligt at anvende relationer til CVR, EJFCustom_PersonSimpelAlt og EJFCustom_PersonSimpelBegrænset i det almindelige flexible-miljø, da disse entiteter ikke understøtter filtrering på registreringstid pga. krav fra registrene. De kan derfor udelukkende tilgås via flexibleCurrent-tjenesten.

7 Paging

For fleksibel opslagslogik fungerer pagingpå præcis samme måde som i de entitetbaserede GraphQL-tjenester. Der henvises derfor til **GraphQL på Datafordeleren** for en dybdegående forklaring af paging. Dette afsnit vil kun tage udgangspunkt i, hvad der er nyt for fleksibel opslagslogik.

7.1 PageInfo for underentiteter

I **GraphQL på Datafordeleren** nævnes det, at du kan hente pageInfo på entiteter med én-til-mange-relationer. Det samme gælder for underentiteter. Det anbefales stærkt at benytte pageInfo både for over- og underentiteter, da det giver det bedst mulige overblik over data og paging.

Querien i figur [7.1] nedenunder er lavet uden filtrering (where-klausul) for at understrege vigtigheden af pageInfo. I querien hentes der 10 resultater fra overentiteten DAR_Adresse og kun 1 resultat fra underentiteten BBR_Enhed gennem én-til-mange joinet AdresseHarEnhed.

```
{
  DAR_Adresse(
    first: 10
    virkningstid: "2025-01-23T10:39:04.658307Z"
    where: {}
  ) {
    pageInfo {
      hasNextPage
      endCursor
    }
    nodes {
      adressebetegnelse
      bygning
      doerbetegnelse
      doerpunkt
      etagebetegnelse
      forretningshaendelse
      forretningsomraade
      forretningsproces
      husnummer
      id_lokalId
      id_namespace
      AdresseHarEnhed(first: 1)
      {
        pageInfo {
          hasNextPage
          endCursor
        }
        nodes {
          adresseIdentificerer
          bygning
          id_lokalId
          forretningsproces
          forretningsomraade
          forretningshaendelse
        }
      }
    }
  }
}
```

Kode 7.1: Én-til-mange relation med få resultater

Hvis du så forestiller dig, at man gerne vil have alle resultaterne ud, hvor der er mere end én enhed på en adresse, så giver denne query ikke nok resultater. Du vil i dit resultat kun få ét resultat pr. adresse, og det vil derfor ikke være det korrekte resultat, som du henter ud. I figur [7.2], kan man se, hvordan pageInfo belyser dette.


```
                {
                    "adressebetegnelse": "Spindelvej 11, Guderup, 6430 Nordborg",
                    "bygning": null,
                    "doerbetegnelse": "",
                    "doerpunkt": null,
                    "etagebetegnelse": "",
                    "forretningshaendelse": "2",
                    "forretningsomraade": "54.15.10.08",
                    "forretningsproces": "0",
                    "husnummer": "ba5a9150-df2c-48f6-84bd-47092e1b573e",
                    "id_lokalId": "00009b67-504c-4fe0-b1d1-39126722df0f",
                    "id_namespace": "http://data.gov.dk/dar/adresse",
                    "AdresseHarEnhed": {
                        "pageInfo": {
                            "hasNextPage": true,
                            "endCursor":
"TjJabE16RmhOR010WVRrMU1DMDBOemMxTFRoa09EUXRZVFZrTWpKa01qUTJaalUyO01qQXlNeTB3T1MweU0xUXdPVG95TlRvMU9T
NHlOamt3T0RJd0t6QXdPakF3O01qQXlNeTB3T1MweU0xUXdPVG95TlRvMU9TNHlOamt3T0RJd0t6QXdPakF3"
                        },
                        "nodes": [
                            {
                                "adresseIdentificerer": "00009b67-504c-4fe0-b1d1-39126722df0f",
                                "bygning": "2fa95df9-3f96-46ad-953f-39fe744fc2f3",
                                "id_lokalId": "7fe31a4c-a950-4775-8d84-a5d22d246f56",
                                "forretningsproces": "0",
                                "forretningsomraade": "54.15.05.05",
                                "forretningshaendelse": null
                            }
                        ]
                    }
                },
```


Kode 7.2: Resultat af query fra figur [7.1]

Fordi du bruger pageinfo på underentiteten, kan du se, at ”hasNextPage” er true. Det vil sige, at der findes flere resultater, end der er vist. Var pageInfo ikke blevet inkluderet i querien, så ville du ikke vide, at du manglede resultater.

