

Sami Hellsten

# SKAALAUTUVA DATAINTEGRAATIO DRUPAL-JÄRJESTELMÄSSÄ

Diplomityö  
Johtamisen ja talouden tiedekunta  
Kesäkuu 2025

# TIIVISTELMÄ

Sami Hellsten  
Skaalautuva dataintegraatio Drupal-järjestelmässä  
Tampereen Yliopisto  
Johtamisen ja tietotekniikan DI-ohjelma  
Diplomityö  
Kesäkuu 2025

---

Tämä työ pohjautuu erään suuren suomalaisyrityksen Drupal-järjestelmän tarpeeseen välittää kaikki järjestelmän tieto ulkoiseen raportointirajapintaan. Järjestelmä on mittakaavassaan yksi suurimpia ja monimutkaisimpia, joita Suomessa on toteutettu. Olemassa olevat ratkaisut datan vientiin eivät nykyisellään skaalaudu suuriin tietomääriin, vaan esteiksi muodostuu joko äärimmäinen hitaus tai vaikea ja aikaa vievä ylläpito jotka, molemmat näkyvät myös järjestelmän kustannuksissa.

Työn tavoitteena oli löytää sellainen ratkaisumalli, jossa suuret tietomäärät pystytään viemään riittävän nopeasti ulkoiseen järjestelmään siten, että integraation ylläpitäminen onnistuu ilman merkittävää aikapanostusta ja erikoisosaamista. Työssä perehdytään lisäksi eri tapoihin luoda entiteettejä ja tutkitaan vaikuttaako luontitapa merkittävästi käsittelynopeuteen.

Työn tuloksena syntyi ratkaisu, jonka tiedonsiirtonopeus on lähes yhtä tehokas kuin suoran tietokantayhteyden ja ylläpitotarve minimaalinen. Ratkaisun keskeisenä osana toimii optimoitu latausluokka, joka on räätälöity Drupal-ytimen alkuperäisestä entiteettien varastointiluokasta. Optimoitu latausluokka on tämän työn liitteenä ilman sen ympärille toteutettua moduulia jolla tiedonsiirron hallinta on siirretty ohjelmakoodin puolelta käyttöliittymään mahdollistaen integraation käytön ilman vahvaa Drupal-osaamista.

Työssä käydään lisäksi läpi muita ratkaisuvaihtoehtoja ja esitetään niiden vahvuudet sekä heikkoudet. Tuloksista on hyötyä myös sellaisissa tilanteissa missä tiedonvientitarve on rajallinen, etenkin mikäli olemassa olevan ratkaisun tiedonsiirto kestää useita minutteja tai pidempään. Lisäksi työ osoittaa, että entiteettejä rakentaessa on syytä suunnitella tarvittavat kentät huolellisesti, sillä niiden merkitys järjestelmän nopeuteen saattaa olla huomattava.

**Avainsanat:** Drupal, dataintegraatio, suorituskyky

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality -ohjelmalla.

# ABSTRACT

Sami Hellsten  
Scalable data integration in Drupal  
Tampere University  
Master's Programme in Management and Information Technology  
Master's Thesis  
June 2025

---

This work is based on the need of a large Finnish company's Drupal system to transmit all of its data to an external reporting interface. The system is, in terms of scale, one of the largest and most complex Drupal implementations ever carried out in Finland. Existing solutions for data export do not currently scale well to large data volumes; they are either extremely slow or difficult and time-consuming to maintain - both of which also impact the overall system cost.

The goal of this work was to find a solution model capable of transferring large volumes of data to an external system quickly enough, while ensuring that the integration remains maintainable without significant time investment or specialized expertise. The work also explores different methods for creating entities and examines whether the method of creation has a significant impact on processing speed.

The outcome of the project was a solution which data transfer speed is nearly identical to direct database queries, while requiring minimal maintenance. A key component of the solution is an optimized loading class, tailored from Drupal core's original entity storage class. This optimized loader is included as an appendix to this work, without the accompanying module used to manage data transfers via the user interface which enables use of the integration without requiring in-depth Drupal knowledge.

The work also reviews other alternative solutions, highlighting their strengths and weaknesses. The findings are useful even in scenarios where the data export needs are limited, particularly if the existing solution takes several minutes or more to complete data transfers. Additionally, the work demonstrates that careful planning of the necessary fields during entity construction is important, as they can significantly affect system performance.

**Keywords:** Drupal, data integration, performance

The originality of this thesis has been checked using the Turnitin Originality service.

# ALKUSANAT

Voitto. Täysipäiväisenä työntekijänä maisteriopintojen suorittaminen tuntui aikanaan helpolta tavalta laajentaa omaa osaamista ja kehittää uusia taitoja, olinhan jo aiemmatkin opinnot saattanut ajallaan loppuun, vaikka puolet niistäkin tapahtui työn ohessa. Tämä lopputyö osoittautui kuitenkin äärimmäisen raskaaksi operaatioksi, eikä arjen välistä löytynyt vuosiin sellaista hetkeä, jolloin energiaa ja mielenkiintoa suoritukselle olisi ollut tarjolla samanaikaisesti. Voimia jokaiselle, joka taistelee saman ongelman kanssa; mitä suurempi este, sitä paremmalta sen ylittäminen tuntuu.

Olen urallani päässyt kehittämään useita sellaisia ratkaisuja, joista voi olla vähintäänkin ylpeä, yksi niistä on esitelty tässä. Suuri kiitos Aleksi Kaupille ja Viktor Gregurekille loistavasta johtajuudesta, jossa tekijöihin luotetaan ja annetaan tilaa kehittää hyvistä erinomainen.

Haluan kiittää jokaista läheistä, ystävää, opiskelu- ja työkaveria, teillä jokaisella on ollut oma osanne tämän työn onnistumiselle. Erityisen iso kiitos Leo Lehtiselle sekä Mikko Haikulle tuesta niin opinnoissa kuin elämässä.

Vihdissä 5. kesäkuuta 2025,  
Sami Hellsten

# SISÄLLYS

<b>1</b>	<b>JOHDANTO.....</b>	<b>1</b>
<b>1.1</b>	<b>TYÖN TAUSTA.....</b>	<b>1</b>
<b>1.2</b>	<b>TYÖN TAVOITTEET.....</b>	<b>2</b>
<b>1.3</b>	<b>TUTKIMUSMENETELMÄ.....</b>	<b>2</b>
<b>1.4</b>	<b>TYÖN RAKENNE JA RAJAUKSET.....</b>	<b>3</b>
<b>2</b>	<b>TEOREETTINEN TAUSTA.....</b>	<b>5</b>
<b>2.1</b>	<b>SISÄLTÖ DRUPALISSA.....</b>	<b>5</b>
<b>2.2</b>	<b>TIETUEET DRUPALISSA.....</b>	<b>7</b>
<b>2.2.1</b>	<b>TAVALLINEN KENTTÄ (BASIC FIELD).....</b>	<b>7</b>
<b>2.2.2</b>	<b>PERUSKENTTÄ (BASE FIELD).....</b>	<b>9</b>
<b>2.2.3</b>	<b>ERIKOISKENTÄT.....</b>	<b>12</b>
<b>3</b>	<b>TESTITAPAUKSEN LUONTI.....</b>	<b>14</b>
<b>3.1</b>	<b>LUOKKARAKENNE.....</b>	<b>14</b>
<b>3.1.1</b>	<b>ENSIMMÄINEN TIETOMALLI.....</b>	<b>15</b>
<b>3.1.2</b>	<b>TOINEN TIETOMALLI.....</b>	<b>15</b>
<b>3.1.3</b>	<b>KOLMAS TIETOMALLI.....</b>	<b>16</b>
<b>3.2</b>	<b>DATAN VIENTIMALLI.....</b>	<b>17</b>
<b>3.3</b>	<b>TESTIRAKENNE.....</b>	<b>18</b>
<b>4</b>	<b>TULOKSET JA NIIDEN ARVIOINTI.....</b>	<b>19</b>
<b>4.1</b>	<b>DATAN LUONTI.....</b>	<b>19</b>
<b>4.2</b>	<b>SUORA TIETOKANTAYHTEYS.....</b>	<b>20</b>

---

<b>4.3</b>	<b>VIEWS DATA EXPORT -MODUULI.....</b>	<b>21</b>
<b>4.4</b>	<b>ENTITY STORAGE -LUOKKA.....</b>	<b>22</b>
<b>4.5</b>	<b>OPTIMOITU LATAUSLUOKKA.....</b>	<b>23</b>
<b>5</b>	<b>YHTEENVETO.....</b>	<b>25</b>
<b>5.1</b>	<b>JOHTOPÄÄTÖKSET.....</b>	<b>25</b>
<b>5.2</b>	<b>JATKOKEHITYSEHDOTUKSET.....</b>	<b>26</b>
<b>VIITTEET</b>	<b>.....</b>	<b>28</b>
<b>LIITE A:</b>	<b>DRUPAL-DATAN LATAUKSEEN OPTIMOITU LATAUSLUOKKA.</b>	<b>30</b>

# LYHENTEET JA MERKINNÄT

**API** Rajapinta, (engl. Application programming interface)

**CRUD** Lyhenne sanoista Create, Read, Update, Delete

**Entiteettimalli** Toteutus jostain tietomallista

**Tietomalli** Pseudo-versio entiteettimallista

# KUVALUETTELO

<b>Kuva 2.1.</b> Esimerkki entiteetin luontilomakkeesta (Drupal.org).....	6
<b>Kuva 2.2.</b> Tavallisen kentän lisäys entiteettityypille (Drupal.org).....	8
<b>Kuva 2.3.</b> Tavallinen kenttä tietokannassa.....	9
<b>Kuva 2.4.</b> Peruskenttä tietokannassa.....	11
<b>Kuva 2.5.</b> Virtuaalinen kenttä hidastaa latausaikoja.....	12
<b>Kuva 3.1.</b> Yleinen luokkarakenne.....	14
<b>Kuva 3.2.</b> Ensimmäisen tietomallin luokkarakenne.....	15
<b>Kuva 3.3.</b> Toisen tietomallin luokkarakenne.....	16
<b>Kuva 3.4.</b> Kolmannen tietomallin luokkarakenne.....	17

# TAULUKKOLUETTELO

<b>Taulukko 1.1.</b> Design science tutkimuksen suuntaviivat (Hevner ym. 2004).....	3
<b>Taulukko 3.1.</b> Poimittavat tietueet.....	18
<b>Taulukko 3.2.</b> Datan vientimalli.....	18
<b>Taulukko 4.1.</b> Datan luontiajat eri tietomalleilla.....	19
<b>Taulukko 4.2.</b> Datan latausajat suoralla tietokantayhteydellä.....	20
<b>Taulukko 4.3.</b> Datan latausajat VDE-moduulilla.....	21
<b>Taulukko 4.4.</b> Datan latausajat entity storage -luokalla.....	22
<b>Taulukko 4.5.</b> Datan latausajat optimoidulla latausluokalla.....	23

# **OHJELMA- JA ALGORITMILUETTELO**

<b>Listaus 2.1.</b> Peruskentän määrittely koodissa.....	10
--	----

# TEKOÄLYN KÄYTTÖ TÄSSÄ TYÖSSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

- Kyllä
- Ei

## RISKIEN TIEDOSTAMINEN

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joiden tuottamisessa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista tästä seuranneista eettisten ohjeiden rikkomuksista.

# 1 JOHDANTO

Tämä työ perustuu vuonna 2017 alkunsa saaneeseen Siilin asiakasprojektiin, joka on ollut aktiivisessa kehityksessä siitä lähtien. Aktiivisimpina vuosina projektin parissa työskenteli useita kymmeniä henkilöitä yhtäaikaisesti. Allekirjoittanut itse oli reilut neljä vuotta mukana projektin kehityksessä.

Työn taustalla on tarve käyttää projektin muodostamaa dataa asiakasyrityksen raportoinnissa yhdessä muiden datalähteiden kanssa. Projektin käyttöasteen lisääntyessä ja uusien ominaisuuksien myötä datan määrä lähti jyrkkään nousuun jolloin myös datan vienti tarvitsi uudenlaisia ratkaisuja pysyäkseen mukana.

Suuret datamäärät luovat monenlaisia haasteita esimerkiksi latausaikojen, palvelinkuormituksen, integraatioiden ylläpidon kuin myös taustajärjestelmien erikoisosaamisen kanssa. Erityisesti sellaisissa tapauksissa joissa skaalautuvuutta ei ole huomioitu tarpeeksi varhaisessa vaiheessa.

## 1.1 TYÖN TAUSTA

Asiakasprojekti on luotu avoimen lähdekoodin Drupal-sisällönhallintajärjestelmän päälle. Projekti on tarkoitettu lähtökohtaisesti asiakasyrityksen ja sen omien asiakkaiden väliseen kanssakäymiseen, jossa asiakasyritys toimii hallinnoivina pääkäyttäjinä ja asiakkaan asiakkaat sisältöä tuottavina loppukäyttäjinä.

Drupalin lähtöajatus on yksinkertaisesti hallittavassa sivustossa, joka kykenee mukautumaan myös monimutkaisiin tarpeisiin ilman ongelmia [1]. Usein Drupal-sivustoihin voikin törmätä sellaisissa paikoissa, joissa on paljon erityyppistä sisältöä ja mahdollisesti useita eritasoisia käyttäjiä, joilla on pääsy eri toimintoihin. Hyvin perinteisenä esimerkkinä Drupal-sivustona voidaankin käyttää julkista, alun perin Turun kaupungille kehitettyä sivustoa [2], jossa sivuston pääidea on tukea isoa kävijämäärää, jotka käyvät vain luke-massa satunnaisia artikkeleita, sekä sivuston avainkäyttäjiä, jotka hallinnoivat sisältöä. Tällaisella sivustolla entiteetit ovat verrattain kevyitä, eikä niillä ole välttämättä kytköksiä muihin entiteetteihin lainkaan.

