Klimadatastyrelsen

Transitionsguide for GraphQL-tjenester

Drift og modernisering af Datafordeleren

Januar 2025

Version 2.3 – 10-01-2025

Indholdsfortegnelse

1 Indledning

Dette dokument er en guide til hvordan anvendere kan bruge Datafordelerens entitetsbaserede GraphQL-tjenester.

Dokumentet er opdelt i tre dele:

1) Første del af dokumentet beskriver hvad entitetsbaserede GraphQL-tjenester er og hvordan de bruges.
2) Anden del af dokumentet beskriver hvordan autentifikation og autorisation håndteres.
3) Tredje del af dokumentet forklarer forskelle mellem GraphQL og de nuværende REST-tjenester på Datafordeleren, samt hvordan anvendere kan komme i gang med at anvende GraphQL-tjenester.

Dokumentet er tiltænkt følgende læsere:

1) Eksisterende anvendere af Datafordeleren der bruger den Nuværende Datafordelers REST-tjenester, som ønsker at komme i gang med at bruge de entitetsbasereede GraphQL-tjenester i stedet.
2) Nye anvendere, der vil i gang med at bruge GraphQL-tjenesterne på Datafordeleren.

1.1 Begreber

I dette dokument bruges begreberne i Tabel 1. Bemærk at begreberne også er yderligere forklaret i løbet af dokumentet med eksempler.

Begreb	Beskrivelse
Nuværende Datafordeler	
REST-tjenester:	REST-tjenester er en metode til at få data fra Datafordeleren. Man kan som anvender hente data ved at bruge en URL. REST-tjenester er beskrevet her https://confluence.sdfi.dk/pages/viewpage.action?pageId=17138461.
Moderniserede Datafordeler	
GraphQL:	GraphQL beskrives i denne vejledning.
Entitet:	En entitet er en fysisk repræsentation af et registerobjekt som findes i Grunddatamodellen. De total- og deltadownload, der kan hentes fra Datafordeleren, indeholder hver især data bestående af en enkelt entitet.
GraphQL-skemaer:	GraphQL-skemaer er .graphql-filer der genereres på baggrund af registrenes datamodeller og er yderligere beriget med metadata fra hvert register. Skemaerne beskriver hvordan data er struktureret og hvordan data kan hentes.
Paging:	Paging er når resultater inddeles i sider således at resultater returneres én side ad gangen.

Tabel 1: Begrebsliste.

2 GraphQL på Datafordeleren

GraphQL er en funktionalitet på Datafordeleren til udstilling af data. Dette dokument beskriver hvordan anvendere kan benytte GraphQL-tjenesterne til at hente tabulære data fra registrene på Datafordeleren.

2.1 Hvad er GraphQL?

GraphQL er et moderne API-sprog, der gør det muligt for anvendere at hente data på en effektiv og fleksibel måde. I modsætning til Datafordelerens nuværende REST-tjenester, hvor man ofte får for meget data, giver GraphQL mulighed for at hente præcis de oplysninger, anvenderen skal bruge. GraphQL-tjenesterne er derfor velegnede når anvendere ønsker at hente mindre datamængder og ikke nødvendigvis ønsker at postprocessere data efterfølgende.

Med GraphQL kan anvendere definere hvilke datafelter de ønsker returneret. Dette betyder, at de kun modtager de oplysninger, de har brug for, hvilket kan forbedre ydeevnen og reducere mængden af data, der overføres.

Anvendere kan bruge GraphQL-tjenesterne ved at sende en forespørgsel, der beskriver de data de ønsker, til Datafordeleren, hvorefter serveren kun svarer med de specifikke data, der blev anmodet om. Dette kan blandt andet gøres gennem en desktop applikation eller gennem browseren ved brug af en browser-extension.

Yderligere information om GraphQL kan findes på **graphql.org**.

Måden man bruger GraphQL-tjenesterne på, er ved at sende en såkaldt *query* til et af Datafordelerens GraphQL-endepunkter. En query er anvendernes måde at definere hvilke data de ønsker at hente. Et eksempel på en query kan ses i Figur 1 nedenfor.

```
query {
  BBR_Bygning( # Entitet
    where: { # Standardfiltre
      kommunekode: { eq: "0550" }
      virkningsaktoer: { startsWith: "Konvertering" }
    }
    virkningstid: "2024-11-12T14:41:33Z" # Bitemporalt filter
    first: 10 # Paging
  ) {
    pageInfo { # Paginginformation
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
    nodes {
      id_lokalId
      kommunekode
      virkningsaktoer
    }
  }
}
```

Figur 1: Eksempel på en GraphQL-query

GraphQL-queries fungerer således:

- **Entitet** (beskrevet i afsnit 2.2): Hver query skal specificere hvilken entitet den omhandler. Eksemplet ovenfor viser en query for entiteten BBR_Bygning fra BBR og specifikt kun for 3 kolonner (angivet i ”nodes”): id_lokalId, kommunekode og virkningsaktoer.
- **Endepunkt** (beskrevet i afsnit 2.3): En query sendes til et GraphQL-endepunkt som et GET- eller POST-request.
- **Standardfiltre** (beskrevet i afsnit 2.4): En query kan indeholde standardfiltre der filtrerer data. Query’en ovenfor indeholder to standardfiltre: et lighedsfilter (eq: "0550") på kommunekode-kolonnen og et tekstfilter (startsWith: "Konvertering") på virkningsaktoer-kolonnen.