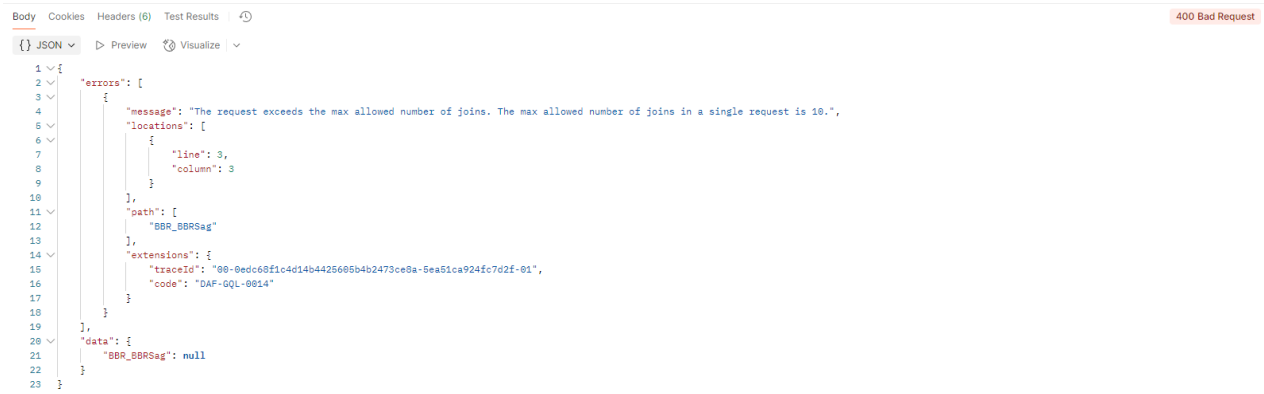
8 Fejlkoder

Dette afsnit vil gennemgå de fejlkoder, en anvender af fleksibel opslagslogik vil kunne støde på og dermed skabe en forståelse for hvilke type fejl, du kan møde samt hvordan disse løses.

Det bør nævnes, at hvis du som anvender ikke har lavet opt-in på moderne HTTP koder i din API- eller GraphQL-klient eller din browser, vil du ikke altid modtage en fejlkode på din forespørgsel. Du vil i stedet modtage HTTP-kode OK-200. Korrekte fejlkoder er vigtige, da det er sådan systemer bygget på fleksibel opslagslogik vil kunne flage, at der er en fejl.

8.1 Maksimalt antal joins overskredet

En ny type fejl anvendere kan møde er 400 Bad Request med beskrivelsen: ”The request exceeds the max allowed number of joins. The max allowed number of joins in a single request is 20.” (Se eksempel nedenunder)



Figur 8.1: Fejlkode for maksimalt antal joins overskredet

Denne fejl opstår, fordi der som nævnt – på trods af muligheden for at lave joins og sammensætte data i fleksibel opslagslogik – er indført en begrænsning for at undgå overbelastning af systemet og sikre god ydeevne. Som tidligere nævnt er grænsen sat til 20 joins i alt eller 8 joins i dybden. Hvis du modtager denne fejl, skyldes det, at man har overskredet det tilladte antal joins i en enkelt request. Der er to løsninger på dette problem:

1. Du kan konsekvent fjerne unødvendige joins.
2. Du kan opdele forespørgslen i flere mindre forespørgsler.

Det bør altid vurderes for hver request om anvenderne har brug for al dataen og om nogle dele kan udelades.

8.2 Anvender har ikke adgang til denne ressource

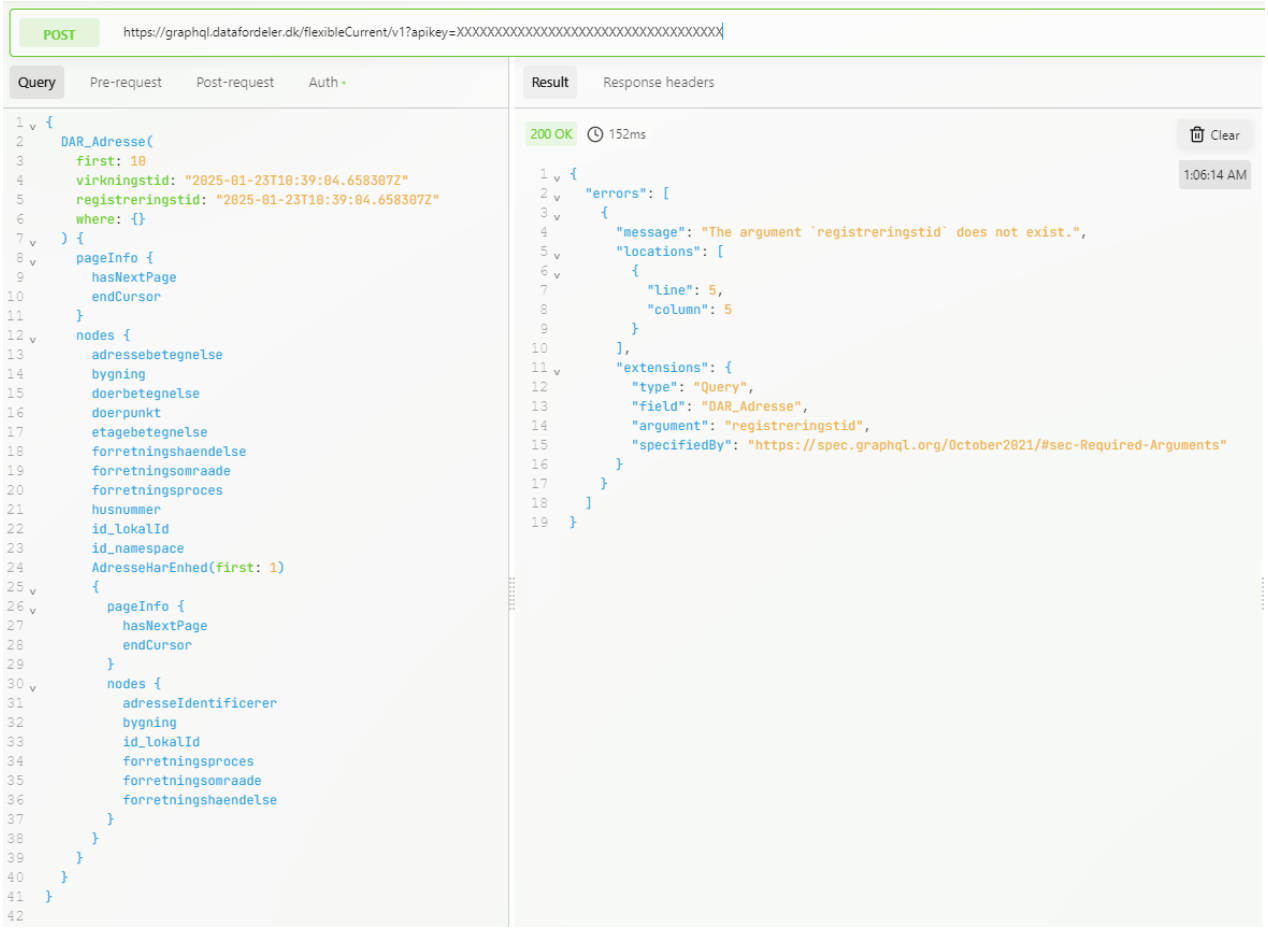
I entitetsbaseret GraphQL kan du kun tilgå en entitet ad gangen, hvilket gjorde det tydeligt, om du hvade adgang til en given entitet eller ej. Med fleksibel opslagslogik er det nu muligt at joine til entiteter, som du ikke nødvendigvis har adgang til. Hvis dette sker, vil en query ikke altid resultere i en 401 Unauthorized-fejl, men kan i stedet returnere en 200 OK-status med beskeden: "The current user is not authorized to access this resource." Se eksempel nedenfor:

Figur 8.2: Fejlkode for ingen adgang til ressource

Den returnerer kun HTTP kode OK-200, hvis ikke man har lavet opt-in til moderne HTTP koder.

8.3 Specielt for flexibleCurrent tjenesten

I entitetsbaseret GraphQL har du tidligere kunnet foretage forespørgsler direkte mod hvert registers egne endepunkter. I fleksibel opslagslogik skal man være opmærksom på forskellen mellem flexible- og flexibleCurrent-tjenesterne. I flexible-tjenesten skal der filtreres både på registreringstid og virkningstid. Forsøger du at anvende queries fra flexible-tjenesten direkte på flexibleCurrent-tjenesten, vil du få fejlen: "The argument 'registreringstid' does not exist". (Se figur [8.3]).

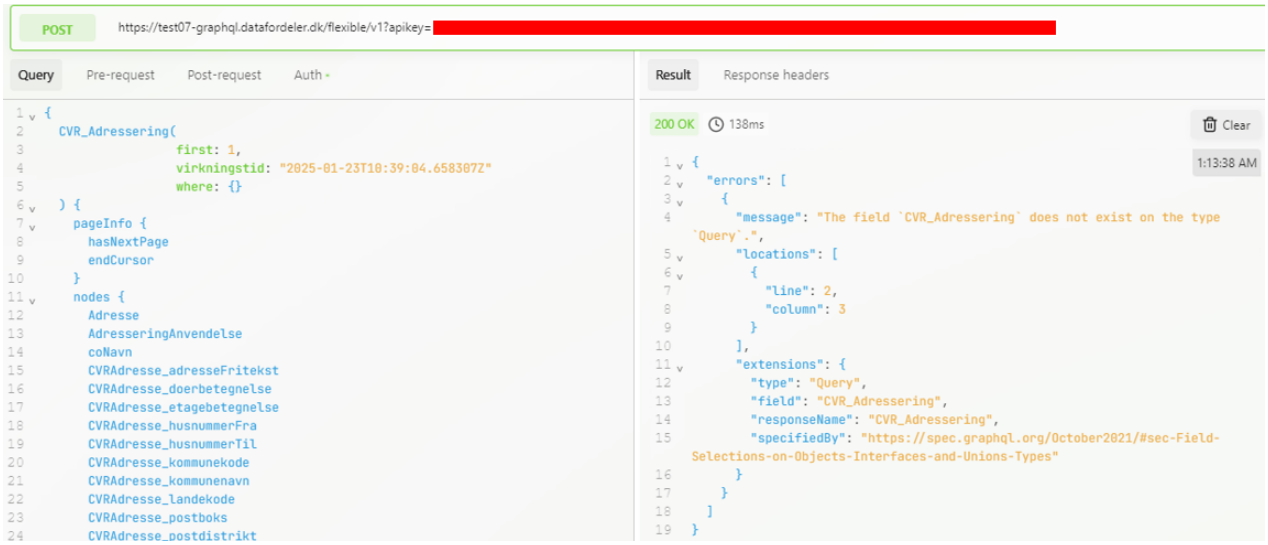


Figur 8.3: Registreringstid findes ikke

I eksemplet forsøges der at blive kaldt en query for DAR_Adresse, på flexibleCurrent-tjenesten med registreringstid udfyldt, da man vil joine til en CVR entitet. Da parameteren registreringstid ikke findes i flexibleCurrent, svarer serveren med HTTP-status 400 Bad Request. Løsningen på dette problem er blot at fjerne parameteren registreringstid fra forespørgslen.