Mukautuvuutensa vuoksi Drupal onkin suosittu järjestelmä myös monimutkaisille sivuille sekä esimerkiksi intraneteille, joista esimerkkinä Tampere3-hankkeen toteutus [3]. Tällaisissa tapauksissa sivuston ei välttämättä ole tarkoitus palvella niinkään suurta satunnaiskävijämäärää vaan tiettyjä pääkäyttäjryhmiä, kuten opiskelijoita ja opetushenkilökuntaa.

Sisältö asiakasprojektissa on laaja-alaista, mutta sen keskiössä on puumainen entiteettirakenne, jossa kukin loppukäyttäjryitys muodostaa oman puun ja lähes kaikki sisältö

on yhdistettävissä yksittäiseen puuhun. Tässä työssä on käytetty pelkistettyä rakennetta alkuperäisestä rakenteesta, täyttäen kuitenkin olennaisen kompleksisuuden, eli mittauksissa käytetty entiteettirakenne voitaisiin esittää puuna (parent), sen oksina (child) sekä oksien lehtinä (grandchild).

Raporttien luominen tapahtuu omassa projektissaan, joka kerää tietoja myös muista asiakasyrityksen sisäisistä lähteistä. Drupal-projektin alkuvaiheissa tietorakenne oli kohdalaisen yksinkertainen ja yhtenevä, joten myös datan keruu tapahtui suoraan Drupal-projektin tietokannasta raporteista vastaavan tiimin toimesta. Drupal-projektin kehittyessä ja uusien ominaisuuksien myötä myös tietorakenne muuttui monimutkaisemmaksi, jonka myötä tietokanta paisui useamman tuhannen taulun kokoiseksi. Tietokannan kasvaessa tietokantakyselyjen ylläpito muuttui nopeasti vaikeasti hallittavaksi ja ilman vahvaa Drupal-osaamista liki mahdottomaksi.

Tietokantakyselyjen korvaaminen kestäväällä ratkaisulla päätyi allekirjoittaneen ratkaistavaksi. Tämä työ käy läpi useamman vuoden kestäneen kehitysprosessin oleellisimpia vaiheita, eri ratkaisutapoja sekä niissä havaittuja rajoitteita ja ongelmakohtia.

## 1.2 TYÖN TAVOITTEET

Tämän työn tavoitteena on löytää ratkaisu siihen, miten Drupal-järjestelmän sisältämä data saadaan ladattua joko sellaisenaan ulkoisiin järjestelmiin jatkokäsittelyä varten rajapintaa hyödyntäen tai suoraan Drupal-sivustolta ladataan raportointitarkoituksiin. Työn tavoitteena on siis vastata seuraaviin tutkimuskysymyksiin:

1. Miten Drupal-järjestelmän data viedään ulkoisiin järjestelmiin skaalautuvasti?
2. Miten entiteetin toteutustapa vaikuttaa datan käsittelynopeuteen Drupal-järjestelmässä?

Ensimmäisessä tutkimuskysymyksessä keskitytään erityisesti itse datan muodostamiseen vientiä varten riippumatta siitä miten lopullinen vienti tapahtuu. Tavoitteena on löytää ratkaisu mikä toimii myös suurilla datamäärillä, ilman että odotusajat kasvavat sietämättömiksi. Lisäksi ratkaisun keskeisenä osana on sen ylläpidettävyys siten, että tietorakennetta voidaan edelleen muokata ilman erityisosaamista dataintegraatioista. Toisessa tutkimuskysymyksessä vertaillaan Drupalin kahta eri perustapaa luoda kenttiä entiteeteille sekä kahta eri tapaa muodostaa relaatioita. Vertailun tavoitteena on selvittää vaikuttaako entiteetin kenttien rakennustapa latausnopeuksiin merkittävästi ja luoda ajatus siitä kannattaako jo olemassa olevia tietorakenteita pyrkiä muuttamaan.

## 1.3 TUTKIMUSMENETELMÄ

Tässä työssä tutkimuskysymyksiin haetaan vastausta tietojärjestelmien Design science -tutkimusmenetelmällä [4], joka Hevnerin ynnä muiden (2004) mukaan pyrkii kehittämään

tietoa, jota alan ammattilaiset voivat hyödyntää ja käyttää hyväksi suunnitellessaan ratkaisuja alan ongelmiinsa. Tutkimukselle on annettu seitsemän suuntaviivaa.

**Taulukko 1.1.** Design science tutkimuksen suuntaviivat (Hevner ym. 2004).

Guideline	Description
1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Tutkimuksessa luodaan vaihtoehtoisia ratkaisuja, joita testataan annetuilla vaatimuksilla tai rajoitteilla. Tässä työssä määritteiksi on valittu suoraan mitattavissa oleva suoritus aika sekä epäsuorasti kustannustehokkuuteen sitoutuva ylläpidettävyyys ilman erikoisosamista. Työssä on lisäksi suppea kirjallisuuskatsaus kattaen oleellimmat seikat eri ratkaisuvaihtoehtojen taustalla.

## 1.4 TYÖN RAKENNE JA RAJAUKSET

Työn teoriaosassa käydään läpi merkittävimmät määritteet erityisesti toiseen tutkimuskysymykseen liittyen, siinä käydään läpi Drupal-järjestelmän tapa säilöä tietoa eri tapauksissa. Teoriaosa perustuu vakiintuneisiin käytänteisiin, eikä niillä välttämättä ole tieteellistä näyttöä.

---

Testien luontiosiossa luodaan yksinkertaistettu tietomalli, joka kuvastaa tyypillisiä relatioita eri entiteettien välillä Drupal-järjestelmässä sekä kolme eri toteutusmallia jotka vastaavat yleisimpiä käytettyjä tapoja toteuttaa entiteettejä. Datalle luodaan vientimalli, joka simuloi perinteistä raporttia, jossa on poimintoja tietomallin eri entiteeteistä. Lisäksi luodaan rakenne, jolla eri toteutusmallien suorituskykyä voidaan mitata sekä vertailla.

Tuloksissa käydään läpi simulaatiomallin testituloksia sekä arvioidaan niiden pohjalta sellaisia käyttötapauksia, joissa kyseinen malli on toimiva ratkaisu ja missä tapauksissa vastaavasti malli ei ole riittävä tai siinä on huomioitavia puutteita. Tuloksissa käydään myös läpi toteutusmallin vaikutus suorituskykyyn. Tulokset vastaavat molempiin tutkimuskysymyksiin.

Työn loppuksi on yhteenveto, jossa pohditaan havaintojen merkitystä eri kokoisille Drupal-toteutuksille ja käydään läpi työn aikana ilmenneitä seikkoja. Osio sisältää myös ehdotukset jatkokehittämisestä.

## 2 TEOREETTINEN TAUSTA

Drupal on avoimen lähdekoodin sisällönhallintajärjestelmä (engl. CMS), jossa sisältöä hallitaan tyypillisesti verkkoselaimesta käsin käyttäjien toimesta. [5] Drupalin isoimpiin vahvuuksiin kuuluu muun muassa sen modulaarinen rakenne, joka mahdollistaa sen käyttämisen eri käyttötarkoituksiin. Erityisesti useamman käyttötarkoituksen sivustoille Drupal on usein helpompi ratkaisu kuin tiettyihin käyttötarkoituksiin räätälöityjen järjestelmien muokkaaminen tai yhdistely.

Yleisimpänä pohjana Drupalia käytetään Linux-käyttöjärjestelmän, Apache-verkkopalvelinta, MySQL-tietokantaa sekä PHP-ohjelmointikieltä, eli niin kutsuttua LAMP-pinoa. [6] Ohjelmointikieltä lukuun ottamatta pino on kuitenkin lähes vapaasti muokattavissa omiin käyttötarkoituksiin sopivaksi, yleisimpiin vaihtoehtoihin Drupal on myös suoraan yhteensopiva.

Tämän työ on toteutettu LAMP-pinoa hyödyntäen, painottuen ohjelmointikielen ja tietokannan väliseen tiedonsiirtonopeuteen. Lukijalla oletetaan olevan perustason ymmärrys kyseisistä komponenteista.

### 2.1 SISÄLTÖ DRUPALISSA

Drupal-järjestelmässä esiintyvät asiat ovat pääasiallisesti entiteettejä (engl. entity), nämä jakautuvat kahteen eri tyyppiin; konfigurointientiteetti (engl. configuration entity) ja sisältöentiteetti (engl. content entity) [7]. Konfigurointientiteetti määrittää esimerkiksi käyttäjäoikeuksia tai jonkin osion sijaintia sivustolla. Konfigurointientiteetit tallennetaan tyypillisesti erillisiin tiedostoihin, jolloin kyseiset asetukset voidaan siirtää järjestelmästä toiseen ja näin ollen saadaan esimerkiksi testausympäristö toimimaan samoilla säännöillä kuin tuotantoympäristö. Sisältöentiteetit puolestaan ovat tyypillisesti vain yhden ympäristön sisältämää tietoa, kuten uutisartikkeleita tai käyttäjiä, jotka tallennetaan tietokantaan.

Entiteettejä käsitellään tavallisesti oman rajapintansa (Entity API) avulla CRUD-operaatioita hyödyntäen esimerkiksi sivuston käyttöliittymässä olevilla lomakkeilla ja näkymillä (Kuva 2.1). Rajapinta pitää huolen muun muassa tarvittavista käyttöoikeuksista kullekin operaatiolle. Myös sellaisissa tapauksissa, joissa entiteettejä käsitellään ohjelmakoodissa suoraan ohittaen entiteettirajapinnan käytetään tyypillisesti jotain muuta Drupalin oletuksena tarjoamaa rajapintaa, kuten tietokantarajapintaa (database API) [8]. Tietokantarajapintaa ei yleisesti suositella käyttämään, mutta se tarjoaa lähes suoran pääsyn tietokantaan mahdollistaen esimerkiksi massapäivitykset sellaisissa tilanteissa missä entiteettien normaali luku, päivitys ja tallennus olisi liian hidasta. Suorilla muokkauksilla

ohitetaan kuitenkin samalla myös muun muassa välimuisti, joten siinä on myös omat riskinsä.

The screenshot shows the 'Create Article' form in Drupal 10. The form is divided into several sections:

- Title:** A text field containing 'Managing content'.
- Image:** A section for uploading an image. It shows a thumbnail of 'administer-01-front-page.png' (330.85 KB) with a 'Remove' button. Below the image is an 'Alternative text' field containing 'Managing content'.
- Body:** A rich text editor with a toolbar. The main text reads:
 

**Adding content**

This guide assumes that you have used the standard profile when installing, which defines the two content types *Basic Page* and *Article*. (If you used the minimal profile, you need to [define content types](#) first.)

**Before we begin:** Make sure you are logged in as a user who has the right to create content – ask your system administrator if you are not sure; otherwise, some of the fields you need to select will not be visible.

  1. Select *Add content* from the front page ( either in the navigation menu or from the administrator menu ).
  2. This brings up a screen asking what type of content you wish to add. By default, Drupal 8 and higher comes with two content types *Article* and *Basic Page*.
  3. Select *Article*. The only major difference between *Basic Page* and *Article* is that you have the ability to upload an image when creating an *Article*; a *Basic Page* is a static page. You can update the fields in the content type later.
  4. A form will appear allowing you to enter information for your *Article*. You can also configure the revision log, menu, comment, authoring and promotion option settings. These settings vary as you [extend your Drupal instance](#) and [use the contributed modules](#).
  5. Enter a *Title* for the page in the *Title* text box, for example, *Test Article*.
  6. In the *Summary* area enter some text that describes your article briefly.
- Text format:** A dropdown menu set to 'Basic HTML'.
- Tags:** A text field containing 'Drupal, Drupal 10'.
- Published:** A checked checkbox.
- Buttons:** 'Save' and 'Preview' buttons.
- Settings sidebar:** A vertical sidebar on the right with sections for:
  - Last saved:** Not saved yet
  - Author:** admin
  - Revision log message:** A text area for describing changes.
  - Menu settings:** Not in menu
  - Comment settings:** (empty)
  - URL alias:** No alias
  - Authoring information:** By admin (1) on 2024-07-01
  - Promotion options:** Promoted to front page

### Kuva 2.1. Esimerkki entiteetin luontilomakkeesta (Drupal.org).

Lähtökohtaisesti yksittäisen entiteettityypin sisältö on säilötty tietokannassa yhteen pohjatauluun (engl. base table), mutta eri tavoin luodut tietueet saattavat muodostaa erillisiä tietokantatauluja. Tyypillisestä relaatiotietokantamallista [9] poiketen Drupal ei muodosta tietokantaan viittauksia eri tietueiden välille vaan ne ladataan luetaan id-tunnisteiden avulla. Myöskään tapauksissa missä entiteetillä on referenssejä muihin entiteetteihin

tietokanta ei oletuksena pidä yllä viittauksia näiden tietokantarivien välillä. Tämän takia on mahdollista, että entiteetti viittaa johonkin toiseen entiteettiin jota ei enää ole olemassa tai vaihtoehtoisesti tietokantaan jää sellaisia entiteettejä joita ei enää käytetä missään [10]. Normaalissa käytössä rikkiäiset viittaukset tai ylimääräiset entiteetit eivät aiheuta ongelmia, mutta ne kuitenkin vievät turhaa tilaa ja saattavat hidastaa tietokantakyselyjä, mikäli tapauksia on merkittävästi.