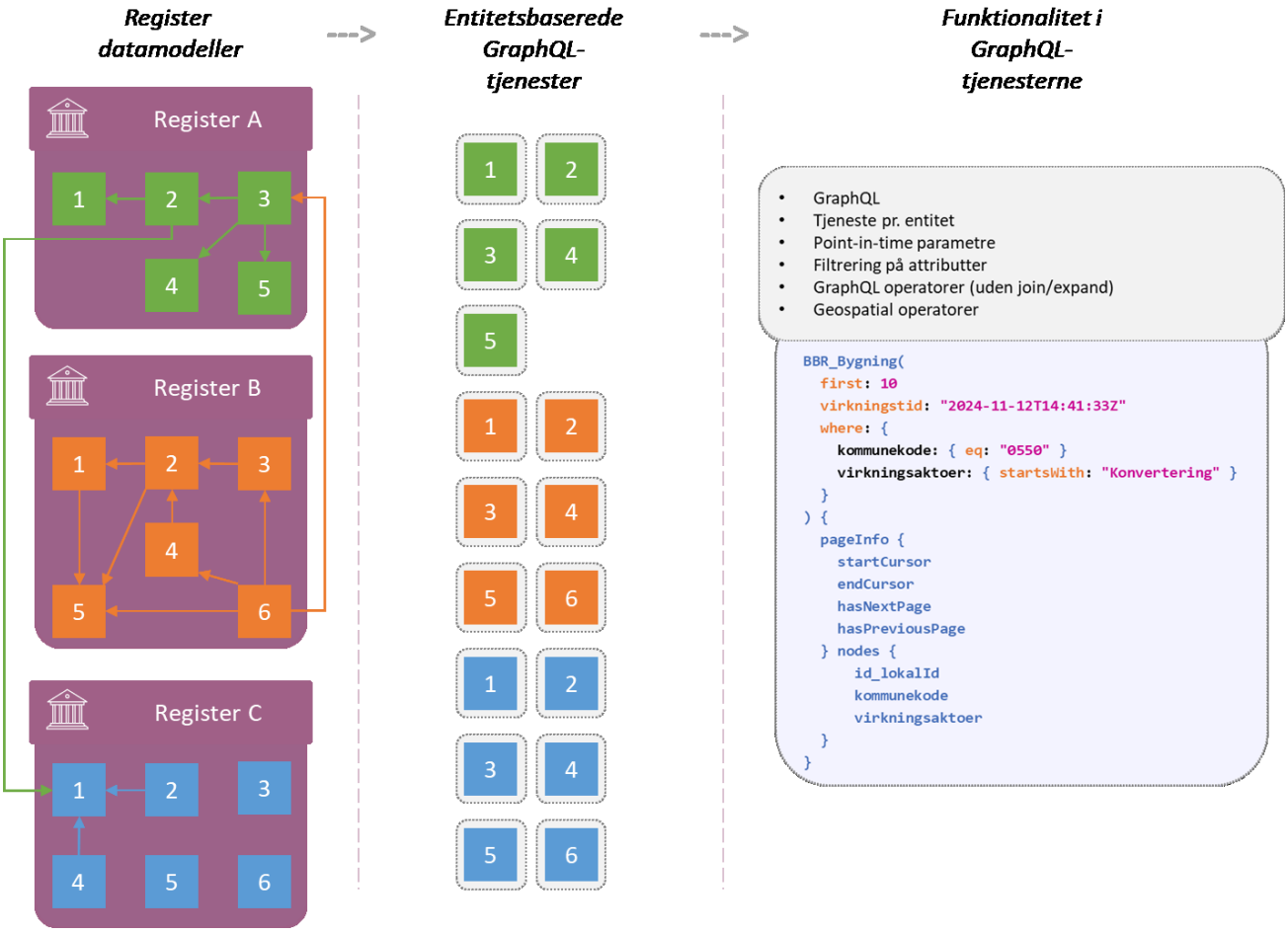
- **Geometriske filtre** (beskrevet i afsnit 2.5): En query kan også indeholde geometriske filtre, der filtrerer data på baggrund af geometri-felter.
- **Bitemporale filtre** (beskrevet i afsnit 2.6): En query kan også indeholde bitemporale filtre. Query'en oven for indeholder et bitemporalt point-in-time-filter (virkningstid: "2024-11-12T14:41:33Z") på virkningstidspunktet.
- **Paging** (beskrevet i afsnit **Fejl! Henvisningskilde ikke fundet.**): En query kan også definere hvordan data pagineres. Query'en oven for returnerer de første 10 rækker (first: 10) givet filtreringen samt paginginformation for resultatsættet (angivet i "pageInfo").

2.2 Om entitetsbaserede GraphQL-tjenester

GraphQL-tjenester tilbydes som **entitetsbaserede GraphQL-tjenester** og baserer sig på de datamodeller som registrene har på Datafordeleren.

En entitet svarer til en tabel defineret i registrenes datamodel fra Grunddatamodellen: https://datafordeler.dk/grunddatamodel/ En entitet kan eksempelvis være en "Adresse" fra DAR, en "Bygning" fra BBR eller et "Jordstykke" fra MAT. Det at GraphQL-tjenesterne er **entitetsbaserede** betyder at et kald til en af tjenesterne returnerer data for en enkelt type af entitet. De entitetsbaserede GraphQL-tjenester leverer udelukkende tabulære data og ikke rasterdata (billeddata). Tabulære data forstås som strukturerede data der kan opbevares i tabeller, som fx adresser men ikke billedtyper som skærmkort.

Brug af API'et beskrives nærmere i afsnittet 2.11.



Figur 2: Opsummering af GraphQL-funktionalitet

En opsummering hvad GraphQL-tjenesterne består af, kan ses i ovenstående figur. Figuren viser 3 registre:

- **Register A**, som har 5 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret og har også en enkelt reference til en entitet i register C udenfor registeret.
- **Register B**, som har 6 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret og har også en enkelt reference til en entitet i register A udenfor registeret.
- **Register C**, som har 6 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret.

I modsætning til registrenes datamodeller har de entitetsbaserede GraphQL-tjenester på nuværende tidspunkt ingen relationer men kaldes enkeltvis per entitet.

2.3 GraphQL-endepunkter

GraphQL-tjenesterne udstilles gennem URL'er, der følger nedenstående form:

https://graphql.datafordeler.dk/<register>/<version>

<register> er forkortelsen for det register anvenderen forsøger at hente og <version> er versionen for det pågældende register. Versionering er yderligere beskrevet i afsnit 2.9.

Alle registre, hvis entiteter udstilles via GraphQL-tjenesterne, er vist i Tabel 2 nedenfor.