8.4 Specielt for flexible tjenesten

I flexible-tjenesten er det ikke muligt at forespørge på alle CVR-entiteter samt EJFCustom_PersonSimpelAlt og EJFCustom_PersonSimpelBegrænset. Dette er tidligere nævnt, men dette afsnit beskriver, hvilken fejlmeddelelse du vil få i sådanne tilfælde. I eksemplet nedenfor forsøges det at kalde en query for CVR på flexible-endepunktet i stedet for flexibleCurrent:



Figur 8.4: Entitet findes ikke

Her vil man modtage en HTTP 400 Bad Request med beskeden: "The field '{entitet}' does not exist on the type 'Query'." Som løsning bør du altid først kontrollere, om entitetens navn er stavet korrekt, da entitetsnavne er case sensitive og selv små stavefejl kan forårsage fejl. Hvis du er sikker på, at queryen er korrekt skrevet, bør du derefter sikre sig, at du kalder den rette tjeneste. Er du sikker på, at du kalder den korrekte tjeneste, bør du undersøge om, at dit kald er tilladt. Dette kan du finde ud af i den givne tjenestes GraphQL-skema.

9 Transition og eksempel på anvendelse

Denne sektion går i dybden med, hvordan du som anvender kan overgå til fleksibel opslagslogik, og indeholder et eksempel på, hvordan en REST-tjeneste kan oversættes til en GraphQL-forespørgsel, der viser anvendelsen af fleksibel opslagslogik i praksis. Eksemplet kommer fra et dokument på Confluence, som Datafordeleren har lavet, for at vise, hvordan de top 10 mest anvendte REST-tjenester kan oversættes til GraphQL tjenester. Resten af oversættelserne kan findes i Brugervejledning - Udvalgte REST-tjenester omsat til GraphQL-queries.

9.1 Overgangen fra REST-tjenester til GraphQL-tjenester

Med entitetsbaseret GraphQL har det hidtil været vanskeligt at hente større mængder sammenhængende data ad gangen. Her har du som anvender skulle lave flere forskellige kald, for at få samme mængde data som i REST-tjenesterne. Med indførelsen af fleksibel opslagslogik bliver det nu muligt at oversætte REST-tjenester direkte til GraphQL. For nogle REST-kald, der anvender mere end 8 joins i dybden eller 20 joins i alt for at sammensætte data, vil det dog være nødvendigt at opdele forespørgslerne i flere GraphQL-queries. En væsentlig fordel ved GraphQL er, at du nu selv kan vælge præcis, hvilke felter og relationer, der skal returneres. Hvor REST-tjenester returnerer alle, kan du i GraphQL frit fravælge uønskede felter og relationer. Det betyder, at man kun henter den data, man reelt har brug for, hvilket kan minimere behovet for efterfølgende datarensning og øge effektiviteten i dataudtrækket.

Det er vigtigt at nævne, at planen er, at GraphQL-tjenesterne skal erstatte REST-tjenesterne, og at REST-tjenesterne vil blive udfaset og gjort utilgængelige i fremtiden. Planen for dette vil være, at der vil være paralleldrift med både REST og GraphQL-tjenester, hvorefter REST-tjenesten vil blive lukket. Du bør derfor allerede nu overveje at oversætte de tjenester, du benytter, til GraphQL. Hvis oversættelsen ikke sker i tide, vil dune risikere at miste adgang til de tjenester, du er afhængig af, hvilket kan få konsekvenser for systemer, der baserer sig på disse data.

For at gøre overgangen til GraphQL så enkel som mulig, anbefales det kraftigt, at dune anvender et GraphQL-klient. I afsnittet [4.2] om GraphQL-skemaer gennemgås, hvordan GraphQL-skemaerne læses, og hvorfor det er en fordel at benytte et GraphQL-klient.

9.2 Eksempel på oversættelse af REST-tjeneste til GraphQL-query

I dette afsnit gennemgås et eksempel på, hvordan en nuværende REST-tjeneste kan oversættes til GraphQL-queries. Formålet med eksemplet er at vise, hvordan du som anvender kan konvertere REST-tjenester til GraphQL-queries og dermed få inspiration til at udforme egne queries. Det er også vigtigt at fremhæve, at du ikke længere er bundet af REST-tjenesternes struktur. Så længe relationerne i relationsbilagene tillader det, kan du nu sammensætte queries, der præcist matcher dine behov. Det åbner mulighed for at opbygge 'egne' tjenester, hvilket ikke tidligere var muligt.