## 2.2 TIETUEET DRUPALISSA

Käytännössä kaikille sisältöentiteeteille voidaan lisätä tarvittaessa uusia tietueita, eli kenttiä (engl. field) joilla entiteetti saadaan palvelemaan omaa käyttötarkoitusta [11]. Kenttätyyppiä on useita erilaisia sisältäen perustyyppit, kuten numeeriset (decimal, float jne.), tekstipohjaiset (string, text jne.), sekä erikoisemmat sisällöt kuten kuva, tiedosto tai viittaus toiseen entiteettiin [12].

Kenttätyyppi vastaa siitä, miten tietue käytännössä säilötään tietokantaan ja miten se yhdistetään sieltä kullekin entiteetille. Käyttäjän tavasta syöttää kyseinen tieto sen sijaan vastaa kyseiselle kenttätypille määritelty widgetti (FieldWidget) ja vastaavasti tiedon esittämisestä vastaa formaatti (FieldFormat). Tämän työn osalta ainoastaan tietokannan sisällöllä on merkitystä, sillä tietueet tullaan esittämään sellaisenaan.

Oletuksena Drupal tarjoaa Node-nimisen entiteettityypin, joka on hyvin avoin ylämääritelmä sisällölle. Tämä ylämääritelmä jakautuu erinäisiin tarkentaviin alatyyppeihin (engl. bundle), kuten esimerkiksi uutisartikkeliin. Alatyypitys ei ole pakollista, vaan jotkin sisältötyypit, kuten käyttäjätyyppi (User) sisältää pelkän ylämääritelmän. Teknisesti näissä tapauksissa kuitenkin kullakin entiteetillä on myös alamääriyksenä sama tyyppinimi kuin ylämääriyksenä.

### 2.2.1 TAVALLINEN KENTTÄ (BASIC FIELD)

Tavallinen kenttä on käyttöliittymästä (Kuva 2.2) käsin luotu lisäkenttä entiteettityypille [11]. Field API [13] mahdollistaa kenttien lisäämisen kaikille entiteettityypeille. Luonnin yhteydessä Field API muodostaa kentälle oman tietokantataulunsa. Samaa kenttää voidaan uudelleen käyttää toisessa saman entiteettityypin alatyypissä, tällöin kaikkien alatyypien käyttämä tieto kyseisestä kentästä löytyy samasta taulusta. Eri entiteettityypit eivät käytä samaa tietokantataulua, vaan mikäli samanlainen kenttä luotaisiin toiselle entiteettityypille, sille muodostuisi oma tietokantataulunsa.

Home > Administration > Structure > Content types > Vendor > Manage fields

## Add field ☆

Choose a type of field \*

<b>T</b> Plain text Text field that does not support markup.	<b>T≡</b> Formatted text Text field with markup support and optional editor.	<b>123</b> Number Field to store number. I.e. id, price, or quantity.
<b>📁</b> Reference Field to reference other content.	<b>📄</b> File upload Field to upload any type of files.	<b>☰</b> Selection list Field to select from predefined options.
<b>📅</b> Date and time Field to store date and time values.	<b>🔘</b> Boolean Field to store a true or false value.	<b>💬</b> Comments This field manages configuration and presentation of comments on an entity.
<b>@</b> Email Field to store an email address.	<b>🔗</b> Link Stores a URL string, optional varchar link text, and optional blob of attributes to assemble a link.	

[Continue](#)

**Kuva 2.2.** Tavallisen kentän lisäys entiteettityypille (Drupal.org).

Kuvassa Kuva 2.3 on hahmotelma siitä miten `longtext` tyyppinen kenttä esiintyy tietokannassa. Tilanteessa entiteettityypin `first_child` perustiedot ovat omassa tietokantataulussaan, ja kenttä `field_text_a` toisessa. Kuvasta nähdään myös, kuinka yksittäistä tavallista kenttää varten tietokantaan todellisuudessa tallennetaan useita eri tietueita, joita käytetään kentän yhdistämiseen sen omistavaan entiteettiin. Varsinainen oleellinen tieto tallentuu tässä tapauksessa kentän tietokantatauluun `field_text_a_value` sarakkeeseen.

The image shows a screenshot of a database schema with two tables. The first table, `first_child`, is described as '/\* The base table for first\_child entities. \*/' and contains fields: `uuid` (varchar(128)), `label` (varchar(255)), `status` (tinyint(4)), `created` (int(11)), `changed` (int(11)), and `id` (int(10) unsigned). The second table, `first_child_field_text_a`, is described as '/\* Data storage for first\_child field field\_text\_a. \*/' and contains fields: `bundle` (varchar(128)), `revision_id` (int(10) unsigned), `field_text_a_value` (longtext), `deleted` (tinyint(4)), `entity_id` (int(10) unsigned), `langcode` (varchar(32)), and `delta` (int(10) unsigned).

Table Name	Description	Field Name	Field Type
first_child	/* The base table for first_child entities. */	uuid	varchar(128)
		label	varchar(255)
		status	tinyint(4)
		created	int(11)
		changed	int(11)
		id	int(10) unsigned
first_child_field_text_a	/* Data storage for first_child field field_text_a. */	bundle	varchar(128)
		revision_id	int(10) unsigned
		field_text_a_value	longtext
		deleted	tinyint(4)
		entity_id	int(10) unsigned
		langcode	varchar(32)
		delta	int(10) unsigned

**Kuva 2.3.** Tavallinen kenttä tietokannassa.

Käyttöliittymän kautta kenttiä on helppo luoda ja muokata etenkin luontivaiheessa omaan käyttötarkoitukseen sopivaksi. Kentistä muodostuu erilliset konfiguraatiotiedostot joilla ne on mahdollista synkronoida muiden ympäristöjen kanssa eikä ylimääräisiä välivaiheita tarvita silloin kun kenttiä lisätään tai poistetaan. Käyttöön otetun kentän muokkaaminen sen sijaan on hieman monimutkaisempaa, Drupal estää joidenkin asetusten muuttamisen kokonaan siinä vaiheessa, kun kyseiseen kenttään on tallennettu jotain dataa ja vaikka paikallisessa ympäristössä poistaisi kyseiset tietueet, jonka jälkeen kenttää olisi mahdollista muokata, on huomioitava, että synkronointivaiheessa muihin ympäristöihin Drupal todellisuudessa ensin poistaa kentän kaikkine tietoineen ja lisää sen jälkeen kentän uudelleen. Muokatessa kenttää onkin syytä miettiä, tarvitseeko kyseisiä tietoja säilyttää ja tarvittaessa luoda samassa yhteydessä päivityspolku (engl. update hook) [14] joka päivittää tiedot vanhasta kentästä uuteen kenttään.

### 2.2.2 PERUSKENTTÄ (BASE FIELD)

Peruskenttä on ohjelmakoodissa määritelty (Listaus 2.1) kenttä entiteettityypissä [15]. Tavallisesta kentästä poiketen peruskentille ei lähtökohtaisesti muodostu omia tietokantatauluja, vaan kaikki yhden entiteettityypin peruskentät muodostavat yhden tietokanta-

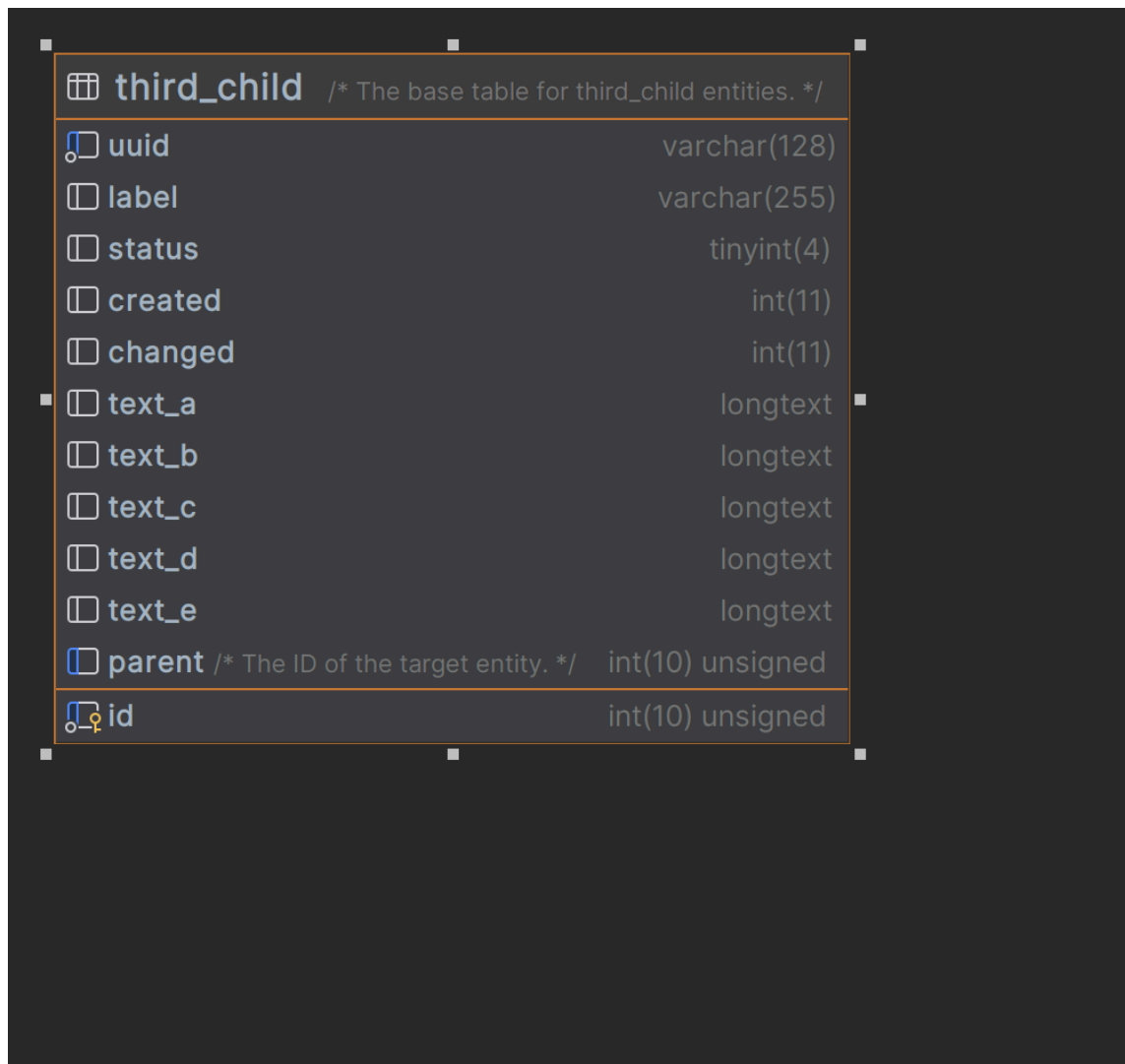
taulun. Peruskenttä on aina jokaisella saman entiteettityypin alatyypityksistä, mutta mikäli sitä ei tarvita, se voidaan piilottaa käyttöliittymistä.

```
1 <?php
2 // ...
3 final class ThirdChild
4     extends ContentEntityBase
5     implements ThirdChildInterface {
6     // ...
7     public static function baseFieldDefinitions(
8         EntityTypeInterface $entity_type
9     ): array {
10        // ...
11        $fields['text_a'] =
12            BaseFieldDefinition::create('string_long')
13                →setLabel(t('Text A'))
14                →setRequired(FALSE)
15                →setDisplayOptions('form', [
16                    'type' ⇒ 'string_textarea',
17                    'weight' ⇒ 0,
18                    'settings' ⇒ [
19                        'rows' ⇒ 12,
20                    ],
21                ])
22                →setDisplayConfigurable('form', TRUE)
23                →setDisplayOptions('view', [
24                    'type' ⇒ 'string',
25                    'weight' ⇒ 0,
26                    'label' ⇒ 'above',
27                ])
28                →setDisplayConfigurable('view', TRUE);
29        // ...
30        return $fields;
31    }
32    // ...
33 }
```

### Listaus 2.1. Peruskentän määrittäminen koodissa.

Kuvassa Kuva 2.4 on hahmotelma siitä, miten `longtext` tyyppinen kenttä esiin-tyy tietokannassa kun se on luotu peruskenttänä. Tilanteessa kaikki entiteettityypin `third_child` kentät ovat luotu peruskenttinä, joten myös `text_a` kenttä nähdään samaisessa taulussa, eikä tietojen yhdistämiseen tarvita erillisiä yhdistämistietueita (vrt.

Kuva 2.3). Peruskentän oleellinen tieto on siis tällä kertaa suoraan saatavilla muiden perustietojen lomassa `text_a` sarakkeesta.



```
third_child /* The base table for third_child entities. */
  uuid      varchar(128)
  label     varchar(255)
  status    tinyint(4)
  created   int(11)
  changed   int(11)
  text_a    longtext
  text_b    longtext
  text_c    longtext
  text_d    longtext
  text_e    longtext
  parent /* The ID of the target entity. */ int(10) unsigned
  id       int(10) unsigned
```

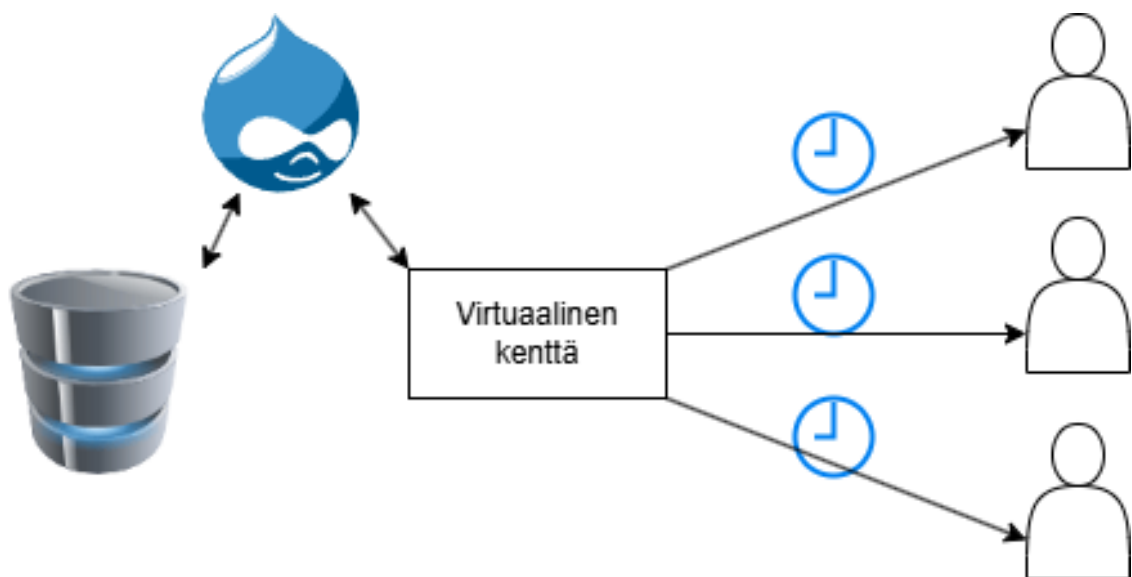
**Kuva 2.4.** Peruskenttä tietokannassa.