Register	Forkortelse
Bygnings- og Boligregistret	BBR
Danmarks Administrative Geografiske Inddeling	DAGI
Danmarks Adresseregister	DAR
Danmarks Fikspunktregister	FIKSPUNKT
DHM Højdekurver	DHMHoejdekurver
DHM Oprindelse	DHMOprindelse
Danske Stednavne	DS
Det Centrale Virksomhedsregister	CVR
Ejendomsbeliggenhedsregistret	EBR
Ejendomsvurdering	VUR
Ejerfortegnelsen	EJF
GeoDanmark Vektor	GEODKV
Historiske kort og data	HISTKORT
Matriklen2	MAT
Skatteforvaltningens Virksomhedsregister	SVR

Tabel 2: Oversigt over registre der udstilles via GraphQL-tjenesterne.

2.3.1 Adgangsbegrænsede registre

Nogle registre indeholder beskyttet og sensitivt data som kræver speciel adgang for at kunne tilgås. Beskyttede og sensitive data kaldes på Datafordeleren også for adgangsbegrænsede data. Adgang til adgangsbegrænsede data tildeles af registrene ved at en anvender opretter et IT-system og laver en ansøgning gennem selvbetjeningen til det specifikke register på vegne af det nyoprettede IT-system, hvorefter det pågældende register først skal godkende ansøgningen før data kan tilgås. Læs mere om hvordan et IT-system oprettes i **Guide til brugeroprettelse på Datafordeler Administration**.

Tabellen neden for viser hvilke registre der indeholder adgangsbegrænsede data

Register	Entiteter
CVR	CVRPerson
EJF	Alle entiteter
SVR	Alle entiteter

Tabel 3: Oversigt over registre med adgangsbegrænsede data.

2.3.2 Specialudviklet GraphQL-tjeneste for CVR

Som beskrevet i afsnit 2.3.1 er der kun et enkelt felt i entiteten CVRPerson fra CVR som indeholder fortrolige data, mens resten af felterne ikke er adgangsbegrænsede. Datafordeleren udstiller i den forbindelse den specialudviklede entitet CVRPersonBegraenset gennem en GraphQL-tjeneste, der giver anvendere mulighed for at hente en ikke-adgangsbegrænset udgave af entiteten CVRPerson. Entiteten CVRPersonBegraenset er identisk med CVRPerson bortset fra at feltet CPRPerson er fjernet.

Filtreringsmulighederne på CVRPersonBegraenset-entiteten er de samme som tillades på CVRPerson-entiteten, bortset fra at det ikke vil være muligt at filtrere på CPRPerson-feltet. GraphQL-skemaet vil ligeledes ligne det for CVRPerson-entiteten, dog uden CVRPerson-feltet.

Tjenesten er tilgængelig på følgende URL:

https://graphql.datafordeler.dk/CVR/custom/<version>

2.4 Standardfiltrering

GraphQL-tjenesterne understøtter almindelige sammenligningsoperatorer og filtreringsmuligheder (f.eks. <, >, =, in) samt tekstfiltre som “contains”, “startWith” med flere. Som udgangspunkt er alle filtre aktiveret, hvis filtrering er aktiveret på et felt. Der er dog felter, hvor bestemte filtre ikke er tlladt.

De forskellige datatyper og tilhørende operatorer kan ses i Tabel 4. Filtreringsreglerne for hver entitet og felt er defineret i GraphQL-skemaet for det pågældende register. Læs mere om GraphQL-skemaer i afsnit 2.8. Det er muligt at bruge flere filtre i den samme query, selvom der er nogle begrænsninger for, hvordan filtrering er tilladt.

Begræsninger for filtrering for hver entitet og dets felter fremgår af registrets GraphQL-skema. GraphQL-skemaer er beskrevet i afsnit 2.8.

Datatyper	Understøttede operatorer
Comparable (Date, DateTime, Float, Int, Long, Short, TimeSpan)	• Equal to (eq) • Less than (lt) • Less than or equal to (lte) • Not less than (nlt) • Not less than or equal to (nlte) • Greater than (gt) • Greater than or equal to (gte) • Not greater than (ngt) • Not greater than or equal to (ngte)
Text (String)	• Equals to (eq) • Starts with (startsWith)
Boolean	• Equals to (eq) • Not equals to (neq)

Tabel 4: Understøttede datatyper og operatorer

2.5 Geometrisk filtrering

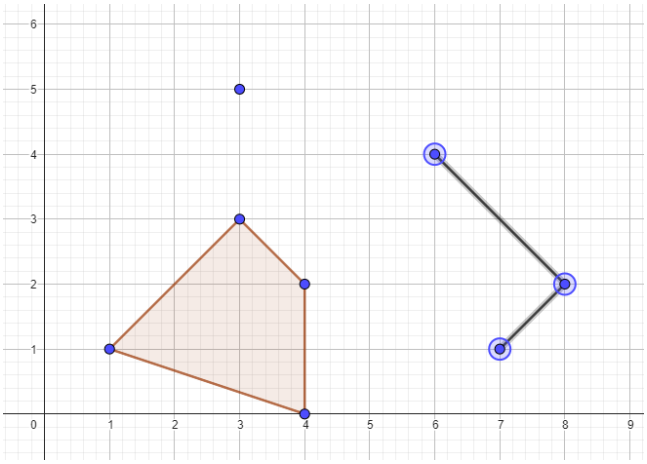
GraphQL-tjenesterne understøtter også geospatiale data og anvendere kan derfor benytte filtre i form af WKT-strenge (Well Known Text) som gives som input til de geometriske filtre. Disse WKT'er repræsenterer geospatiale data, enten i form af en koordinattype, en geometri-type eller lignende datatyper.

De mest almindelige typer af geometriske former er punkter, linjer og polygoner. Et punkt repræsenteres af et koordinatsæt med to eller tre værdier, afhængigt af om det er et punkt i to eller tre dimensioner. En linje repræsenteres af koordinatparrene eller koordinattripletterne for de punkter, der udgør linjen, og en polygon repræsenteres af koordinatpar eller koordinattripletter, der udgør hjørnerne af polygonen. Et eksempel på WKT-formatet for hver af de tre geometrityper i 2D, er vist i Figur 3.

```
Point
 POINT (3 5)

Line
 LINESTRING (6 4, 8 2, 7 1)

Polygon
 POLYGON (1 1, 3 3, 4 2, 1 1)
```



Figur 3: Eksempler på 2D-geometrier

Datafordeleren understøtter en række forskellige operatorer. Alle operatorerne fremgår sammen med et link til deres PostGIS-dokumentation i Tabel 5 nedenfor, og hver af funktionerne tager to geometriske former og returnerer en Boolean værdi. Kombinationer af forskellige filtre understøttes også ved hjælp af AND/OR-operatorer.