Alle oversættelserne i Brugervejledning - Udvalgte REST-tjenester omsat til GraphQL-queries er lavet baseret på pseudo-SQL'en fra Dataleverancespecifikationen (DLS). SQL'en kan findes på DLS-udstilling i ZIP-filerne. Vil du oversætte en REST-tjeneste, som du anvender, kan du bruge SQL'en som udgangspunkt og oversætte så vidt muligt.

9.2.1 Eksempel: Oversættelse af REST-Tjenesten BBRSag

I figur [10.1] oversættes REST-tjenesten for BBRSag. For tjenesten BBRSag ser pseudo-SQL'en for tjenesten således ud:

```
SELECT BBRSagSource.*, SagsniveauSource.* FROM BBRSag AS BBRSagSource
LEFT JOIN Sagsniveau AS SagsniveauSource ON SagsniveauSource.byggesag = BBRSagSource.id_lokalId
WHERE (@Id = NULL OR BBRSagSource.id_lokalId In @Id)
AND (@VirkningFra = NULL OR BBRSagSource.virkningTil = NULL OR @VirkningFra <
BBRSagSource.virkningTil)
AND (@VirkningTil = NULL OR BBRSagSource.virkningFra = NULL OR @VirkningTil >=
BBRSagSource.virkningFra)
AND (@Virkningsaktoer = NULL OR BBRSagSource.virkningsaktoer = @Virkningsaktoer)
AND (@RegistreringFra = NULL OR BBRSagSource.registreringTil = NULL OR @RegistreringFra <
BBRSagSource.registreringTil)
AND (@RegistreringTil = NULL OR BBRSagSource.registreringFra = NULL OR @RegistreringTil >=
BBRSagSource.registreringFra)
AND (@Registreringsaktoer = NULL OR BBRSagSource.registreringsaktoer = @Registreringsaktoer)
AND (@Status = NULL OR BBRSagSource.status IN @Status)
AND (@Forretningsproces = NULL OR BBRSagSource.forretningsproces = @Forretningsproces)
AND (@Forretningsomraade = NULL OR BBRSagSource.forretningsomraade = @Forretningsomraade)
AND (@Forretningshaendelse = NULL OR BBRSagSource.forretningshaendelse = @Forretningshaendelse)
AND (@Kommunekode = NULL OR BBRSagSource.Kommunekode = @Kommunekode)
AND (@DAFTimestampFra = NULL OR @DAFTimestampFra <= BBRSagSource.UpdateTimestamp())
AND (@DAFTimestampTil = NULL OR @DAFTimestampTil > BBRSagSource.UpdateTimestamp())
AND (@Bygning = NULL OR SagsniveauSource.stamdataBygning = @Bygning OR
SagsniveauSource.sagsdataBygning = @Bygning)
AND (@Enhed = NULL OR SagsniveauSource.stamdataEnhed = @Enhed OR SagsniveauSource.sagsdataEnhed =
@Enhed)
AND (@Etage = NULL OR SagsniveauSource.stamdataEtage = @Etage OR SagsniveauSource.sagsdataEtage =
@Etage)
AND (@Grund = NULL OR SagsniveauSource.stamdataGrund = @Grund OR SagsniveauSource.sagsdataGrund =
@Grund)
AND (@Opgang = NULL OR SagsniveauSource.stamdataOpgang = @Opgang OR SagsniveauSource.sagsdataOpgang =
@Opgang)
AND (@TekniskAnlaeg = NULL OR SagsniveauSource.stamdataTekniskAnlaeg = @TekniskAnlaeg OR
SagsniveauSource.sagsdataTekniskAnlaeg = @TekniskAnlaeg)
AND (
    @PeriodeaendringFra = NULL
    OR
    @PeriodeaendringTil = NULL
    OR (
    BBRSagSource.Id_lokalId IN
    (
        SELECT DISTINCT BBRSagChanges.Id_lokalId FROM BBRSag AS BBRSagChanges
        WHERE BBRSagChanges.Id_lokalId IN
        (
            SELECT DISTINCT BS.Id_lokalId FROM BBRSag AS BS
            WHERE
                (BS.registreringFra >= @PeriodeaendringFra AND BS.registreringFra <=
@PeriodeaendringTil)
                OR
                (BS.registreringTil >= @PeriodeaendringFra AND BS.registreringTil <=
@PeriodeaendringTil)
        )
        UNION
        SELECT DISTINCT SagsniveauChanges.byggesag FROM Sagsniveau AS SagsniveauChanges
        WHERE SagsniveauChanges.Id_lokalId IN
        (
            SELECT DISTINCT SN.id_lokalId FROM Sagsniveau AS SN
            WHERE
                (SN.registreringFra >= @PeriodeaendringFra AND SN.registreringFra <=
@PeriodeaendringTil)
                OR
                (SN.registreringTil >= @PeriodeaendringFra AND SN.registreringTil <=
@PeriodeaendringTil)
        )
    )
    AND BBRSagSource.virkningTil = NULL
```

```
    AND
    (
        (
            @KunNyesteIPeriode = FALSE
            AND
            BBRSagSource.registreringFra <= @PeriodeaendringFra AND (BBRSagSource.registreringTil >
@PeriodeaendringFra OR BBRSagSource.registreringTil = NULL)
            AND
            (
                SagsniveauSource.id_lokalId = NULL OR
                (
                    SagsniveauSource.virkningTil = NULL AND
                    SagsniveauSource.registreringFra <= @PeriodeaendringFra AND
(SagsniveauSource.registreringTil > @PeriodeaendringFra OR SagsniveauSource.registreringTil = NULL)
                )
            )
        )
        OR
        (
            BBRSagSource.registreringFra <= @PeriodeaendringTil AND (BBRSagSource.registreringTil >
@PeriodeaendringTil OR BBRSagSource.registreringTil = NULL)
            AND
            (
                SagsniveauSource.id_lokalId = NULL OR
                (
                    SagsniveauSource.virkningTil = NULL AND
                    SagsniveauSource.registreringFra <= @PeriodeaendringTil AND
(SagsniveauSource.registreringTil > @PeriodeaendringTil OR SagsniveauSource.registreringTil = NULL)
                )
            )
        )
    )
))
```