Peruskenttien luonti itsessään on varsin suoraviivaista, ohjelmakoodissa määritellään kyseisen kentän asetukset ja määritteet, joista suurin osa on vapaavalintaisia. Peruskentät ovat kuitenkin hieman mutkikkaampia ylläpitää kuin tavalliset kentät, sillä Drupal ei automaattisesti poimi muutoksia peruskenttämäärittelyyn, jos kyseinen entiteettityyppi on jo kertaalleen rekisteröity kyseiseen ympäristöön. Tämän takia kaikki muutokset peruskenttiin, mukaan lukien lisäykset ja poistot tarvitsevat päivityspolun [16] joka pitää huolen, että olemassa oleva tietokanta päivittyy samanlaiseksi kuin se muodostuisi uudessakin ympäristössä. Päivityspolku on usein vaikeampi määritellä kuin varsinainen peruskenttä, joten omat kenttatarpeet kannattaa suunnitella huolellisesti ennen ensimmäistä käyttöön-ottoa.

### 2.2.3 ERIKOISKENTÄT

Edellä mainittujen lisäksi on joitain poikkeuksia missä kentät eivät ole tietokannassa joko suoraan peruskenttätaulussa tai entiteettityypin mukaan nimetyissä lisätauluissa. Tällaisissa tapauksissa tieto näkyy käyttöliittymässä ja entity API:n [7] kautta ohjelmakoodissa käytettynä kuten muutkin tietueet, mutta niiden ulkopuolelta tiedon löytäminen vaatii hieman syvällisempää perehtyneisyyttä Drupaliin.

Virtuaalisten kenttien tapauksessa tietokantaan ei tallenneta mitään, vaan kentän arvo muodostetaan aina sitä kysyttäessä [17]. Näitä saatetaan käyttää esimerkiksi sellaisissa tilanteissa missä tietoa tarvitaan vain satunnaisissa tapauksissa tai tieto johdetaan ulkoisesta lähteestä. Toistuvalla latauksella on kuitenkin suotavampaa käyttää dynaamisia kenttiä, joihin sama tieto tallennetaan aina entiteetin tallennuksen yhteydessä, tällöin tieto on saatavilla suoraan tietokannasta mahdollistaen nopean prosessoinnin (Kuva 2.5). Huomionarvoista kuitenkin on, että mikäli tieto saattaa ulkoisessa lähteessä muuttua epäsäännöllisesti, saattaa virtuaalinen kenttä olla ainoa reaaliaikainen ratkaisu tiedon esittämistä varten. Virtuaalisten kenttien korvaaminen oli oleellinen osa tämän työn taustalla olleessa projektissa, jotta vientiaikoja oli mahdollista parantaa.



**Kuva 2.5.** Virtuaalinen kenttä hidastaa latausaikoja.

Eräs yleisesti käytetty erikoiskenttä on moderointitila (engl. moderation state), joka mahdollistaa entiteetin julkaisun rytmittämisen [18]. Moderointitila ohjaa entiteetin `status` kenttää, joka oletuksena määrittää onko kyseinen entiteetti julkaistu. Moderointitiloja voidaan määrittää tarpeen mukaan, esimerkiksi jonkin entiteetin tilat voisi olla luonnos, julkaisu ja arkistoitu, joista vain julkaisu olisi yleisesti saatavilla sivustolla sekä vastaavasti luonnos ja arkistoitu näkyisi vain esimerkiksi kyseisen entiteetin tekijälle.

Moderointitila on saatavilla muiden kenttien tapaan käyttöliittymästä sekä entity API:n kautta, mutta taustalla se on todellisuudessa oma entiteettinsä, joka varastoidaan tieto-

---

kannassa omana taulunaan. Ohjelmallisesti moderointitilat ovat myös ladattavissa ilman niihin liitetyjä entiteettejä, mutta käyttöliittymästä ne eivät ole saatavilla. Mahdollisia tiloja ja niiden muutoksia erilliset *workflows* -määritelmät, jotka tallennetaan konfiguraationa, eivätkä ne ole saatavilla tietokannan puolelta.

Tässä työssä erikoiskentät eivät ole mukana tehtävissä testitapauksissa, jotta esimerkit saadaan pidettyä riittävän yksinkertaisina ja helposti ymmärrettävinä sekä mahdollisimman vertailukelpoisina keskenään. Nämä ovat siitä huolimatta kriittisiä tapauksia ottaa huomioon mikäli vastaavaa datan vientimallia ollaan rakentamassa, sillä niiden lisääminen jälkikäteen valmiiseen ratkaisuun voi tehdä koko ratkaisusta käyttökelvottoman.

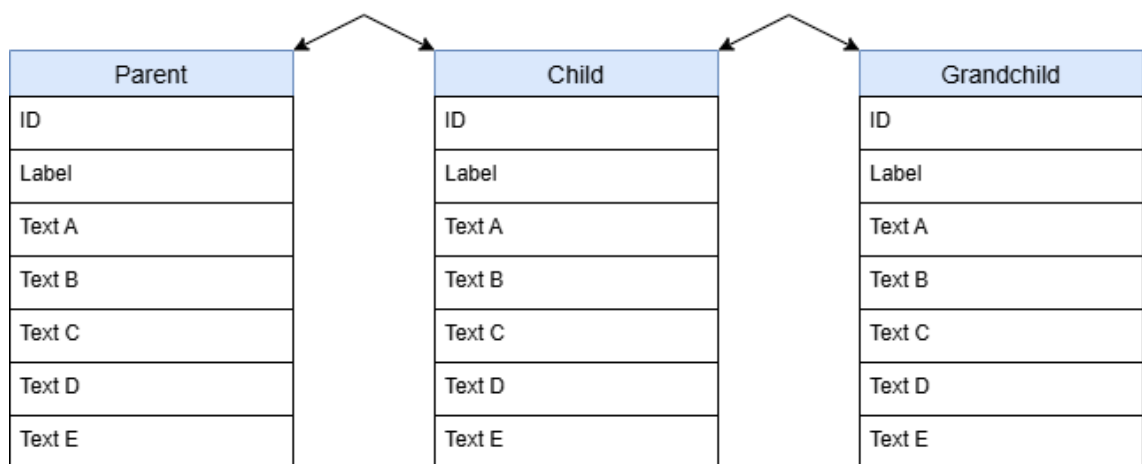
## 3 TESTITAPAUKSEN LUONTI

Työssä rakennettiin testausmalli, jossa ensin luotiin tietomalli, jolla voidaan vastata toiseen tutkimuskysymykseen siitä, vaikuttaako tietomallin luontitapa merkittävästi lopputulokseen. Tietomallin jälkeen luotiin datan vientimalli, millä simuloidaan sitä lopputulosta, joka näkyy järjestelmän ulkopuolelle samanlaisena riippumatta siitä miten se on muodostettu tai ladattu. Lopuksi luotiin testirakenne, jolla mitattiin erikokoisilla tietueilla eri tietomallien nopeutta luoda entiteettejä sekä vientimallien nopeutta ladata haluttu tieto esitettäväksi.

Testausmalli luotiin lähes täysin automaattiseksi, pois lukien mittaukset, joissa on käytetty käyttöliittymän kautta käytettävää datan vientiä. Malli varmistaa jokaisen mittauksen alkavan samasta lähtötilanteesta, jotta esimerkiksi välimuisti ei vääristä mittaustuloksia.

### 3.1 LUOKKARAKENNE

Kaikki tietomallit muodostavat saman yleisen rakenteen kolmesta eri entiteettityypistä. Tietomalli muodostuu kolmesta entiteettityypistä (parent, child, grandchild) jotka ovat linkitetty toisiinsa (Kuva 3.1). Tietomalleissa eroavaisuudet muodostuvat linkityksen tavasta sekä kenttien luontitavasta. Muilta osin entiteettityypit ovat perustyyppisiä revisioimattomia sisältöluokkia [19].

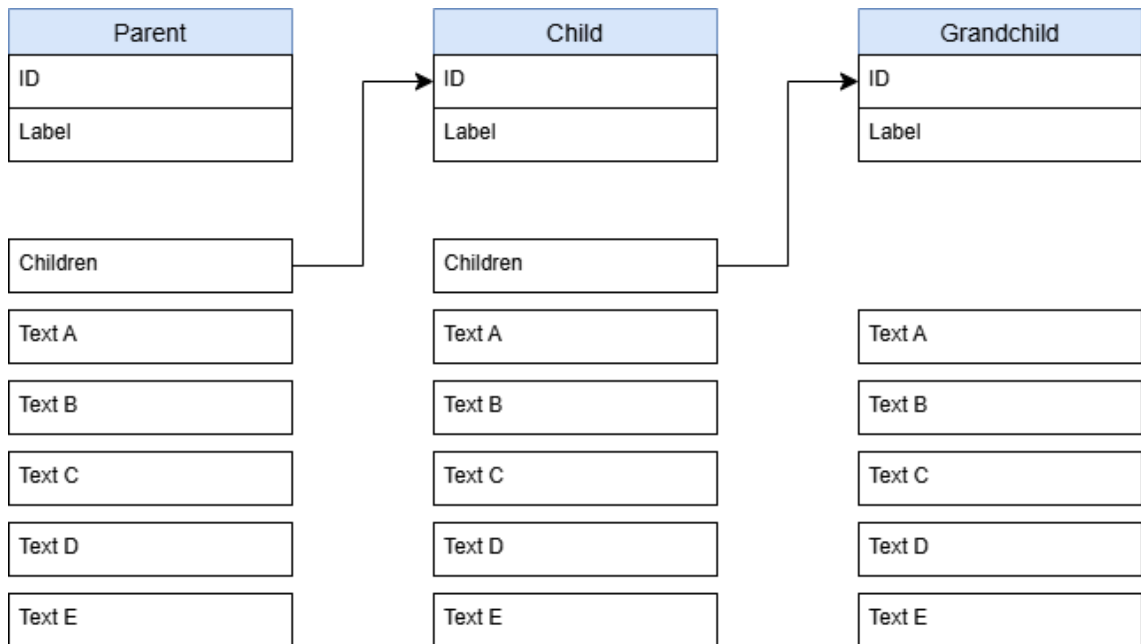


**Kuva 3.1.** Yleinen luokkarakenne.

Työssä käytetyt tietomallit ovat sisällöltään yksinkertaistettuja, kullakin entiteettityypillä on viisi `string_long` -tyyppistä tekstikenttää sekä oletuskentät kuten `id` ja `label`. Yksinkertaisen mallin takia suoritusnopeudet yksittäisen entiteetin kohdalla ovat selvästi nopeampia kuin todellisissa järjestelmissä tyypillisesti käytettyjen entiteettien olisivat. Yksittäisten entiteettien käsittelyaikojen sijaan työssä keskitytään siihen, kuinka pitkälle käytetty menetelmä skaalautuu kasvattamalla entiteettien määrää.

### 3.1.1 ENSIMMÄINEN TIETOMALLI

Ensimmäinen tietomalli toteutettiin käyttämällä tavallisia kenttiä. Kuva 3.2 esittää mallin oleellisimman rakenteen, jossa kullakin entiteettityypillä on oma pohjataulunsa sisältäen vain oletuskentät ja varsinaiset sisältökentät muodostavat kukin oman tietokantataulunsa. Tässä mallissa entiteettien väliset viittaukset on toteutettu siten, että jokaisella entiteetillä on viittaus tämän alapuolella suoraan oleviin entiteetteihin.



