SPARQL queries used in Linked Data generation for the NorwegianSoE dataset

SPARQL queries are used to transform the source datasets to RDF and they are also used for data augmentation.

1. RDF transformation for the Norwegian SoE dataset using SPARQL queries

The table below lists up the SPARQL queries that are used to transform the source datasets used for generating the NorwegianSoE dataset.

Table 1: Transformation script files for the involved datasets

Transformation scripts	Source format	Output, triples	Public
The central government organization transformation[1]	CSV	5049	Yes
The cadastral parcel ownership transformation[2]	CSV	559743	No
The cadastral parcel geospatial dataset transformation[3]	Shapefile	~118 M	No
Buildings built on the state-owned or state-leased cadastral parcels transformation[4]	CSV	454192	No
The building geospatial dataset transformation[5]	Shapefile	23188270	No
The building accessibility dataset transformation[6]	CSV	19285	Yes
The previous SoE dataset transformation[7]	CSV	663219	Yes
The municipality boundaries dataset transformation[8]	Shapefile	3424	Yes

2. Data augmentation for the Norwegian SoE dataset using SPARQL queries

The table below lists up the SPARQL CONSTRUCT queries that are used to generate the state-owned or state-leased cadastral parcels, buildings and also to enrich the datasets with new attributes.

Table 2: SPARQL CONSTRUCT queries for data augmentation

CONSTRUCT queries	Function	Result Dataset
Query #1[9]	indicate state-owned or state-leased cadastral parcels	-
Query #2[10]	select ownership information about state-owned or state-leased cadastral parcels	The state-owned or state-leased cadastral parcel ownership dataset
Query #3[11]	select geospatial information about state-owned or state-leased cadastral	The state-owned or state-leased cadastral parcel geospatial dataset
Query #4[12]	calculate the area of each cadastral parcel as summary of the belonging land parcels	The state-owned or state-leased cadastral parcel areas dataset
Query #5[13]	select information about buildings built on state-owned or state-leased cadastral parcels	State-owned building dataset
Query #6[14]	generate ownership dataset for state-owned buildings	State-owned building ownership dataset

[1] https://datagraft.io/prodatamarket_publisher/transformations/the-central-government-organization-transformation-1010c106-2254-4c8b-9480-b88d47a41323

[2] https://datagraft.io/prodatamarket_publisher/transformations/the-cadastral-parcel-ownership-transformation

[3] https://datagraft.io/prodatamarket_publisher/transformations/the-cadastral-parcel-geospatial-dataset-transformation

[4] https://datagraft.io/prodatamarket_publisher/transformations/buildings-built-on-the-state-owned-or-state-leased-cadastral-parcels-transformation

[5] https://datagraft.io/prodatamarket_publisher/transformations/the-building-geospatial-dataset-transformation

[6] https://datagraft.io/prodatamarket_publisher/transformations/the-building-accessibility-dataset-transformation

[7] https://datagraft.io/prodatamarket_publisher/transformations/the-historical-soe-dataset-transformation

[8] https://datagraft.io/prodatamarket_publisher/transformations/the-munincipalitiy-boundaries-dataset-trans-formation

[9] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-1

[10] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-2

[11] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-3

[12] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-4

[13] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-5

[14] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-6

Query #7[15]	generate geospatial dataset of state-owned buildings	State-owned building geospatial dataset