Kode 10.1: Pseudo-SQL for REST-tjenesten BBRSag

I pseudo SQL-querien henter man alt fra tabellerne BBRSag og BBRSagsniveau og filtrerer på flere parametre. For simplificering er næsten alle disse parametre udeladt og de bitemporale felter håndteres af registreringstid og virkningstid parametrene i GraphQL.

Med entitetsbaseret GraphQL var du nødt til at lave to separate GraphQL-kald for at oversætte denne tjeneste. Et kald for at få BBR_BBRSag entiteten og et kald for at få BBR_Sagsniveau. Så skulle du selv manuelt sammensætte dataen i et system. Med fleksibel opslagslogik er det muligt at joine direkte på sin entitet.

For at lave dette join skal vi først i GraphQL-skemaet finde ud af, hvad relationen hedder. I dette eksempel har relationen et sigende navn, og den var derfor nem at finde. Relationen ser således ud i GraphQL-skemaet:

```
  "Field representing the result of a to-many left-join from BBR_BBRSag.id_lokalId to BBR_Sagsniveau.byggesag with a standard point-in-time filter applied."
  bbrSagErTilknyttetSagsniveau("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list that come after the specified cursor." after: String where: BBR_SagsniveauFilterInput): BBR_SagsniveauConnection
@cost(weight: "30000") @listSize(assumedSize: 1000, slicingArguments: [ "first" ], sizedFields: [ "nodes", "edges" ], requireOneSlicingArgument: false) @entityJoin(joinKind: StandardBiTemporalLeftJoin)
}
```

Bruger man i stedet et GraphQL-klient vil det fremstå under BBR_BBRSag entiteten. Da BBR entiteten indgår i flexible tjenesten kalder vi vores request fra følgende URL:

https://graphql.datafordeler.dk/flexible/v1?apikey={apikey}

Nu hvor relationen kendes, kan GraphQL-querien opbygges.

```
{
   BBR_BBRSag(
     first: 10
     registreringstid: "2025-03-28T22:12:56.184158Z"
     virkningstid: "2025-11-14T11:28:27.184158Z"
     where: {
       status: { in: ["11", "12"] }
       kommunekode: { eq: "0101" }
     }
   )
```

Kode 10.2: Første del af BBRSag i GraphQL

Queries er opbygget sådan, at du tager udgangspunkt i overentiteten BBR_BBRSag. Du anvender "first"-parameteren for at begrænse søgningen til de første 10 resultater. Da du kalder flexible-tjenesten skal du benytte de bitemporale felter registreringstid og virkningstid til at filtrere data. Derudover tilføjes valgfrie klausuler, hvor du i dette tilfælde ønsker at hente alle resultater med statuskode 11 eller 12 med kommunekode 0101.

```
{
     pageInfo {
       hasNextPage
       endCursor
     }
```