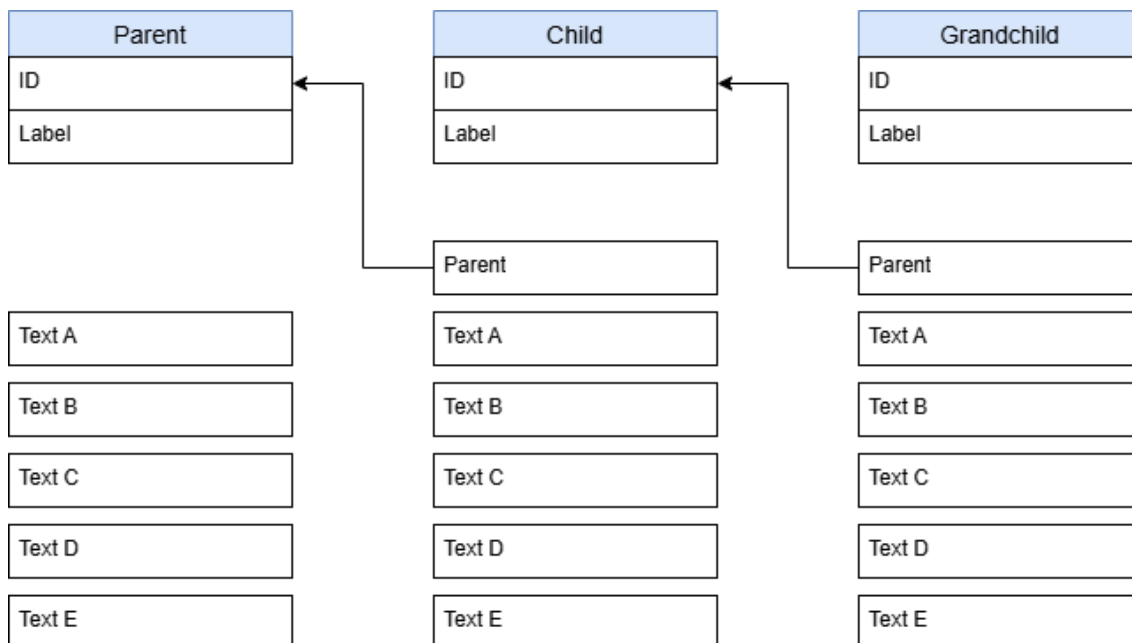
**Kuva 3.2.** Ensimmäisen tietomallin luokkarakenne.

Tämän tietomalli on yleinen Drupal-järjestelmissä erityisesti silloin, kun tietorakenteet ovat luotu Node-entiteettityypin päälle. Osaltaan malli on yleinen myös tilanteissa missä varsinaista tietomallia ei ole suunniteltu lainkaan, johtuen käyttöliittymän helppoudesta hallita entiteettityyppejä.

Kenttien osalta tietomalli soveltuu parhaiten tilanteisiin, joissa entiteetille on useita eri näkymiä, joissa esitetään eri osia samasta entiteetistä, tällöin hajautettujen tietokantataulujen ansiosta järjestelmä ei joudu käsittelemään käyttökohteeseen nähden turhia tietueita. Viittauksien puolesta tietomalli soveltuu parhaiten tilanteisiin, jossa on tarkoituksenmukaista, että aina entiteettiä ladatessa halutaan ladata myös kaikki alapuoliset entiteetit.

### 3.1.2 TOINEN TIETOMALLI

Toinen tietomalli toteutettiin ensimmäisen tietomallin tavoin tavallisia käyttäen. Tietomallin ainoa ero ensimmäiseen malliin on viittausten muodostaminen alhaalta ylöspäin (Kuva 3.3). Tässä mallissa on paremmin huomioitu viittausten muodostaminen [20].



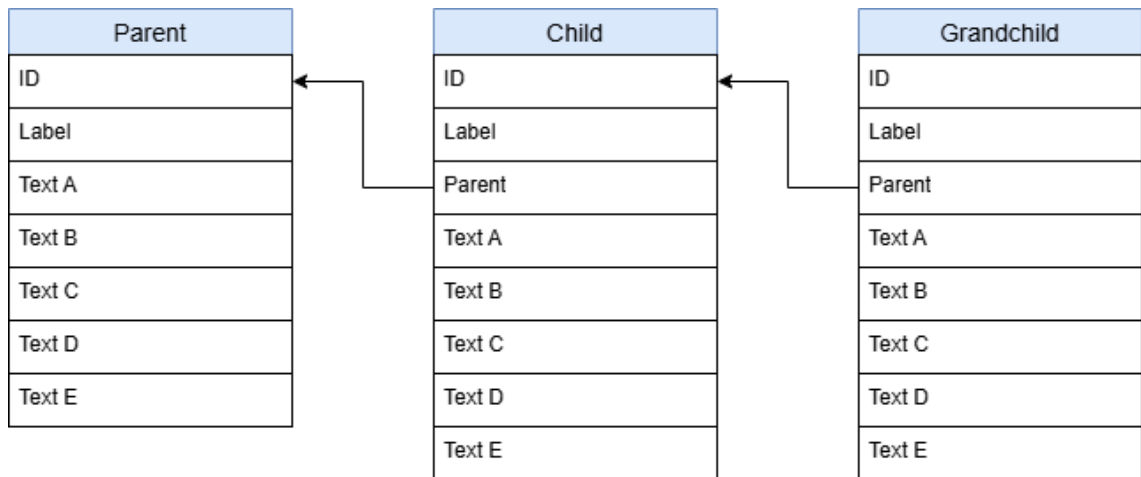
**Kuva 3.3.** Toisen tietomallin luokkarakenne.

Toisin kuin ensimmäisessä tietomallissa, joka käytti 1:N viittauksia, tässä mallissa on käytetty 1:1 viittauksia. Jokainen entiteetti sisältää viittauksen ns. omistajaansa, jolloin kentän koko saadaan vakioitua yhteen, vaikka yhdellä entiteetillä voikin edelleen olla useita alapuolisia entiteettejä.

Tämä malli on Drupal-toteutuksissa hieman harvinaisempi, vaikka se mahdollistaa ylimääräisten latausten minimoinnin, esimerkiksi tässä mallissa yhden entiteetin lataus aiheuttaa yhteensä korkeintaan kolmen entiteetin tietojen latauksen. Malli soveltuukin viittauksien puolesta käytettäväksi lähes kaikissa toteutuksissa. Drupalin käyttöliittymät tukevat molempia viittaustapoja oletuksena.

### 3.1.3 KOLMAS TIETOMALLI

Kolmas tietomalli luotiin aiemmista malleista poiketen käyttäen pelkästään peruskenttiä. Tässä toteutuksessa näin ollen kaikki tieto tallennetaan yhteensä kolmeen eri tietokantatauluun (Kuva 3.4). Toisen tietomallin mukaisesti kolmannessa mallissa käytettiin 1:1 viittauksia alhaalta ylöspäin.



**Kuva 3.4.** Kolmannen tietomallin luokkarakenne.

Peruskentillä luotuna malli on huomattavasti tiiviimpi, eikä tietokannassa tarvitse säilöä ylimääräisiä tietueita kenttien yhdistelyä varten. Tämän mallin idea onkin toimia suorana verrokkina toiseen tietokantamalliin, jotta voidaan vertailla toteutustavan tehokkuutta kahden identtisen tietorakenteen välillä.

Tämä toteutusmalli on yleinen erityisesti Drupalin yhteisömoduuleissa, joissa entiteettimallia on suunniteltu pitkään useiden eri kehittäjien toimesta. Yhteisömoduulit vaativat kuitenkin usein räätälöintiä omaan käyttötarkoitukseen soveltuvaksi, jolloin herkästi lisätään tavallisia kenttiä peruskenttätoteutuksen päälle.

Malli soveltuu käytännössä kaikkiin toteutuksiin erinomaisesti, esimerkiksi alatyypitetyt entiteetit on mahdollista korvata omalla entiteettityypityksellä, käyttäen rajapintaluokkaa (engl. interface) yhdistämään eri entiteettityypit toisiinsa. Todellisuudessa kuitenkin mallin täydellinen toteutus saattaa olla huomattavan monimutkaista lopulliseen hyötyyn nähden.

## 3.2 DATAN VIENTIMALLI

Datan viennistä rakennettiin malli, joka edustaa työn taustalla olevan toteutuksen lähtötilannetta. Lähtötilanteessa entiteettien tiedoista vietiin vain osa ulkoiseen järjestelmään, käytännössä tiettyä käytötapausta tai raporttia varten. Taulukko 3.1 esittää kustakin entiteettityypistä poimittavat tiedot, eli tunnistetiedot ( ID ja Label ) sekä kaksi erikseen valittua kenttää.

**Taulukko 3.1.** Poimittavat tietueet.

Parent (P)	Child (C)	Grandchild (G)
ID (I)	ID (I)	ID (I)
Label (L)	Label (L)	Label (L)
Text A	Text B	Text A
Text C	Text E	Text D

Jotta valitut tiedot voidaan tuoda esimerkiksi CSV-tiedostona, edellä esitetty (Taulukko 3.1) tiedot muodostettiin yksiulotteiseksi riviksi (Taulukko 3.2) jolloin jokainen entiteetti muodostaa oman rivinsä datatulosteeseen. Oleellista tässä artefaktissa on, että saatu tuloste on aina täsmälleen samanlainen samaisella entiteettimäärällä, riippumatta siitä millä tavoin entiteetit ovat luotu.

**Taulukko 3.2.** Datan vientimalli.

P-I	P-L	P-A	P-C	C-I	C-L	C-B	C-E	G-I	G-L	G-A	G-D
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Kuten taulukosta Taulukko 3.2 voidaan havaita, ulostulossa todellisuudessa toistuu yläpuolisten entiteettien tiedot aina jokaista alapuolista entiteettiä kohden. Tämä on tarkoituksenmukainen replikointi todellisista raportointitarpeista, sillä loppuraportilla saatetaan tehdä erinäisiä rajoituksia jotka olisi hankala toteuttaa muussa tapauksessa.

### 3.3 TESTIRAKENNE

Testaus rakennettiin lähes täysin automaattiseksi, ohjelma aloittaa nollaamalla tietokannan ja asentamaan Drupal-sivuston tallennetuista konfiguraatitiedostoista, luo tämän jälkeen halutun määrän entiteettejä ja ajaa vaaditun testin mitaten suoritusajan jokaisesta vaiheesta. Käyttöliittymän kautta tehtävät lataukset tehtiin käsin, ja niistä suoritus aika mitattiin selaimen kehittäjätyökaluja käyttämällä. Jokaista datan vientitestiä kohden testi alkaa nollatilanteesta.

Testejä ajettiin erikokoisilla näytteillä, joista esimerkiksi 10 / 20 -näyte tarkoittaa, että testi luo 10 *Parent* entiteettiä, joista kullakin on 20 *Child* entiteettiä alapuolellaan, joista niin ikään kullakin on 20 *Granchild* entiteettiä alapuolellaan. Toisin sanoen 10 / 20 -näyte tarkoittaa että entiteettejä luodaan yhteensä  $10 * 20 * 20 = 4000$  kappaletta. Vastaavasti 1000 / 50 -näyte tarkoittaa kokonaismäärältä  $1000 * 50 * 50 = 2500000$  entiteettiä.

## 4 TULOKSET JA NIIDEN ARVIOINTI

Tässä osiossa käydään läpi tietomallin vaikutus eri prosesseissa. Työssä tehtiin mittaukset sekä datan luomisesta että neljällä eri tavalla ladattuna CSV-tiedostoksi. Mittaukset tehtiin yhdeksällä eri näytekoolla, jolla pyrittiin hahmottamaan paremmin missä vaiheessa niin tietomallilla kuin datan vientimallilla alkaa olla riittävä merkitys, että niihin on suotavaa kiinnittää huomiota järjestelmän kehityksessä.

Työ keskittyy enimmäkseen datan vientimallien vertailuun, joka onkin taustalla olleen projektin oleellisin ratkaistava asia - saada riittävän kestävä ratkaisu, joka skaalautuu myös tulevaisuuden tarpeisiin. Koska työtä varten luotiin erillinen entiteettimalli, luontaisena lisäyksenä tuli sen myötä myös mitattua luonnin nopeutta. Vaikka työ keskittyykin suurilta osin vain yhteen käyttötapaukseen, on huomionarvoista, että entiteettimallin vaikutus eri tilanteissa on verrannollinen myös käytännössä kaikkeen muuhunkin toimintaan Drupal-järjestelmässä.

### 4.1 DATAN LUONTI

Datan luonti vastaa lähes kaikkia ohjelmallisia käsittelyitä entiteeteille. Taulukko 4.1 taulukossa on esitetty mitatut nopeudet tietyn entiteettimäärän luonnista eri tietomalleilla.

**Taulukko 4.1.** Datan luontiajat eri tietomalleilla.

Sample size	First	Second	Third
10/10	0m1.350s	0m1.393s	0m1.029s
100/10	0m10.351s	0m10.642s	0m7.810s
1000/10	1m39.462s	1m43.818s	1m16.572s
10/20	0m4.095s	0m4.308s	0m3.209s
100/20	0m37.963s	0m38.956s	0m28.851s
1000/20	6m16.607s	6m30.487s	4m48.264s
10/50	0m22.644s	0m23.578s	0m17.485s
100/50	3m46.436s	3m56.710s	2m54.820s
1000/50	76m16.663s	40m35.339s	30m3.846s

Pienemmissä entiteettimäärissä erot ensimmäisen ja toisen tietomallin välillä oli hyvin vähäiset, vastoin ennako-olettamusta toinen tietomalli pärjäsi jopa hieman ensimmäistä paremmin. Vasta äärimmäisen suurella entiteettimäärällä (2,5 miljoonaa) toinen tietomalli oli merkittävästi ensimmäistä nopeampi. Kolmas tietomalli oli kaikissa entiteettimäärissä selvästi muita nopeampi, korostuen etenkin keskisuurissa ja suurissa entiteettimäärissä.

Mittaus osoittaa, että jo suhteellisen pienellä entiteettimallin kompleksisuudella on syytä tarkastella tietomallin toteutustapaa, mikäli entiteettejä on tarve käsitellä massana

esimerkiksi päivityspoluissa tai aikataulutetuissa rutiineissa. Hyvin kompleksisissa entiteettimalleissa merkittävä parannetun tietomallin hyöty voidaan saavuttaa jo pienissäkin entiteettimäärissä.

## 4.2 SUORA TIETOKANTAYHTEYS

Suoralla tietokantayhteydellä lataaminen vastaa ihanteellista tavoiteaikaa. Taulukko 4.2 taulukossa on esitetty mitatut nopeudet tietyn entiteettimäärän lataamisesta eri tietomalleilla.