3. Details of the transformation scripts

Nr	Transformation scripts
1	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def RightsHolderOrganization (prefixer "http://www.datagraft.net/prodm/RightsHolderOrganization/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:RightsHolderOrganization (prodm-cad "RightsHolderOrganization")) (def schema (prefixer "http://schema.org/")) (def schema:leiCode (schema "leiCode")) (def schema:legalName (schema "legalName")) (def schema:foundingDate (schema "foundingDate")) (def dbo (prefixer "http://dbpedia.org/ontology/")) (def dbo:type (dbo "type")) (def schema:numberOfEmployees (schema "numberOfEmployees")) (def schema:location (schema "location")) (def schema:streetAddress (schema "streetAddress")) (def schema:PostalAddress (schema "PostalAddress")) (def schema:postalCode (schema "postalCode")) (def schema:addressCountry (schema "addressCountry")) (def schema:Country (schema "Country")) (def prodm (prefixer "http://vocabs.datagraft.net/proDataMarket#")) (def prodm:hasName (prodm "hasName")) (def schema:address (schema "address")) (def schema:postOfficeBoxNumber (schema "postOfficeBoxNumber")) (def schema:telephone (schema "telephone")) (def schema:parentOrganization (schema "parentOrganization")) (def prodm-cad:RightsHolder (prodm-cad "RightsHolder")) (def schema:Organization (schema "Organization")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (def make-graph (graph-fn [{:keys [orgnr navn stiftelsesdato organisasjonsform ansatte_antall forretningsadr forradrpostnr forradrland postadresse ppostnr ppostland tlf tlf_mobil hovedenhet]}] (graph "http://www.datagraft.net/prodm/" (if (not-empty? stiftelsesdato) [(RightsHolderOrganization orgnr) [schema:foundingDate (datatypes/convert-literal stiftelsesdato "date")]]) (if (not-empty? tlf) [(RightsHolderOrganization orgnr) [schema:telephone (datatypes/convert-literal tlf "string")]]) (if (not-empty? tlf_mobil) [(RightsHolderOrganization orgnr) [schema:telephone (datatypes/convert-literal tlf_mobil "string")]]) (if (not-empty? hovedenhet) [(RightsHolderOrganization orgnr) [schema:parentOrganization (RightsHolderOrganization hovedenhet)]]) [(RightsHolderOrganization orgnr) [rdf:a prodm-cad:RightsHolderOrganization] [schema:leiCode (datatypes/convert-literal orgnr "string")] [schema:legalName (datatypes/convert-literal navn "string")] [dbo:type (datatypes/convert-literal organisasjonsform "string")] [schema:numberOfEmployees (datatypes/convert-literal ansatte_antall "integer")] [schema:location [[schema:streetAddress (datatypes/convert-literal forretningsadr "string")] [rdf:a schema:PostalAddress] [schema:postalCode (datatypes/convert-literal forradrpostnr "string")] [schema:addressCountry [[rdf:a schema:Country] [prodm:hasName (datatypes/convert-literal forradrland "string")]]]]] [schema:address [[schema:postOfficeBoxNumber (datatypes/convert-literal