Datatyper	Understøttede operatorer	PostGis Link
Geometri	contains(koordinatsystem, geometri)	https://postgis.net/docs/ST_Contains.html
	covers(koordinatsystem, geometri)	https://postgis.net/docs/ST_Covers.html
	coveredBy(koordinatsystem, geometri)	https://postgis.net/docs/ST_CoveredBy.html
	crosses(koordinatsystem, geometri)	https://postgis.net/docs/ST_Crosses.html
	intersects(koordinatsystem, geometri)	https://postgis.net/docs/ST_Intersects.html
	within(koordinatsystem, geometri)	https://postgis.net/docs/ST_Within.html
	overlaps(koordinatsystem, geometri)	https://postgis.net/docs/ST_Overlaps.html
	touches(koordinatsystem, geometri)	https://postgis.net/docs/ST_Touches.html

Tabel 5: Understøttede geometriske filtre og tilhørende PostGis Link

Et eksempel på hvordan operatoren contains bruges kan ses i Figur 4:

```
query {
  DAR_NavngivenVej(
    where: {
      vejnavnebeliggenhed_vejnavneomraade: {
        contains: {
          crs: 25832, # koordinatsystem
          wkt: "POLYGON((
            507269.454538909 6220414.46082892,
            507305.574611149 6220539.06107812,
            507470.77494155  6220525.9010518,
            507433.33273166  6220424.33582511,
            507384.047971257 6220421.36601678,
            507269.454538909 6220414.46082892
          ))" # geometri
        }
      }
    }
  ) {
    nodes {
      id_lokalId
    }
  }
}
```

Figur 4: Eksempel på en GraphQL-query med det geometriske filter contains.

De geometriske filtre er implementeret således, at hvis man læser informationen i PostGis-linkene i Tabel 5, er geometri A altid værdien af registerdatafeltet, og geometri B altid inputgeometrien. Det vil sige, at rækkefølgen er operator(<registerdatafelt>, <input>). Hvis man ønsker den modsatte rækkefølge, bør man i stedet bruge en anden geometrisk operator, der repræsenterer den omvendte operator. En oversigt over disse kan ses i Tabel 6.

Geometrisk operator	Omvendte operator
contains	within
covers	coveredBy
coveredBy	covers
crosses	*Ingen. crosses er symmetrisk*
intersects	*Ingen. intersects er symmetrisk*
within	contains
overlaps	*Ingen. overlaps er symmetrisk*
touches	*Ingen. touches er symmetrisk*

Tabel 6: Geometriske operatorer og deres omvendte operatorer.

Syntaksen for at bruge geometriske filtre, og hvilke felter der kan filtreres geometrisk, er specificeret af GraphQL-skemaet for den pågældende tjeneste, som man ønsker at bruge. Når man angiver WKT-strengen for den geometri, der skal filtreres på, skal man også angive, hvilket koordinatsystem (CRS) deres WKT tilhører, som illustreret i eksemplet i Figur 4.

Da geospatiale data kan have forskellige formater afhængigt af, hvilket register det drejer sig om, udfører Datafordeleren udelukkende validering på om det angivne koordinatsystem (CRS) er det samme koordinatsystem som registrets data og at den angivne WKT er valid og repræsenterer en topologisk valid geometri som specificeret i OGC SFS-specifikationen.

2.6 Bitemporal filtrering

Der er to typer queries, som man kan udføre på bitemporale data:

- **Interval-queries**, hvor man ønsker data, der er bitemporalt gyldige i en bestemt tidsperiode.
- **Point-in-time-queries**, hvor man ønsker bitemporalt gyldige data pr. deres angivne tidspunkt.

Ved interval-queries har man mulighed for at specificere standard-tidsfiltre på de bitemporale kolonner.

Ved point-in-time-queries understøttes valgfrie argumenter for registreringstid og virkningstid, som kan angives, når man sender queries. Når disse argumenter ikke gives, returneres fuldt bitemporale data. Bemærk, at for at få data som er gyldigt på et givent tidspunkt, skal man sætte *både* virkningstid og registreringstid til forespørgselstidspunktet for at få data, der er gyldige på det pågældende tidspunkt.

GraphQL-tjenesterne understøtter muligheden for at kombinere point-in-time-filtrering med enhver anden filtrering, der er tilladt, så man kan hente data, der overholder det angivne tidspunkt samt eventuelle intervalfiltre, geo-filtre eller lignende, de har angivet. Det vil sige, når point-in-time-filteret kombineres med andre filtre, vil forespørgslen ende med at være "point-in-time-filter OG (andre filtre)". Det er ikke muligt at sige point-in-time-filtreret 'ELLER' noget andet filter.

2.6.1 Bitemporal filtrering for CVR

Alle registre der understøtter bitemporal filtrering, understøtter filtrering på både virkningstidspunkt og registreringstidspunkt bortset fra CVR. CVR er det eneste register, hvor anvendere ikke kan specificere et registreringspunkt i deres query. For CVR sættes registreringstidspunktet derimod *altid* til tidspunktet for forespørgslen, dvs. "registreringstid = NOW()". Det er muligt at filtrere på virkningstidspunktet på sædvanlig vis.

2.7 Paging

GraphQL-tjenesterne understøtter paging af resultater. At resultaterne *pagineres* vil sige at resultaterne inddeles i "sider" og returneres én ad gangen. Anvendere vil således få én side returneret ad gangen og kan efterfølgende lave et nyt kald for at få næste side for på den måde at gennemgå hele datasættet.

Pagingen er cursor-baseret, hvilket vil sige at der returneres en reference, en såkaldt *cursor,* til første og sidste resultat (for edges ved hvert enkelt resultat) i det returnerede datasæt ved hvert kald. Denne cursor kan derefter bruges til at hente flere sider ved at anmode om mere data fra cursorens værdi. I praksis kan anvendere bruge cursor-værdien i en efterfølgende query, til at hente den næste side af data, startende fra den position, der er angivet ved den pågældende cursor. Paging er implementeret således at flere identiske anmodninger ved hjælp af den samme cursor vil returnere de samme data, forudsat at registrene ikke opdaterer de underliggende data i mellemtiden.