**Taulukko 4.2.** Datan latausajat suoralla tietokantayhteydellä.

Sample size	First	Second	Third
10/10	0m0.221s	0m0.214s	0m0.201s
100/10	0m0.380s	0m0.395s	0m0.340s
1000/10	0m3.305s	0m3.262s	0m2.876s
10/20	0m0.277s	0m0.264s	0m0.249s
100/20	0m1.168s	0m1.165s	0m0.946s
1000/20	0m13.422s	0m13.556s	0m11.474s
10/50	0m0.724s	0m0.721s	0m0.618s
100/50	0m8.161s	0m8.129s	0m6.926s
1000/50	1m23.626s	1m23.230s	1m10.630s

Mittaus osoitti ensimmäisen ja toisen tietomallin olevan käytännössä yhtä nopeita ladattavia, erot näiden tietomallien latausaikojen välillä olivat marginaalisia, toinen tietomalli oli niukasti ensimmäistä nopeampi ennako-oletuksen mukaisesti, toisin kuin datan luonnin kohdalla. Kolmas tietomalli oli muita malleja nopeampi, mutta merkittävät eroavaisuudet vauhdissa alkoi vasta keskisuurten ja suurten entiteettimäärien kohdalla.

Mittauksista voidaan myös havaita, että vaikka entiteettien luomiseen kuluisi yli tunti aikaa, voidaan raportti muodostaa samaisista entiteeteistä vain reilussa minuutissa. Käytännössä suoralla tietokantayhteydellä ladatun raportin nopeus vastaa ihanteellista tavoiteaikaa, ja tarkoitus onkin löytää vaihtoehtoinen toteutustapa, joka pääsisi mahdollisimman lähelle näitä lukemia.

Työn taustalla olleen projektin lähtökohta oli toteutettu juuri suorilla tietokantakyselyillä, se on täysin toimiva ja jopa suotava tapa toteuttaa integraatio ulkoiseen järjestelmään sellaisissa tapauksissa missä tietomallit eivät muutu jatkuvasti järjestelmän kehityksessä. Projektin kehitysvauhdin takia nämä tietokantakyselyt olivat hyvin usein hajalla, ja asia huomattiin monesti vasta varsinaisia raportteja läpi käydessä, kun lukuja verrattiin keskenään. Suorien tietokantakyselyjen ylläpitäminen vaatii jatkuvaa ylläpitämistä jos järjestelmää kehitetään, ja tietokantaosaamisen lisäksi tekijältä vaaditaan vahvaa

Drupal-osaamista, jotta tietoja haetaan varmasti oikeista tietokantatauluista. Entiteettien revisiot sekä kielikäännökset voivat tehdä kyselyistä huomattavan monimutkaisia.

### 4.3 VIEWS DATA EXPORT -MODUULI

VDE-yhteisömoduuli on yleisesti käytössä oleva datan vientityökalu. Taulukko 4.3 taulukossa on esitetty mitatut nopeudet tietyn entiteettimäärän lataamisesta eri tietomalleilla.

**Taulukko 4.3.** Datan latausajat VDE-moduulilla.

Sample size	First	Second	Third
10/10	3.4s	2.75s	2.67s
100/10	18.63s	18.21s	17.13s
1000/10	3.1m	3.1m	2.7m
10/20	8.37s	7.70s	7.38s
100/20	1.2m	1.2m	1.1m
1000/20	15m	14.9m	11.5m
10/50	44.55s	43.87s	41.26s
100/50	8.3m	8.3m	6.9m
1000/50	* 49min / 14%	* 42min / 14%	* 45min / 37%

Mittaus osoitti, ettei tietomallilla ole merkittävää vaikutusta latausnopeuteen VDE-moduulia käyttäessä kuin vasta äärimmäisen suurissa entiteettimäärissä (2,5 miljoonaa), jolloin kolmas tietomalli on noin puolta nopeampi kuin ensimmäinen ja toinen tietomalli. Mittaus kuitenkin osoittaa, että VDE-moduuli on tuskaisen hidas ja jo keskisuurista entiteettimääristä luotavat raportit kannattaa sijoittaa esimerkiksi kahvitauolle tapahtuviksi. Mittaus keskeytettiin äärimmäisten suurten entiteettimäärien kohdalla kesken, joista ilmoitettuna on moduulin itsensä ilmoittama edistymisen. Valmistuneissa mittauksissa tulos on tarkastettu selaimen todellisesti käyttämästä ajasta.

Views Data Export on yleisesti käytössä oleva yhteisömoduuli [21], moduulia käytetään yli sadalla tuhannella Drupal-sivustolla. Moduuli mahdollistaa datan viemisen esimerkiksi CSV ja XLS-formaateissa muihin järjestelmiin. Teoriassa siis moduuli on kuin luotu työn taustalla olleen projektin käyttötarkoitukseen, mittauksesta kuitenkin voidaan havaita ettei tämä ratkaisu skaalautu tilanteisiin, joissa suuria tietomääriä on tarve ladata toistuvasti.

Allekirjoittaneen omien kokemusten mukaan moduuli voi todellisissa järjestelmissä olla jo hyvin pienissäkin tietomäärissä niin hidas, että yksinkertaisen raportin ulosvienti kestää useita tunteja. Tilannetta voi pyrkiä parantamaan sopivilla tietokantaindekseillä, mutta jatkuvaan käyttöön tarvittavat raportit ovat suotavaa luoda jollain muulla tavoin.

## 4.4 ENTITY STORAGE -LUOKKA

Entity storage on Drupal-ytimen tarjoama varastoinnista vastaava ohjelmaluokka. Taulukko 4.4 taulukossa on esitetty mitatut nopeudet tietyn entiteettimäärän lataamisesta eri tietomalleilla.

**Taulukko 4.4.** Datan latausajat entity storage -luokalla.

Sample size	First	Second	Third
10/10	0m0.525s	0m0.586s	0m0.470s
100/10	0m3.407s	0m4.283s	0m2.969s
1000/10	0m37.361s	1m37.481s	1m4.122s
10/20	0m1.194s	0m1.339s	0m1.000s
100/20	0m11.086s	0m19.546s	0m13.224s
1000/20	2m9.814s	15m7.771s	9m56.603s
10/50	0m5.582s	0m8.461s	0m5.846s
100/50	1m2.606s	5m33.867s	3m42.237s
1000/50	DNF	DNF	DNF

Tämä mittaus oli yllättävin ennako-oletusten kannalta. Mittaus esittää toisen tietomallin selvästi muita hitaampana entiteettejä ladatessa. Ensimmäinen tietomalli on etenkin keskisuurissa ja suurissa tietomäärissä huomattavasti nopeampi jopa kolmatta tietomallia.

Taustalla näiden mittauksien erosta muihin mittauksiin tulee entity storagen tavasta käsitellä entiteettejä, joka tässä tapauksessa suosii ensimmäistä tietomallia suuresti, sillä ladatessa `parent`-entiteetti, latautuu myös alapuoliset entiteetit samalla, tämä vähentää määrää erikseen tapahtuvista latauksista, joka nopeuttaa prosessia merkittävästi. Toinen tulosta selittävä tekijä on työssä käytetty puumainen rakenne, jossa alapuolisia entiteettejä on merkittävän paljon. Erittäin suurien entiteettimäärien kohdalla entity storage vaatii liikaa resursseja ja mittaus keskeytyi.

Entity storage on tavanomainen tapa hallinnoida entiteettejä ohjelmakoodissa [22]. Luokka tarjoaa kaikki tarvittavat rajapinnat entiteettien käsittelyyn, kuten esimerkiksi tässä mittauksessa käytetyn `loadMultiple`-funktion. Luokan käyttäminen vaatii kuitenkin hieman ymmärrystä käsiteltävistä entiteeteistä, sillä siinä ei ole sisäänrakennettuna mitään suojaa liiallisen tietomäärän lataamiselle kerrallaan. Mittauksesta voidaankin havaita, että mikäli ladattavaa tietoa tulee kerralla liikaa, on riskinä että palvelimelta loppuu muisti (tmv. resurssi) kesken. Lataus kuitenkin hidastuu merkittävästi mitä pienempiin osiin ladattava määrä joudutaan jakamaan, joten optimaalisen latauskoon löytäminen voi olla haastavaa.

Työn taustalla olleeseen projektiin ensimmäinen suorat tietokantayhteydet korvaava rajapinta rakennettiin entity storagea käyttämällä. Ajansaatossa tietotarve kuitenkin ke-

hittyi suppeista raporteista laaja-alaisiksi kokonaisen entiteettipuurakenteen sisältäviksi tiedonsiirtoajoiksi. Tällaiset tietomäärät kestivät tunteja ajaa läpi ja samalla palvelinkuormitus oli niin valtava, että jokin toinen raskas prosessi saattoi kaataa koko palvelinympäristön.

Tätä mallia voi suositella käytettäväksi tavanomaisissa Drupal-järjestelmissä, etenkin sellaisissa toteutuksissa, joissa VDE-moduuli on ollut vielä käytettävissä mutta koettu turhan hitaaksi. Isoissa järjestelmissä ja integraatioissa, joissa data voidaan lähettää deltaerotuksella tämä ratkaisu on myös täysin varteenotettava vaihtoehto. Mikäli tarve on ajaa täyttä datansiirtoa toistuvasti isossa järjestelmässä, tämä ratkaisu saattaa olla riittämätön ja aiheuttaa turhia kustannuksia esimerkiksi pilvipalveluissa.

## 4.5 OPTIMOITU LATAUSLUOKKA

Optimoitu latausluokka on muunneltu versio entity storage luokasta. Taulukko 4.5 taulukossa on esitetty mitatut nopeudet tietyn entiteettimäärän lataamisesta eri tietomalleilla.

**Taulukko 4.5.** Datan latausajat optimoidulla latausluokalla.

Sample size	First	Second	Third
10/10	0m0.237s	0m0.237s	0m0.219s
100/10	0m0.588s	0m0.617s	0m0.482s
1000/10	0m4.603s	0m4.631s	0m3.237s
10/20	0m0.351s	0m0.352s	0m0.310s
100/20	0m1.852s	0m1.943s	0m1.354s
1000/20	0m18.640s	0m19.589s	0m12.243s
10/50	0m1.200s	0m1.253s	0m0.927s
100/50	0m11.449s	0m11.598s	0m7.924s
1000/50	11m13.091s	9m59.812s	1m22.726s

Mittaus osoitti ensimmäisen ja toisen tietomallin olevan käytännössä yhtä nopeita, ensimmäisen ollessa hieman nopeampi suurempien entiteettipuiden kanssa. Etenkin suuremmissa entiteettimäärissä kolmas tietomalli oli kuitenkin lyömätön, päästen lähes samoihin lukuihin suoran tietokantayhteyden kanssa (Taulukko 4.2). Erittäin suurissa entiteettimäärissä (2,5 miljoonaa) myös ensimmäinen ja toinen tietomalli on siedettävä verrattuna muihin kuin suoraan tietokantakyselyyn.

Mittaus korostaa etenkin kompleksisissa entiteettirakenteissa tai suurissa entiteettimäärissä tietomallin optimoinnin tärkeyden. Oikein optimoitu tietomallia voidaan käsitellä merkittävästi nopeammin, jopa kymmenesosalla alkuperäisestä ajasta tietyissä olosuhteissa.

Optimointi luotiin entity storage -luokan pohjalta, kyseisen luokan `loadMultiple` funktiota muokattiin siten että se loi samat tietokantakyselyt kuin normaalistikin, mutta sen

---

sijaan että ladatulla datalla muodostettaisiin entiteettiäluokkia, tieto palautettiin sellaiseenaan. Näin tehtynä kaikki haluttu data on työn tapauksessa jo valmiiksi sellaisessa muodossa kuin sen pitäkin olla, poislukien referenssikentät. Referenssien osalta ns. raakadata pitää sisällään vain entiteetin ID-numeron, joten näitä varten täytyy ohjelmallisesti tehdä yhdistelyitä. Optimoitu latausluokka on silti merkittävästi nopeampi lataamaan kaikkien tarvittavien entiteettityyppien datat ja yhdistelemään ne kuin valmiit tarjolla olevat ratkaisut. Työssä tuotettu optimoitu latausluokka löytyy liitteestä LIITE A.

Optimoitua latausluokkaa olisi mahdollista käyttää korvaamaan kaikki datavientitarpeet Drupal-järjestelmissä. Erityisesti CSV-vienteihin optimoitu latausluokka tarjoaisi merkittävää nopeutusta.

## 5 YHTEENVETO

Työn tavoitteena oli selvittää tapoja parantaa Drupal-järjestelmän suorituskykyä, erityisesti datan viennin osalta muihin järjestelmiin. Pyrkimyksenä oli käydä läpi tarvittavat seikat työn taustalta lyhyesti ja ytimekkäästi läpi siten, että alalla työskentelevän on mahdollista saada kiinni tavoitteista ja tuloksista ilman ylimääräistä jaarittelua.

Taustalla ollut projekti oli loistava tilaisuus pyrkiä parempaan kuin mitä valmiilla ratkaisuilla oli mahdollista saavuttaa, sillä projektissa jaettiin näkemys siitä, että kun kerralla tekee hyvän, säästää jatkokehityksessä sekä ylläpidossa. Tuloksena syntyi toteutus, josta allekirjoittanut uskoo olevan hyötyä jokaisessa Drupal-toteutuksessa.

### 5.1 JOHTOPÄÄTÖKSET

Työ onnistui tavoitteissaan kiitettävästi, molempiin tutkimuskysymyksiin löytyi kattava vastaus siten, että siitä on hyötyä niin pienissä kuin isoissakin Drupal-toteutuksissa. Parantamisen varaa työhön silti jäi, sillä mittausvertailu suoritettiin paikallisessa ympäristössä, jolloin mittauksissa ei heijastu se hitaus, joka muodostuu kuormittuneessa tuotantoympäristössä.