Kode 10.3: Paging info

I næste del hentes paging informationer, hvilket gør det lettere for anvenderen at holde overblik over de hentede data. Da "first"-parameteren er sat til kun at vise 10 resultater, kan der være yderligere data på næste side, som ikke vises i det aktuelle udtræk. Det anbefales altid at medtage paginginformationer i sine queries.

```
     nodes {
       datafordelerOpdateringstid
       datafordelerRegisterImportSequenceNumber
       datafordelerRowId
       datafordelerRowVersion
       forretningshaendelse
       forretningsomraade
       forretningsproces
       id_lokalId
       id_namespace
       kommunekode
       registreringFra
       registreringsaktoer
       registreringTil
       sag001Byggesagsnummer
       sag002Byggesagsdato
       sag003Byggetilladelsesdato
       sag004ForventetPaabegyndelsesdato
       sag005Paabegyndelsesdato
       sag006IbrugtagningsTilladelse
       sag007Henlaeggelse
       sag008FaerdigtBygningsareal
       sag009ForventetFuldfoertDato
       sag010FuldfoerelseAfByggeri
       sag012Byggesagskode
       sag013AnmeldelseAfByggearbejde
       sag016DelvisIbrugtagningsTilladelse
       sag017ForeloebigFaerdiggjortBygningsareal
       sag018ForeloebigFaerdiggjortAntalLejligheder
       sag019Bygherreforhold
```

```
        sag024DatoForModtagelseAfAnsoegningOmByggetilladelse
        sag025DatoForFyldestgoerendeAnsoegning
        sag026DatoForNaboorientering
        sag027DatoForFaerdigbehandlingAfNaboorientering
        sag032Litra
        sag033ForeloebigFaerdiggjortAntalLejlighederUdenKoekken
        status
        virkningFra
        virkningsaktoer
        virkningTil
        bbrSagErTilknyttetSagsniveau(
          first: 10
        ) {
          pageInfo {
            hasNextPage
            endCursor
          }
          nodes {
            datafordelerOpdateringstid
            byggesag
            forretningshaendelse
            forretningsomraade
            forretningsproces
            id_lokalId
            id_namespace
            kommunekode
            niveautype
            registreringFra
            registreringsaktoer
            registreringTil
            sagsdataBygning
            sagsdataEnhed
            sagsdataEtage
            sagsdataGrund
            sagsdataOpgang
            sagsdataTekniskAnlaeg
            sagstype
            stamdataBygning
            stamdataEnhed
            stamdataEtage
            stamdataGrund
            stamdataOpgang
            stamdataTekniskAnlaeg
            status
            virkningFra
            virkningsaktoer
            virkningTil
          }
        }
```

Kode 10.4: Nodes til BBRSag i GraphQL

Så henter du alle nodes ind. Nodes er der, hvor der bør udfyldes, hvilke felter du ønsker returneret som svar. I pesudo-SQL'en er der "Select *" og du ønsker derfor alle felter ud. Som noget nyt i fleksibel opslagslogik kan du som nævnt lave joins og det vil ske i nodes. Her har vi joinet bbrSagErTilknyttetSagsniveau. Opstillingen af underentiteten vil være på præcis samme måde som opstillingen af en overentitet med undtagelse af, at du ikke kan indstille registreringstid eller virkningstid, da disse arves fra overentiteten. Den samlede query vil så ende med at se således ud:

```
{
   BBR_BBRSag(
     first: 10
     registreringstid: "2025-03-28T22:12:56.184158Z"
     virkningstid: "2025-11-14T11:28:27.184158Z"
     where: {
       status: { in: ["11", "12"] }
       kommunekode: { eq: "0101" }
     }
   ) {
     pageInfo {
       hasNextPage
       endCursor
     }
     nodes {
       datafordelerOpdateringstid
       datafordelerRegisterImportSequenceNumber
       datafordelerRowId
       datafordelerRowVersion
       forretningshaendelse
       forretningsomraade
       forretningsproces
       id_lokalId
       id_namespace
       kommunekode
       registreringFra
       registreringsaktoer
       registreringTil
       sag001Byggesagsnummer
       sag002Byggesagsdato
       sag003Byggetilladelsesdato
       sag004ForventetPaabegyndelsesdato
       sag005Paabegyndelsesdato
       sag006IbrugtagningsTilladelse
       sag007Henlaeggelse
       sag008FaerdigtBygningsareal
       sag009ForventetFuldfoertDato
       sag010FuldfoerelseAfByggeri
       sag012Byggesagskode
       sag013AnmeldelseAfByggearbejde
       sag016DelvisIbrugtagningsTilladelse
       sag017ForeloebigFaerdiggjortBygningsareal
       sag018ForeloebigFaerdiggjortAntalLejligheder
       sag019Bygherreforhold
       sag024DatoForModtagelseAfAnsoegningOmByggetilladelse
       sag025DatoForFyldestgoerendeAnsoegning
       sag026DatoForNaboorientering
       sag027DatoForFaerdigbehandlingAfNaboorientering
       sag032Litra
       sag033ForeloebigFaerdiggjortAntalLejlighederUdenKoekken
       status
       virkningFra
       virkningsaktoer
       virkningTil
       bbrSagErTilknyttetSagsniveau(
         first: 10
       ) {
         pageInfo {
           hasNextPage
           endCursor
         }
         nodes {
           datafordelerOpdateringstid
           byggesag
           forretningshaendelse
           forretningsomraade
           forretningsproces
           id_lokalId
           id_namespace
           kommunekode
```

```
            niveautype
            registreringFra
            registreringsaktoer
            registreringTil
            sagsdataBygning
            sagsdataEnhed
            sagsdataEtage
            sagsdataGrund
            sagsdataOpgang
            sagsdataTekniskAnlaeg
            sagstype
            stamdataBygning
            stamdataEnhed
            stamdataEtage
            stamdataGrund
            stamdataOpgang
            stamdataTekniskAnlaeg
            status
            virkningFra
            virkningsaktoer
            virkningTil
          }
        }
      }
    }
  }
```