Paging er udelukkende implementeret som fremadgående paging. Det vil sige, at anvenderne kan hente den næste side af resultater fra en given cursor. Bagudgående paging understøttes ikke.

2.7.1 Paging i queries: PageInfo, nodes og edges

Paging består af tre datastrukturer: *PageInfo* samt *nodes* eller *edges*.

PageInfo indeholder metadata om den nuværende paging og består af følgende felter:

- **hasNextPage**: Dette felt angiver, om der er endnu en side med resultater.
- **hasPreviousPage**: Dette felt angiver, om der eksisterer en forrige side med resultater. Da vi ikke understøtter bagudgående pagination, er dette altid sat til false.
- **startCursor**: Dette felt er en cursor, der peger på det første datapunkt i resultatdatasættet.
- **endCursor**: Dette er en cursor, der peger på det sidste datapunkt i resultatsættet. Denne cursor kan bruges af anvenderen til at hente den næste side af data.

Mens metadata kan hentes via PageInfo, kan data hentes enten som en liste af *nodes* eller en liste af *edges*. *Nodes* og *edges* adskiller sig på følgende måde:

- I resultatsættet er *nodes* en liste af datapunkter.
- I resultatsættet er *edges* en liste af datapunkter med et ekstra metadatafelt: Cursoren, der matcher hver datapunkt.

Det anbefales, at anvendere benytter *nodes* til standard paginganmodninger, da PageInfo-metadataene er tilstrækkelige. Hvis anvenderne har brug for at paginere fra en hvilken som helst position på den aktuelle side og ikke fra slutningen af den aktuelle side, skal *edges* benyttes.

Det er tilladt for anvendere at specificere i deres GraphQL-queries, at de ønsker både *nodes* og *edges* returneret. Dette frarådes, da resultatsættet vil indeholde de samme data to gange.

2.7.2 Inputargumenter for paging

Paging har to inputargumenter, der kan specificeres i en query:

- *first*: Dette argument angiver hvor mange resultater anvenderen ønsker returneret. Dette er sidestørrelsen og skal være et ikke-negativt tal. Hvis dette argument udelades fra query'en, returneres standardantallet af resultater.
- *after*: Dette argument angiver cursoren til det første datapunkt i det resulterende datasæt. Hvis dette argument udelades fra query'en, returneres den første side.

Hvis anvenderen angiver ugyldige værdier for nogen af disse argumenter, vil tjenesterne returnere en fejlmeddelelse, der informerer dem om, at de har angivet ugyldige pagingargumenter.

Standardantallet af resultater, der returneres hvis brugeren ikke har angivet en første værdi, er 100. Den maksimalt tilladte sidestørrelse er 1000, og hvis værdien af *first* er større end den maksimalt tilladte sidestørrelse returneres en fejlmeddelelse.

2.7.3 Eksempler på paging-queries

Følgende query kan bruges til at hente de første 5 datapunkter fra første side af BBR_Bygning-entiteten:

```
query {
  BBR_Bygning(first: 5, after: null) {
    pageInfo {
      endCursor
      hasNextPage
    } nodes {
      id_lokalId
    }
  }
}
```

Figur 5: Eksempel på en GraphQL-query med *first=5*.

Tjenesten returnerer følgende resultat:

```
{
  "data": {
    "BBR_Bygning": {
      "pageInfo": {
        "endCursor": "1rsNAAAAAAA=",
        "hasNextPage": true
      },
      "nodes": [
        {
          "id_lokalId": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
        },
        {
          "id_lokalId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"
        },
        {
          "id_lokalId": "cccccccc-cccc-cccc-cccc-cccccccccccc"
        },
        {
          "id_lokalId": "dddddddd-dddd-dddd-dddd-dddddddddddd"
        },
        {
          "id_lokalId": "eeeeeeee-eeee-eeee-eeee-eeeeeeeeeeee"
        }
      ]
    }
  }
}
```

Figur 6: Resultat fra query'en i Figur 6.

Fra hasNextPage i resultatet oven for kan man se at der er flere sider, da den er sat til true. For at hente den næste side kan anvenderen derfor bruge endCursor som *after*-værdi i den efterfølgende query:

```
query {
  BBR_Bygning(first: 5, after: "1rsNAAAAAAA=") {
    pageInfo {
      endCursor
      hasNextPage
    } nodes {
      id_lokalId
    }
  }
}
```

Figur 7: Eksempel på en GraphQL-query med *after="1rsNAAAAAAA="*.

Ved iterativt at gentage dette mønster kan alt den ønskede data hentes.

2.8 GraphQL-skemaer

For alle de registre der er udstillet gennem Datafordelerens GraphQL-tjenester, kan anvendere hente GraphQL-skemaer, der beskriver hvordan data er struktureret. Skemaerne specificerer de typer data, der kan hentes, relationerne mellem disse typer og de filtreringsoperationer, der kan understøttes for de pågældende felter. Anvendere kan benytte skemaerne til at se hvilke queries de kan sende, og hvilke data de kan hente.

Skemaerne genereres på baggrund af registrenes datamodeller og er beriget med metadata for hvert register. Denne metadataudtrækningsproces involverer at tage relevant information fra datamodellerne og inkorporere den i skemaerne. Denne ekstra metadata giver anvendere mere omfattende indsigt i de tilgængelige data. Skemaerne findes separat for hver register og kan hentes ved at tilgå nedenstående url:

https://graphql.datafordeler.dk/<register>/<version>/schema

<register> er forkortelsen for det register anvenderen forsøger at hente og <version> er versionen for det pågældende register. Læs mere om versionering i afsnit 2.9. Skemaet for version 1 af Skatteforvaltningens Virksomhedsregister (SVR) hentes for eksempel på følgende url:

https://graphql.datafordeler.dk/SVR/v1/schema

2.9 Versionering

I GraphQL-tjenesterne versioneres hvert register separat, og alle entiteter, der tilhører det samme register, versioneres under det samme versionsnummer som registret. Det vil sige, at DAR og BBR kan eksistere separat som DAR/v2 og BBR/v5. Men hvis BBR har en change til Bygning-entiteten i deres datamodel, vil alle entiteter i BBR blive opgraderet til BBR/v6, og BBR/v5 vil derefter efter en overgangsperiode blive udfaset.