- Miten Drupal-järjestelmän data viedään ulkoisiin järjestelmiin skaalautuvasti?

Ensimmäiseen tutkimuskysymykseen saatiin kaksi pätevää ratkaisua, joilla datan vienti on mahdollista toteuttaa siten, että kun järjestelmän käyttäjien ja sen myötä sisällön määrä kasvaa, järjestelmän toiminta ei häiriinny tai keskeydy ja tiedonsiirtoon käytetty säilyy siedettävänä. Ratkaisun valintaan vaikuttaa merkittävästi kyseisen järjestelmän kehitystila.

Järjestelmille, jotka ovat jo saavuttaneet ylläpitotilan, eli niiden tietomalleja ei muuteta kuin korkeintaan harvoin, parhaimman mahdollisen suorituskyvyn saavuttaa suorilla tietokantakyselyillä. Näiden rakentaminen vaatii sekä Drupal-osaamista entiteettien rakenteista että tietokantaosaamista erityisesti usean tietokantataulun yhdistämisestä. Tämän mallin hyödyt korostuvat esimerkiksi pilvipalveluihin toteutetuissa järjestelmissä, joissa tietokanta voidaan kahdentaa, jolloin varsinainen datan siirto voidaan toteuttaa erilliseen tietokantaan, eikä se tällöin aiheuta kuormitusta Drupal-järjestelmään tai sen käyttäjille. Mallin heikkoutena on työläs pystyttäminen laajoissa järjestelmissä sekä kyvyssä havaita muutoksia Drupal-järjestelmän tietomalleissa.

Sellaisille järjestelmille, jotka ovat edelleen kehitystilassa esimerkiksi muuttuvien vaatimusten tai lainsäädännön vuoksi, optimoitu latausluokka tarjoaa dynaamisen lähestymistavan datan viennille, mahdollistaen lähes suorien tietokantakyselyiden kaltaiset nopeudet. Tämän mallin hyödyt korostuvat erityisesti laajoissa järjestelmissä, sillä uuden

entiteettityypin lisääminen olemassa olevaan integraatioon saadaan luotua tarvittaessa suoraan käyttöliittymästä käsin hyödyntäen Drupalin entiteettinäkymiä, jolloin muutokset ovat hyvin nopea suorittaa.

Optimoitu latausluokka on pitkän kehityksen tuotos sellaisessa projektissa, jossa erilaisia entiteettityyppejä oli useita kymmeniä, eikä toteutustapoja ollut vakioitu tai välttämättä edes erityisemmin harkittu. Käytetyimpien entiteettityyppien kohdalla entiteettien lukumäärät liikkuvat miljoonissa, ja vientimallin tuli skaalautua vielä vuosiksi eteenpäinkin. Malli on suunniteltu siten, että kykenee siirtämään ilman deltaerotusta useiden satojen gigatavujen kokoisen (pakatun) tietokannan sisällön ilman aikakatkaisuja. Tyypillinen Drupal-järjestelmän tietokanta on muutamia gigatavuja pakattuna.

- Miten entiteetin toteutustapa vaikuttaa datan käsittelyn nopeuteen Drupal-järjestelmässä?

Toiseen tutkimuskysymykseen saatiin hieman ympärilyöreämpi vastaus, sillä toteutustapa voi vaikuttaa jopa suuresti tai ei lainkaan käsittelyn nopeuteen riippuen tilanteesta. Mittauksissa yksiselitteiseksi jäi, että peruskenttiä tulisi käyttää tavallisten kenttien sijaan aina kun vain mahdollista. Peruskentillä toteutettu tietomalli on merkittävästi nopeampi käytännössä jokaisessa tilanteessa.

Sellaisissa tapauksissa missä kaikki alapuoliset entiteetit normaalisti ladataan tai esitetään samanaikaisesti, voi 1:N referenssikenttä (parent->child) mahdollistaa huomattavasti 1:1 referenssikenttää (child->parent) nopeammat latausajat. Kyseistä tietomallia ei voi kuitenkaan suoraan suositella ilman tapauskohtaista perehtymistä, sillä pääsääntöisesti molemmat tietomallit olivat pääsääntöisesti käytännössä yhtä nopeita, 1:1 mallin ollessa marginaalisesti nopeampi.

## 5.2 JATKOKEHITYSEHDOTUKSET

Työn toisen tutkimuskysymyksen kannalta olisi hyödyllistä tehdä syvällisempää tutkimusta siitä miten laajemmissa entiteettirakenteissa eri toteutustavat näyttäytyvät erilaisissa tilanteissa. Tyypillisessä Drupal-toteutuksessa sisältöä käsitellään paljon käyttöliittymän kautta eri käyttäjien toimesta ja sopivalla entiteettirakenteella saattaisi olla merkittävä vaikutus käyttäjäkokemukseen, sillä usein välimuistista ei ole hyötyä eri käyttäjien välillä tapahtuvissa toiminnoissa, vaan latausajat muodostuvat ensimmäisellä latauskerralla suoraan rakenteesta ja sitä välittävistä toiminnoista kullekin käyttäjälle erikseen.

Entiteettirakenteen merkitystä kompleksisissa entiteettirakenteissa olisi hyödyllistä myös tutkia päivitysrutiinien kaltaisissa massakäsittelyissä. Nykytilassa massa-ajoissa on mahdollista törmätä ongelmaan, jossa esimerkiksi tietueen päivittäminen ei ole mahdollista päivitysrutiinissa aikakatkaisun vuoksi, jolloin toteutus joudutaan kiertämään jonokäsittelyssä (engl. QueueWorker). Tällöin yhtäaikaisesta käytöstä saattaa syntyä haasteita,

jotka vaativat erityistä huomiota. Sopivilla entiteettirakenteilla voitaisiin välttää jonokäsittelyn tarve.

Ensimmäiseen tutkimuskysymykseen liittyen jatkokehitystä voisi suositella erityisesti Views data export -moduulin parantamisella, työssä syntyneen optimoidun latausluokan pohjalta voidaan todeta, ettei moduulin hitaudelle ole perusteita. Oletus nykytilasta on, että moduulin hitaus syntyy muun muassa kenttäkohtaisista oikeuksien tarkistuksista. Mikäli esimerkiksi raportti on jo itsessään suojattu jollain muulla oikeustarkastuksella, kenttäkohtainen tarkastelu ei luultavasti ole tarpeellista.

## VIITTEET

- [1] Drupal, "Overview of Drupal". Viitattu: 23. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/getting-started/understanding-drupal/overview-of-drupal>
- [2] Drupal, "Kunta.fi". Viitattu: 23. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/project/kada>
- [3] Hilma, "Tampere3-hankkeen verkkopalvelukokonaisuuden suunnittelu ja toteutus Drupal-alustalle.". Viitattu: 23. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.hankintailmoitukset.fi/fi/public/procurement/13813/notice/15776/overview>
- [4] Hevner, March, Park, ja Ram, "Design Science in Information Systems Research", *MIS Quarterly*, vsk. 28, nro 1, s. 75, 2004, doi: [10.2307/25148625](https://doi.org/10.2307/25148625).
- [5] Drupal, "Concept: Drupal as a Content Management System". Viitattu: 23. huhtikuuta 2025. [Verkossa]. Saatavissa: [https://www.drupal.org/docs/user\\_guide/en/understanding-drupal.html](https://www.drupal.org/docs/user_guide/en/understanding-drupal.html)
- [6] A. Bergstein, *Drupal 10 Masterclass: Build responsive Drupal applications to deliver custom and extensible digital experiences to users*, 1. p. Birmingham: Packt Publishing Limited, 2023.
- [7] Drupal, "Introduction to Entity API in Drupal 8". Viitattu: 27. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/entity-api/introduction-to-entity-api-in-drupal-8>
- [8] Drupal, "Database API Overview". Viitattu: 27. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/database-api/database-api-overview>
- [9] L. E. Ullman, *MySQL*, 2nd ed. Berkeley, CA: Peachpit, 2006.
- [10] Drupal, "Broken reference". Viitattu: 27. huhtikuuta 2025. [Verkossa]. Saatavissa: [https://www.drupal.org/project/broken\\_reference](https://www.drupal.org/project/broken_reference)
- [11] Drupal, "Adding Basic Fields to a Content Type". Viitattu: 27. huhtikuuta 2025. [Verkossa]. Saatavissa: [https://www.drupal.org/docs/user\\_guide/en/structure-fields.html](https://www.drupal.org/docs/user_guide/en/structure-fields.html)
- [12] Drupal, "FieldTypes, FieldWidgets and FieldFormatters". Viitattu: 27. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/entity-api/fieldtypes-fieldwidgets-and-fieldformatters>

- 
- [13] Drupal, "Field API". Viitattu: 28. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://api.drupal.org/api/drupal/core%21modules%21field%21field.module/group/field/10>
- [14] Drupal, "Updating Configuration". Viitattu: 28. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/update-api/updates-configuration>
- [15] Drupal, "Defining and using Content Entity Field definitions". Viitattu: 28. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/entity-api/defining-and-using-content-entity-field-definitions>
- [16] Drupal, "Updating Database Schema and/or Data in Drupal". Viitattu: 28. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/update-api/updates-database-schema-and-or-data-in-drupal>
- [17] Drupal, "Dynamic/Virtual field values using computed field property classes". Viitattu: 29. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/drupal-apis/entity-api/dynamic-virtual-field-values-using-computed-field-property-classes>
- [18] Drupal, "Content moderation module overview". Viitattu: 29. huhtikuuta 2025. [Verkossa]. Saatavissa: <https://www.drupal.org/docs/8/core/modules/content-moderation/overview>
- [19] Drupal, "ContentEntityBase class". Viitattu: 6. toukokuuta 2025. [Verkossa]. Saatavissa: <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21ContentEntityBase.php/class/ContentEntityBase/11.x>
- [20] RelationalDBDesign, "Three Types of Relationships". Viitattu: 16. toukokuuta 2025. [Verkossa]. Saatavissa: <https://www.relationaldbdesign.com/database-design/module6/three-relationship-types.php>
- [21] Drupal, "Views Data Export". Viitattu: 26. toukokuuta 2025. [Verkossa]. Saatavissa: [https://www.drupal.org/project/views\\_data\\_export](https://www.drupal.org/project/views_data_export)
- [22] Drupal, "EntityStorageBase class". Viitattu: 26. toukokuuta 2025. [Verkossa]. Saatavissa: <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21EntityStorageBase.php/class/EntityStorageBase/11.x>