Kode 10.5: Fulde query for BBRSag

Svaret på querien vil så have medtaget alle felter fra querien. Hvis underentiteten er tom, er det fordi, at der ikke er nogen data på denne. Ønsker du at kalde denne i flexibleCurrent, skal med fjerne registreringstids parameteren.


```
{
    "data": {
        "BBR_BBRSag": {
            "pageInfo": {
                "hasNextPage": true,
                "endCursor":
"TURBd05qVXlaVEl0WVRBNU9TMDBZbVUyTFdFellqWXRaV1pqTkRRM05XRXpNREZpO01qQXhPQzB3TWkweE5sUXhOem8xT0Rvd01D
NHdNREF4TVRrd0t6QXdPakF3O01qQXhOaTB4TWkwd01WUXhNVG95TXpveU15NDFNVGN3TURBd0t6QXdPakF3"
            },
            "nodes": [
                {
                    "datafordelerOpdateringstid": "2021-04-04T05:39:13.568817Z",
                    "datafordelerRegisterImportSequenceNumber": 1857504,
                    "datafordelerRowId": "5d7438ad-5415-4764-9602-eaab858134a6",
                    "datafordelerRowVersion": 1,
                    "forretningshaendelse": "Byggesag",
                    "forretningsomraade": "54.15.05.05",
                    "forretningsproces": "11",
                    "id_lokalId": "000652e2-a099-4be6-a3b6-efc4475a301b",
                    "id_namespace": "http://data.gov.dk/bbr/bbrsag",
                    "kommunekode": "0101",
                    "registreringFra": "2018-02-16T17:58:00.000119Z",
                    "registreringsaktoer": "BBR",
                    "registreringTil": null,
                    "sag001Byggesagsnummer": "443870",
                    "sag002Byggesagsdato": null,
                    "sag003Byggetilladelsesdato": null,
                    "sag004ForventetPaabegyndelsesdato": null,
                    "sag005Paabegyndelsesdato": null,
                    "sag006IbrugtagningsTilladelse": null,
                    "sag007Henlaeggelse": null,
                    "sag008FaerdigtBygningsareal": null,
                    "sag009ForventetFuldfoertDato": null,
                    "sag010FuldfoerelseAfByggeri": null,
                    "sag012Byggesagskode": "3",
                    "sag013AnmeldelseAfByggearbejde": null,
                    "sag016DelvisIbrugtagningsTilladelse": null,
                    "sag017ForeloebigFaerdiggjortBygningsareal": null,
```

```
                    "sag018ForeloebigFaerdiggjortAntalLejligheder": null,
                    "sag019Bygherreforhold": null,
                    "sag024DatoForModtagelseAfAnsoegningOmByggetilladelse": "2011-01-
13T23:00:00.000000Z",
                    "sag025DatoForFyldestgoerendeAnsoegning": null,
                    "sag026DatoForNaboorientering": null,
                    "sag027DatoForFaerdigbehandlingAfNaboorientering": null,
                    "sag032Litra": "",
                    "sag033ForeloebigFaerdiggjortAntalLejlighederUdenKoekken": null,
                    "status": "11",
                    "virkningFra": "2016-12-01T11:23:23.517000Z",
                    "virkningsaktoer": "Konvertering2017",
                    "virkningTil": null,
                    "bbrSagErTilknyttetSagsniveau": {
                        "pageInfo": {
                            "hasNextPage": false,
                            "endCursor":
"TWpRM01qY3dZamd0TldVMFpDMDBaRE0yTFdJMVkyWXRNVGhqWm1KbU56TTFZelZoO01qQXhOeTB3Tmkwd01sUXhNam95T1Rvek55
NHdNREF3TURBd0t6QXdPakF3O01qQXhOaTB4TWkwd01WUXhNVG95TXpveU15NDFNVGN3TURBd0t6QXdPakF3"
                        },
                        "nodes": [
                            {
                                "datafordelerOpdateringstid": "2021-04-06T11:55:49.595558Z",
                                "byggesag": "000652e2-a099-4be6-a3b6-efc4475a301b",
                                "forretningshaendelse": "Byggesag",
                                "forretningsomraade": "54.15.05.05",
                                "forretningsproces": "0",
                                "id_lokalId": "247270b8-5e4d-4d36-b5cf-18cfbf735c5a",
                                "id_namespace": "http://data.gov.dk/bbr/sagsniveau",
                                "kommunekode": "0101",
                                "niveautype": "1",
                                "registreringFra": "2017-06-02T12:29:37.000000Z",
                                "registreringsaktoer": "BBR",
                                "registreringTil": null,
                                "sagsdataBygning": null,
                                "sagsdataEnhed": null,
                                "sagsdataEtage": null,
                                "sagsdataGrund": "304b8453-8212-407d-86d6-681c3faefd60",
                                "sagsdataOpgang": null,
                                "sagsdataTekniskAnlaeg": null,
                                "sagstype": "0",
                                "stamdataBygning": null,
                                "stamdataEnhed": null,
                                "stamdataEtage": null,
                                "stamdataGrund": "7e86c455-17d3-4d13-ae7f-24367d6966fc",
                                "stamdataOpgang": null,
                                "stamdataTekniskAnlaeg": null,
                                "status": "9",
                                "virkningFra": "2016-12-01T11:23:23.517000Z",
                                "virkningsaktoer": "BBR",
                                "virkningTil": null
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```


Kode 10.6: Resultat fra query (begrænset til ”first: 1”)