2.10 Cost-beregning

Som en sikkerhedsforanstaltning og af performance-hensyn tilskrives hver query en cost. En cost er en beregnet talværdi der indfanger hvor kompleks og omkostningstung en given query er for Datafordeleren at processere. Costen udregnes inden en query processeres, hvorefter den pågældende query enten sendes videre i systemet eller bliver afvist hvorved anvenderen vil modtage en fejlmeddelelse der informerer om at query'en var for kompleks.

Udregningen af cost er blandt andet baseret på følgende specifikationer:

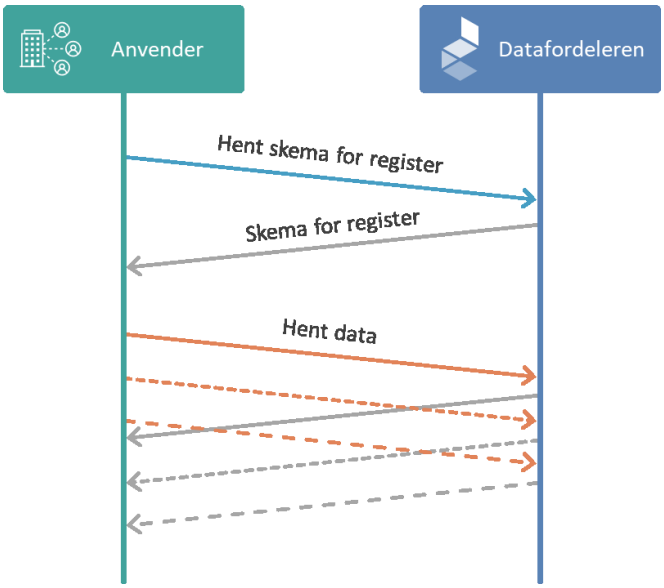
- **Sidestørrelse**: Størrelse på paging-siderne der returneres. Jo større sider, jo mere data returneres, hvilket giver en højere cost.

- **Filtre**: Antallet og type af filtre der benyttes i en query påvirker også costen. Jo flere filtre, jo højere cost. Derudover er geometri-filtre mere omkostningstunge og resulterer defor i en højere cost. Se evt. afsnit 2.4, 2.5 og 2.6 for en oversigt over de forskellige typer af filtre.
- **Entiteter**: Almindelige og store entiteter har forskellige omkostninger associeret med dem. Store entiteter er mere omkostningstunge end mindre entiteter.
- **Felter**: Forskellige typer felter har forskellige omkostninger. Geometrier og sammensatte[1] felter er dyrere end almindelige felter.

Som anvender skal man ikke tage stilling til hvor omkostningstung en query er så længe den ikke overstiger den maksimale cost, i hvilket tilfælde den pågældende query vil blive afvist med en fejlbesked.

2.11 Sådan anvender du entitetsbaserede GraphQL-tjenester

GraphQL-tjenesterne på Datafordeleren tilgås via et GraphQL-API. Dette afsnit beskriver, hvordan det vil være muligt at danne et overblik over hvilke entiteter man kan tilgå, samt hvordan data hentes. GraphQL-tjenesterne kan tilgås på registerbasis og kræver at du benytter en af de tre autentifikationsmetoder beskrevet i afsnit 0.



Figur 8: Overblik over samlet anvendelsesmønster for GraphQL-tjenesterne.

Ovenstående figur viser hvordan GraphQL-API'et overordnet set anvendes. Her hentes GraphQL-skemaet først for det pågældende register ved at tilgå https://graphql.datafordeler.dk/<register>/<version>/schema, hvorefter det ønskede data hentes, ved at sende en query til https://graphql.datafordeler.dk/<register>/<version>.

[1] Et sammensat felt er et felt der indeholder andre felter.

2.11.1.1 Hent Schema

Hent Schema	
URL	https://graphql.datafordeler.dk/<register>/<version>/schema
HTTP-metode	GET
Headere i forespørgsel	Content-Type: application/json
Returværdier	• Returnerer HTTP 200 – OK, ved succes. • Returnerer HTTP 400 – Bad Request, ved angivelse af forkerte parametre. • Returnerer HTTP 404 – Not Found, hvis den angivne ressource ikke kan findes. • Returnerer HTTP 500 – Internal Server Error, hvis der er sket en ukendt fejl
Adgang	Tjenestebruger med brugernavn og password, IT-system med API-nøgle, OAuth Shared Secret eller OAuth Certifikat.

2.11.1.1.1 Parametre

Navn	Type	Beskrivelse	Obligatorisk?
Register	String	Angiver hvilket register det pågældende skema skal hentes for. Følgende registre kan angives: • DAR • DAGI • BBR • DHMOprindelse • DHMHoejdekurver • MAT • EBR • FIKSPUNKT • DS • GEODKV • CVR • CVR/custom • EJF • SVR • VUR • HISTKORT	Ja
Version	String	Angiver hvilken version af datamodellen det pågældende register skal være.	Ja

2.11.1.1.2 Returværdier

Denne tjeneste returnerer altid en fil ved HTTP 200 – OK. Filen indeholder et GraphQL-schema med metadata om GraphQL-tjenesten, der beskriver hvilke entiteter der findes for det pågældende register og hvordan data fra disse hentes.

2.11.1.1.3 Eksempler på brug af tjenesten

Hent skema for DAGI med versionsnummer v2 ved brug API-nøgle:

- https://graphql.datafordeler.dk/DAGI/v2/schema?apiKey=placeholderNoegle

2.11.1.2 Send query

Send query	
URL	https://graphql.datafordeler.dk/<register>/<version>
HTTP-metode	GET & POST
Headere i forespørgsel	For POST-requests: • Content-Type: application/json For GET- og POST-requests (valgfrit)[2]: • Accept: application/graphql-response+json
Body	For GET-requests: • Ingen. Query-parametre skal URL-encodes i overensstemmelse med GraphQL-standarden. For POST-requests: • Valid GraphQL-query.
Returværdier[3]	• Returnerer HTTP 200 – OK, ved succes. • Returnerer HTTP 400 – Bad Request, ved angivelse af forkerte parametre. • Returnerer HTTP 404 – Not Found, hvis den angivne ressource ikke kan findes. • Returnerer HTTP 500 – Internal Server Error, hvis der er sket en ukendt fejl
Adgang	• Ikke-adgangsbegrænset data: IT-system med API-nøgle, OAuth Shared Secret eller OAuth Certifikat. • Adgangsbegrænset data: IT-system med OAuth Shared Secret eller OAuth Certifikat *og* på en defineret IP-Allowlist og så skal IT-systemet have eksplicit godkendelse fra det pågældende register. Se afsnit 0 for yderligere information.