## LIITE A: DRUPAL-DATAN LATAUKSEEN OPTIMOITU LATAUSLUOKKA

```
1 <?php
2
3 namespace Drupal\Hellsten\Entity;
4
5 use Drupal\Core\Database\Database;
6 use Drupal\Core\Entity\ContentEntityTypeInterface;
7 use Drupal\Core\Entity\EntityStorageException;
8 use Drupal\Core\Entity\Sql\DefaultTableMapping;
9 use Drupal\Core\Entity\Sql\SqlContentEntityStorage;
10 use Drupal\Core\Field\FieldStorageDefinitionInterface;
11 use Drupal\Core\Site\Settings;
12
13 /**
14  * Class to quickly load all entity data in array.
15  */
16 class FastLoader {
17
18     /**
19      * Database connection.
20      *
21      * @var \Drupal\Core\Database\Connection
22      */
23     protected $database;
24
25     /**
26      * Entity field manager.
27      *
28      * @var \Drupal\Core\Entity\EntityFieldManagerInterface
29      */
30     protected $entityFieldManager;
31
32     /**
33      * Entity type id.
34      *
35      * @var string
36      */
37     protected $entityTypeId;
38
```

```
39  /**
40   * Field storage definitions.
41   *
42   * @var \Drupal\Core\Field\FieldStorageDefinitionInterface[]
43   */
44  protected $fieldStorageDefinitions;
45
46  /**
47   * Entity type.
48   *
49   * @var \Drupal\Core\Entity\ContentEntityTypeInterface
50   */
51  protected ContentEntityTypeInterface $entityType;
52
53  /**
54   * Base table for the entity data.
55   *
56   * @var string
57   */
58  protected $baseTable;
59
60  /**
61   * Data table for the entity data.
62   *
63   * @var string
64   */
65  protected $dataTable;
66
67  /**
68   * Whether fields should be filtered.
69   *
70   * @var bool
71   */
72  protected $limitFields;
73
74  /**
75   * Fields to be select or empty array for all fields.
76   *
77   * @var array
78   */
79  protected $fields;
80
81  /**
```

```
82     * Selected bundle. Empty string for all bundles.
83     *
84     * @var string
85     */
86     protected $bundle;
87
88     /**
89     * Key for ID value.
90     *
91     * @var false|string
92     */
93     protected $idKey;
94
95     /**
96     * Key for bundle value.
97     *
98     * @var false|string
99     */
100    protected $bundleKey;
101
102    /**
103    * Whether to use temporary table map.
104    *
105    * @var bool
106    */
107    protected $temporary = FALSE;
108
109    /**
110    * The mapping of field columns to SQL tables.
111    *
112    * @var \Drupal\Core\Entity\Sql\DefaultTableMapping
113    */
114    protected $tableMapping;
115
116    /**
117    * Limit of the results returned.
118    *
119    * @var int
120    */
121    protected $queryLimit;
122
123    /**
124    * ID to start query from.
```

```
125     *
126     * @var string|int
127     */
128     protected $startId;
129
130     /**
131     * Constructor.
132     *
133     * @param string $entityTypeId
134     *   The entity type.
135     * @param array $fields
136     *   Limit to these fields (base fields will be added
137     *   regardless)
138     * @param string $bundle
139     *   The bundle (does not matter with non-bundled entities).
140     * @throws \Drupal\Core\Entity\EntityStorageException
141     */
142     public function __construct(string $entityTypeId, array $fields
143     = [], string $bundle = '') {
144         $primary = Settings::get('raw_loader.primary_database',
145         'default');
146         try {
147             $database = Database::getConnection('default', $primary);
148         }
149         catch (\Exception $e) {
150             \Drupal::logger('raw_loader')
151             →error("Unable to use primary database, using default.
152             Error: {$e→getMessage()}");
153             $database = Database::getConnection();
154         }
155
156         $this→database = $database;
157         $this→entityFieldManager =
158         \Drupal::service('entity_field.manager');
159         $this→entityTypeId = $entityTypeId;
160         $this→fieldStorageDefinitions = $this→entityFieldManager→
161         >getActiveFieldStorageDefinitions($entityTypeId);
162         $entityTypeManager = \Drupal::entityTypeManager();
163         $storage = $entityTypeManager→getStorage($this→
164         >entityTypeId);
165
166         if (!$storage instanceof SqlContentEntityStorage) {
```

```
161     throw new EntityStorageException('Not SQL content entity
type');
162   }
163
164   /** @var \Drupal\Core\Entity\ContentEntityTypeInterface
EntityType */
165   $entityType = $storage->getEntityType();
166   $this->entityType = $entityType;
167   $this->baseTable = $storage->getBaseTable();
168   $this->dataTable = $storage->getDataTable();
169   $this->limitFields = (bool) $fields;
170   $this->setFields($fields);
171   $this->bundle = $bundle;
172   $this->idKey = $entityType->getKey('id');
173   $this->bundleKey = $entityType->getKey('bundle');
174   }
175
176   /**
177    * Load all entities.
178    *
179    * @return array
180    *   Array of entities.
181    */
182   public function loadMultiple() {
183     return $this->getFromStorage();
184   }
185
186   /**
187    * Limit the amount of entities loaded at once.
188    *
189    * @param int $limit
190    *   The limit.
191    *
192    * @return $this
193    */
194   public function setQueryLimit($limit): self {
195     $this->queryLimit = $limit;
196     return $this;
197   }
198
199   /**
200    * Begin from ID next to given.
201    *
```

```
202 * @param int|string $id
203 *   ID to start from.
204 *
205 * @return $this
206 */
207 public function setStartId($id): self {
208     $this->startId = $id;
209     return $this;
210 }
211
212 /**
213  * Check if there are more results to return.
214  *
215  * @return bool
216  *   True if there are any results left after this set.
217  */
218 public function hasMoreResults(): bool {
219     $query = $this->buildQuery()
220         ->range($this->queryLimit, 1);
221     if ($this->startId) {
222         $query->condition($this->idKey, $this->startId, '>');
223     }
224     return (bool) $query->execute()->fetchAllAssoc($this->idKey);
225 }
226
227 /**
228  * Gets entities from the storage.
229  *
230  * @return array
231  *   Array of entities from the storage.
232  *
233  * @see
234  */
235 protected function getFromStorage() {
236     $entities = [];
237
238     $query = $this->buildQuery();
239     if ($this->queryLimit) {
240         $query->range(0, $this->queryLimit);
241     }
242     if ($this->startId) {
243         $query->condition($this->idKey, $this->startId, '>');
```

```
244     }
245
246     $records = $query->execute()->fetchAllAssoc($this->idKey);
247
248     // Map the loaded records into entity objects and according
fields.
249     if ($records) {
250         $entities = $this->mapFromStorageRecords($records);
251     }
252
253     return $entities;
254 }
255
256 /**
257  * Set fields to load.
258  *
259  * @param array $fields
260  *   Array of additional fields.
261  *
262  * @return $this
263  */
264 private function setFields(array $fields = []): self {
265     // Add fields from the {entity} table.
266     $table_mapping = $this->getTableMapping();
267     $entity_fields = $table_mapping->getAllColumns($this->
baseTable);
268     $this->fields =
array_values(array_unique(array_merge($entity_fields, $fields)));
269     return $this;
270 }
271
272 /**
273  * Get fields.
274  *
275  * @return array
276  *   The fields.
277  */
278 public function getFields() {
279     return $this->fields;
280 }
281
282 /**
283  * Builds the query to load the entity.
```

```

284  *
285  * @return \Drupal\Core\Database\Query>SelectInterface
286  *   A SelectQuery object for loading the entity.
287  *
288  * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::buildQuery
289  */
290  protected function buildQuery() {
291      $query = $this->database->select($this->baseTable, 'base');
292
293      // Add fields from the {entity} table.
294      $table_mapping = $this->getTableMapping();
295      $entity_fields = $table_mapping->getAllColumns($this->
>baseTable);
296
297      $query->fields('base', $entity_fields);
298      if ($this->bundleKey && $this->bundle) {
299          $query->condition($this->bundleKey, $this->bundle);
300      }
301      $query->orderBy($this->idKey);
302
303      return $query;
304  }
305
306  /**
307   * Gets a table mapping for the entity's SQL tables.
308   *
309   * @return \Drupal\Core\Entity\Sql\DefaultTableMapping
310   *   A table mapping object for the entity's tables.
311   *
312   * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::getTableMapping
313   */
314  protected function getTableMapping() {
315      if (!isset($this->tableMapping)) {
316          $this->tableMapping = $this->getCustomTableMapping($this->
>entityType, $this->fieldStorageDefinitions);
317      }
318      return $this->tableMapping;
319  }
320
321  /**
322   * Gets a table mapping for the specified entity type and

```

```

storage definitions.
323  *
324  * @param \Drupal\Core\Entity\ContentEntityTypeInterface
$entity_type
325  * An entity type definition.
326  * @param \Drupal\Core\Field\FieldStorageDefinitionInterface[]
$storage_definitions
327  * An array of field storage definitions to be used to compute
the table
328  * mapping.
329  * @param string $prefix
330  * (optional) A prefix to be used by all the tables of this
mapping.
331  * Defaults to an empty string.
332  *
333  * @return \Drupal\Core\Entity\Sql\TableMappingInterface
334  * A table mapping object for the entity's tables.
335  *
336  * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::getCustomTableMapping
337  */
338  private function
getCustomTableMapping(ContentEntityTypeInterface $entity_type, array
$storage_definitions, $prefix = '') {
339      $prefix = $prefix ?: ($this->temporary ? 'tmp_' : '');
340      return DefaultTableMapping::create($entity_type,
$storage_definitions, $prefix);
341  }
342
343  /**
344  * Maps from storage records to entity arrays, and attaches
fields.
345  *
346  * @param array $records
347  * Associative array of query results, keyed on the entity ID
or revision
348  * ID.
349  *
350  * @return array
351  * An array of entity arrays.
352  *
353  * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::mapFromStorageRecords

```

```
354  */
355  protected function mapFromStorageRecords(array $records) {
356      if (!$records) {
357          return [];
358      }
359
360      // Get the names of the fields that are stored in the base
table and, if
361      // applicable, the revision table. Other entity data will be
loaded in
362      // loadFromSharedTables() and loadFromDedicatedTables().
363      $field_names = $this->tableMapping->getFieldNames($this-
>baseTable);
364
365      $values = [];
366      foreach ($records as $id => $record) {
367          $values[$id] = [];
368          // Skip the item delta and item value levels (if possible)
but let the
369          // field assign the value as suiting. This avoids
unnecessary array
370          // hierarchies and saves memory here.
371          foreach ($field_names as $field_name) {
372              $field_columns = $this->tableMapping-
>getColumnNames($field_name);
373              // Handle field types that store several properties.
374              if (count($field_columns) > 1) {
375                  $definition_columns = $this-
>fieldStorageDefinitions[$field_name]->getColumns();
376                  foreach ($field_columns as $property_name =>
$column_name) {
377                      if (property_exists($record, $column_name)) {
378                          if (empty($definition_columns[$property_name]
['serialize'])) {
379                              $values[$id][$field_name][$property_name] =
$record->{$column_name};
380                          }
381                          unset($record->{$column_name});
382                      }
383                  }
384              }
385              // Handle field types that store only one property.
386              else {
```

```

387         $column_name = reset($field_columns);
388         if (property_exists($record, $column_name)) {
389             $columns = $this->
>fieldStorageDefinitions[$field_name]→getColumns();
390             $column = reset($columns);
391             if (empty($column['serialize'])) {
392                 $values[$id][$field_name] = $record→{ $column_name };
393             }
394             unset($record→{ $column_name });
395         }
396     }
397 }
398
399     // Handle additional record entries that are not provided by
an entity
400     // field, such as 'isDefaultRevision'.
401     foreach ($record as $name => $value) {
402         $values[$id][$name] = $value;
403     }
404 }
405
406     // Load values from shared and dedicated tables.
407     $this→loadFromSharedTables($values);
408     $this→loadFromDedicatedTables($values);
409
410     return $values;
411 }
412
413 /**
414  * Loads values for fields stored in the shared data tables.
415  *
416  * @param array &$values
417  *   Associative array of entities values, keyed on the entity
ID or the
418  *   revision ID.
419  *
420  * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::loadFromSharedTables
421  */
422 protected function loadFromSharedTables(array &$values) {
423     $record_key = $this→idKey;
424     if ($this→dataTable) {
425         $table = $this→dataTable;

```

```
426     $alias = 'data';
427     $query = $this->database->select($table, $alias, ['fetch' =>
\PDO::FETCH_ASSOC])
428         ->fields($alias)
429         ->condition($alias . '.' . $record_key,
array_keys($values), 'IN')
430         ->orderBy($alias . '.' . $record_key);
431
432     $table_mapping = $this->getTableMapping();
433     $all_fields = $table_mapping->getFieldNames($this-
>dataTable);
434     if ($this->limitFields) {
435         $all_fields = array_values(array_intersect($all_fields,
$this->fields));
436     }
437
438     $result = $query->execute();
439     foreach ($result as $row) {
440         $id = $row[$record_key];
441
442         foreach ($all_fields as $field_name) {
443             $storage_definition = $this-
>fieldStorageDefinitions[$field_name];
444             $definition_columns = $storage_definition->getColumns();
445             $columns = $table_mapping->getColumnNames($field_name);
446             // Do not key single-column fields by property name.
447             if (count($columns) == 1) {
448                 $column_name = reset($columns);
449                 $column_attributes =
$definition_columns[key($columns)];
450                 if (empty($column_attributes['serialize'])) {
451                     $values[$id][$field_name] = $row[$column_name];
452                 }
453             }
454             else {
455                 foreach ($columns as $property_name => $column_name) {
456                     $column_attributes =
$definition_columns[$property_name];
457                     if (empty($column_attributes['serialize'])) {
458                         $values[$id][$field_name][$property_name] =
$row[$column_name];
459                     }
460                 }

```

```

461     }
462   }
463 }
464 }
465 }
466
467 /**
468  * Loads values of fields stored in dedicated tables for a group
of entities.
469  *
470  * @param array &$values
471  *   An array of values keyed by entity ID.
472  *
473  * @see
\Drupal\Core\Entity\Sql\SqlContentEntityStorage::loadFromDedicatedTables
474  */
475 protected function loadFromDedicatedTables(array &$values) {
476   if (empty($values)) {
477     return;
478   }
479
480   // Collect entities ids, bundles and languages.
481   $bundles = [];
482   $ids = [];
483   foreach ($values as $key => $entity_values) {
484     $bundles[$this->bundleKey ? $entity_values[$this->bundleKey] : $this->entityTypeId] = TRUE;
485     $ids[] = $key;
486   }
487
488   // Collect impacted fields.
489   $storage_definitions = [];
490   $definitions = [];
491   $table_mapping = $this->getTableMapping();
492   foreach ($bundles as $bundle => $v) {
493     $definition = $this->entityFieldManager->getFieldDefinitions($this->entityTypeId, $bundle);
494     if ($this->limitFields) {
495       $definition = array_intersect_key($definition, array_flip($this->fields));
496     }
497     $definitions[$bundle] = $definition;
498     foreach ($definitions[$bundle] as $field_name =>

```

```

$field_definition) {
499     $storage_definition = $field_definition-
>getFieldStorageDefinition();
500     if ($table_mapping-
>requiresDedicatedTableStorage($storage_definition)) {
501         $storage_definitions[$field_name] = $storage_definition;
502     }
503 }
504 }
505
506 // Load field data.
507 foreach ($storage_definitions as $field_name =>
$storage_definition) {
508     $table = $table_mapping-
>getDedicatedDataTableName($storage_definition);
509
510     $results = $this->database->select($table, 't')
511         ->fields('t')
512         ->condition('entity_id', $ids, 'IN')
513         ->condition('deleted', 0)
514         ->orderBy('delta')
515         ->execute();
516
517     foreach ($results as $row) {
518         $value_key = $row->entity_id;
519         if (!isset($values[$value_key][$field_name])) {
520             $values[$value_key][$field_name] = [];
521         }
522
523         if ($storage_definition->getCardinality() ==
FieldStorageDefinitionInterface::CARDINALITY_UNLIMITED ||
count($values[$value_key][$field_name]) < $storage_definition-
>getCardinality()) {
524             $item = [];
525             // For each column declared by the field, populate the
item from the
526             // prefixed database column.
527             foreach ($storage_definition->getColumns() as $column =>
$attributes) {
528                 if (!empty($attributes['serialize'])) {
529                     continue;
530                 }
531                 $column_name = $table_mapping-

```

```
>getFieldColumnName($storage_definition, $column);
532     $item[$column] = $row->$column_name;
533     }
534
535     // Add the item to the field values for the entity.
536     $values[$value_key][$field_name][] = $item;
537     }
538
539     }
540 }
541 }
542
543 }
544
```