[2] Anvendere *bør* inkludere Accept-headeren som specificeret her, hvis de vil opt-in til moderne GraphQL-over-HTTP. Dette er dog ikke et krav. Hvis Accept-headeren ikke inkluderes fås andre HTTP-statuskoder.

[3] Returværdierne for HTTP-statuskoderne afhænger af hvilken Accept-header der specificeres. Disse værdier er hvis Accept-headeren ikke er angivet som beskrevet i "Headere i forespørgsel".

2.11.1.2.1 Parametre

Navn	Type	Beskrivelse	Obligatorisk?
Register	String	Angiver hvilket register det pågældende skema skal hentes for. Følgende registre kan angives: • DAR • DAGI • BBR • DHMOprindelse • DHMHoejdekurver • MAT • EBR • FIKSPUNKT • DS • GEODKV • CVR • CVR/custom • EJF • SVR • VUR • HISTKORT	Ja
Version	String	Angiver hvilken version af datamodellen det pågældende register skal være.	Ja

2.11.1.2.2 Returværdier

Denne tjeneste returnerer et json-response ved HTTP 200 – OK.

2.11.1.2.3 Eksempler på brug af tjenesten

Hent de første 100 datapunkter fra entiteten DAR_Adresse fra version 1 af DAR ved at sende nedenstående query ved brug af API-nøgle til endepunktet:

- https://graphql.datafordeler.dk/DAR/v1?apiKey=placeholderNoegle

```
query {
  DAR_Adresse(first: 100) {
    pageInfo {
      endCursor
      hasNextPage
    } nodes {
      adressebetegnelse
      bygning
      id_lokalId
      registreringFra
      registreringTil
      status
      virkningFra
      virkningTil
    }
  }
}
```

Hent data der var gældende den 12/11/2024, kl. 14:41:33 med kommunekode “0550” fra entiteten BBR_Bygning fra version 1 af BBR ved at sende nedenstående query som et POST-request ved brug af API-nøgle til endepunktet:

- https://graphql.datafordeler.dk/BBR/v1/?apiKey=placeholderNoegle

```
query {
  BBR_Bygning(
    virkningstid: "2024-11-12T14:41:33Z"
    registreringstid: "2024-11-12T14:41:33Z"
    where: {
      kommunekode: { eq: "0550" }
    }
  ) {
    pageInfo {
      endCursor
      hasNextPage
    } nodes {
      husnummer
      id_lokalId
      registreringFra
      registreringTil
      status
      virkningFra
      virkningTil
    }
  }
}
```

3 Autentifikation & autorisation for GraphQL-tjenester

Når en anvender ønsker at benytte GraphQL-tjenester kræver det at anvenderen er autentificeret. Datafordeleren har en række forskellige **autentifikationsmetoder**, som man kan vælge at benytte afhængigt af om man forsøger at tilgå frit tilgængeligt eller adgangsbegrænset data:

1) IT-system med **API-nøgle**: En API-nøgle er en privat nøgle, som bruges til at autentificere mod et API. Denne nøgle bruges som en del af godkendelsesprocessen, hvor en klient får adgang til et API uden at afsløre brugerens loginoplysninger. API-nøgler bruges som et query-parameter. Læs om hvordan API-nøgler oprettes og bruges i **Guide til brugeroprettelse på Datafordeler Administration**.

2) IT-system med **OAuth Shared Secret**: En OAuth Shared Secret er en privat nøgle, som deles mellem en klient (f.eks. en applikation) og udbyderen (f.eks. en service som Datafordeleren). Denne nøgle bruges som en del af godkendelsesprocessen, hvor en klient får adgang til ressourcer på vegne af en bruger uden at afsløre brugerens loginoplysninger. For at blive autentificeret ved hjælp af OAuth Shared Secret skal man som anvender først oprette en Shared Secret i selvbetjeningsportal. Læs om hvordan OAuth Shared Secret oprettes i **Guide til brugeroprettelse på Datafordeler AdministrationFejl! Henvisningskilde ikke fundet.**. Når man har oprettet en Shared Secret sendes denne samt Client ID til https://auth.datafordeler.dk/realms/distribution/protocol/openid-connect/token hvorefter man vil modtage et Access Token. En mere detaljeret beskrivelse af hvordan man får et token kan findes på https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-client-creds-grant-flow#first-case-access-token-request-with-a-shared-secret. Det resulterende token er et Bearer Token der kan bruges til autentifikation ved efterfølgende kald til datafordeleren. Bemærk at dette token kun er validt i 60 minutter efter udstedelsen.

3) IT-system med **OAuth Certifikat**: Klientautentificering med certifikater i OAuth er en sikkerhedsmetode, hvor en klient (f.eks. en applikation) bruger et digitalt certifikat i stedet for en hemmelig nøgle til at bevise sin identitet over for autorisationsserveren. Processen involverer, at klienten præsenterer sit certifikat under TLS-håndtrykket, hvorefter autorisationsserveren validerer det. Denne metode er særligt velegnet til miljøer med høje sikkerhedskrav eller ved håndtering af følsomme data. OAuth Certifikater sættes i kaldets header. Læs om hvordan OAuth Certifikater oprettes og bruges i **Guide til brugeroprettelse på Datafordeler Administration.**

   Bemærk at det *ikke* er muligt at benytte tjenestebruger til autentifikation for at hente data fra entiteterne via GraphQL-tjenesterne. GraphQL-skemaer kan dog godt hentes vha. tjenestebruger.

Adgang til nogle GraphQL-tjenesterne kræver yderligere at IP-adressen for klienten der prøver at hente data ved brug af et IT-system er på IP-Allowlisten for det pågældende IT-system. IP-Allowlisten bruges til at verificere at klienten er autoriseret til at hente det efterspurgte data. For ikke-adgangsbegrænset data er det ikke nødvendigt at specificere en IP-Allowliste, men hvis IP-Allowlisten *er* specificeret *skal* klientens IP-adresse komme fra en af de angivne adresse på listen. Læs mere om IP-Allowlister i **Guide til brugeroprettelse på Datafordeler Administration**.

4 Transitionsguide

Denne side beskriver forskelle mellem Datafordelerens nuværende REST-tjenester og GraphQL-tjenesterne, samt hvordan du som anvender af REST-tjenester, kan komme i gang med at bruge GraphQL-tjenesterne.

Det skal bemærkes at GraphQL-tjenester og REST-tjenester driftes parallelt i en periode, så anvendere af Datafordeleren har mulighed for at skifte fra REST-tjenester til GraphQL-tjenester.

Bemærk at de nuværende REST-tjenester understøtter joins mellem forskellige entiteter. Det gør de entitetsbaserede GraphQL-tjenester *ikke*. For at efterligne en af de nuværende REST-tjenester der sammenstiller 4 entiteter i ét kald, skal man, ved brug af GraphQL, lave fire kald – ét for hver entitet – for derefter selv at efterprocessere og sammenstille resultaterne.

4.1 Indhold i filer

De nuværende REST-tjenester på Datafordeleren giver anvendere mulighed for at hente en delmængde af en entitets datapunkter givet en bestemt URL. GraphQL-tjenesterne giver anvendere mulighed for at specificere nøjagtig hvilke data de har brug for og derved udelade de overflødige data. Det er altså det samme data der udstilles ved de forskellige tjenester, men forskellen er at GraphQL-tjenesterne giver anvendere mulighed for at hente en delmængde af de data der udstilles ved REST-tjenester. Du kan læse mere om de nuværende **REST-tjenester** på **datafordeler.dk**.

GraphQL-tjenesterne er derfor velegnede til at hente begrænsede datamængder uden efterfølgende at skulle processere og smide unødig data væk.

4.2 Anvendereksempel på transition fra de nuværende REST-tjenester til GraphQL-tjenester

Dette eksempel tager udgangspunkt i en anvender som benytter entiteten BBR_Bygning i Frederiksberg kommune (kommunekode: “0147”) og derfor jævnligt bruger REST-tjenester.

Afsnittet giver et eksempel på, hvordan en eksisterende anvender på Datafordeleren kan overgå fra at bruge REST-tjenester i sit system til at bruge GraphQL-tjenester. Afsnittet påbegyndes med en situationsbeskrivelse efterfulgt af konkrete trin der beskriver transitionen.

4.2.1 Situationsbeskrivelse

I dette eksempel laver anvenderen et kald til de nuværende REST-tjenester for at hente følgende oplysninger om alle bygninger i Frederiksberg Kommune:

- Husnummer
- byg007Bygningsnummer
- byg024AntalLejlighederMedKoekken
- byg025AntalLejlighederUdenKoekken
- byg030Vandforsyning

Anvenderen benytter følgende parametre (bemærk at registrerings- og virkningstidspunkt sættes til NOW() hvis ikke angivet):

- Kommunekode = 0147
- PageSize = 1000
- Page = 1

Anvenderen bruger en tjenestebruger med brugernavn og password til at tilgå REST-tjenesten på URL'en:

- https://services.datafordeler.dk/BBR/BBRPublic/1/rest/bygning?username=xxxx&password=yyyy&kommunekode=0147&pagesize=1000&page=1

For at hente de resterende data øges page med én og tjenesten kaldes igen indtil alt data er hentet. Dette resulterer i data som indeholder alle informationer fra entiteten BBR_Bygning og skal derfor efterprocesseres for at fjerne overflødigt data.

4.2.2 Transition

1. Anvender tilgår følgende endepunkt med sin API-nøgle for at hente GraphQL-skemaet for BBR:
   - https://graphql.datafordeler.dk/BBR/v1/schema?apiKey=placeholderNoegle
2. Da anvender nu ved hvilke queries der kan sendes for BBR, sender anvender nu en query til URL'en neden for for at hente de første 1.000 datapunkter fra BBR_Bygning for Frederiksberg Kommune med de ønskede oplysninger:
   - https://graphql.datafordeler.dk/BBR/v1?apiKey=placeholderNoegle

```
query {
  BBR_Bygning(
    first: 1000
    virkningstid: "2024-12-12T12:00:00Z" # Dags dato
    registreringstid: "2024-12-12T12:00:00Z" # Dags dato
    where: {
      kommunekode: { eq: "0147" }
    }
  ) {
    pageInfo {
      endCursor
      hasNextPage
    } nodes {
      husnummer
      byg007Bygningsnummer
      byg024AntalLejlighederMedKoekken
      byg025AntalLejlighederUdenKoekken
      byg030Vandforsyning
    }
  }
}
```

3. Anvender modtager nu de første 1000 datapunkter samt paginginformation i json-format som vist nedenfor:

```
{
  "data": {
    "BBR_Bygning": {
      "pageInfo": {
        "endCursor": "1rsNAAAAAAA=",
        "hasNextPage": true
      },
      "nodes": [
        {
          ... # Data
```

```
        }
      ]
    }
  }
}
```

4. Anvender kan nu hente de resterende data ved iterativt at sende queries, med den resulterende endCursor-værdi fra det forrige svar, så længe det resulterende JSON-svar har hasNextPage=true.

```
query {
  BBR_Bygning(
    first: 1000
    after: "1rsNAAAAAAA=" # endCursor fra forrige svar
    virkningstid: "2024-12-12T12:00:00Z" # Dags dato
    registreringstid: "2024-12-12T12:00:00Z" # Dags dato
    where: {
      kommunekode: { eq: "0147" }
    }
  ) {
    pageInfo {
      ... # Paginginformation
    } nodes {
      ... # Kolonner
    }
  }
}
```

5. Anvender er nu i besiddelse af alle datapunkter fra BBR_Bygning entiteten for Frederiksberg Kommune med de ønskede informationer. Data behøver ikke efterfølgende at blive efterprocesseret.