[15] https://datagraft.io/prodatamarket_publisher/queries/soe-construct-query-7

	postadresse "string"))] [rdf:a schema:PostalAddress] [schema:postalCode (datatypes/convert-literal ppostnr "string")] [schema:addressCountry [[rdf:a schema:Country] [prodm:hasName (datatypes/convert-literal ppostland "string")]]]]]] [prodm-cad:RightsHolderOrganization [rdfs:subClassOf prodm-cad:RightsHolder] [rdfs:subClassOf schema:Organization]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
2	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def CadastralParcel (prefixer "http://www.datagraft.net/prodm/CadastralParcel/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:CadastralParcel (prodm-cad "CadastralParcel")) (def prodm-cad:hasCadastralParcelID (prodm-cad "hasCadastralParcelID")) (def prodm-cad:hasCadastralUnitNumber (prodm-cad "hasCadastralUnitNumber")) (def prodm-cad:hasPropertyUnitNumber (prodm-cad "hasPropertyUnitNumber")) (def prodm-cad:hasLeaseholdNumber (prodm-cad "hasLeaseholdNumber")) (def prodm-cad:hasCondominiumUnitNumber (prodm-cad "hasCondominiumUnitNumber")) (def prodm-cad:hasCadastralID (prodm-cad "hasCadastralID")) (def cp (prefixer "http://www.w3.org/2015/03/inspire/cp#")) (def cp:nationalCadastralReference (cp "nationalCadastralReference")) (def dcterms:coverage (dcterms "coverage")) (def AdministrativeUnit (prefixer "http://www.datagraft.net/prodm/AdministrativeUnit/")) (def au (prefixer "http://www.w3.org/2015/03/inspire/au#")) (def au:AdministrativeUnit (au "AdministrativeUnit")) (def au:country (au "country")) (def mdr-eu (prefixer "http://publications.europa.eu/resource/authority/country/")) (def mdr-eu:NOR (mdr-eu "NOR")) (def au:nationalCode (au "nationalCode")) (def au:nationalLevel (au "nationalLevel")) (def inspire-hierarchy (prefixer "http://inspire.ec.europa.eu/codelist/AdministrativeHierarchyLevel/")) (def inspire-hierarchy:3rdOrder (inspire-hierarchy "3rdOrder")) (def RealRights (prefixer "http://www.datagraft.net/prodm/RealRights/")) (def prodm-cad:RealRights (prodm-cad "RealRights")) (def dbo (prefixer "http://dbpedia.org/ontology/")) (def dbo:type (dbo "type")) (def dul (prefixer "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#")) (def dul:defines (dul "defines")) (def RightsHolderOrganization (prefixer "http://www.datagraft.net/prodm/RightsHolderOrganization/")) (def dcterms:source (dcterms "source")) (def prodm-cad:hasNumerator (prodm-cad "hasNumerator")) (def prodm-cad:hasDenominator (prodm-cad "hasDenominator")) (def prodm-cad:hasStartDate (prodm-cad "hasStartDate")) (def prodm-cad:hasEndDate (prodm-cad "hasEndDate")) (def dce (prefixer "http://purl.org/dc/elements/1.1/")) (def dce:Rights (dce "Rights")) (def schema (prefixer "https://schema.org/")) (def schema:leiCode (schema "leiCode")) (def prodm-com (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Common#")) (def prodm-com:hasAlternativeName (prodm-com "hasAlternativeName")) (def AlternativeName (prefixer "http://www.datagraft.net/prodm/AlternativeName/")) (def prodm-com:AlternativeName (prodm-com "AlternativeName")) (def prodm-cad:RightsHolderOrganization (prodm-cad "RightsHolderOrganization")) (def prodm-cad:RightsHolder (prodm-cad "RightsHolder")) (def schema:Organization (schema "Organization")) (def owl:sameAs (owl "sameAs")) (def gn (prefixer "http://sws.geonames.org/")) (def gn:3144096 (gn "3144096")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N")

	"gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (defn pad "Pad string istr with val upto length n" [istr val n] (str (apply str (take (- (Integer/parseInt n) (count istr)) (repeat val))) istr)) (defn rights-type "" [rolle] (subs rolle 3)) (defn reformat-dates [d] (let [arg (if (empty? d) "01.01.1753" d)] (.toDate (clj-time.format/parse (clj-time.format/formatter (clj-time.core/time-zone-for-offset 0) "dd.MM.yyyy" "dd.MM.yyyy HH:mm" "dd.MM.yyyy HH:mm:ss") arg)))) (defn cad-ref "" [komm gnr bnr fnr snr] (str komm "/" gnr "/" bnr "/" fnr "/" snr)) (defn cad-ref-id "" [komm gnr bnr fnr snr] (str komm gnr bnr fnr snr)) (defn rr-id "" [cad-ref-id owner] (str cad-ref-id owner)) (def rights-type-en {"HJEMMELSHAVER" "OWNER" "FESTER" "LESSOR"}) (defn au-komm "" [komm] (str "NOR3-" komm)) (def make-graph (graph-fn [{:keys [cad-ref cad-ref-id rr-id rolle-en au-komm ENUMMER GNR BNR FNR SNR MATRIKKELID KOMMUNEID ROLLE NR TELLER NEVNER DATOFRA DATOTIL NAVN]}] (graph "http://www.datagraft.net/prodm/" [(CadastralParcel cad-ref-id) [rdf:a prodm-cad:CadastralParcel] [prodm-cad:hasCadastralParcelID (datatypes/convert-literal ENUMMER "string")] [prodm-cad:hasCadastralUnitNumber (datatypes/convert-literal GNR "string")] [prodm-cad:hasPropertyUnitNumber (datatypes/convert-literal BNR "string")] [prodm-cad:hasLeaseholdNumber (datatypes/convert-literal FNR "string")] [prodm-cad:hasCondominiumUnitNumber (datatypes/convert-literal SNR "string")] [prodm-cad:hasCadastralID (datatypes/convert-literal MATRIKKELID "string")] [cp:nationalCadastralReference (datatypes/convert-literal cad-ref "string")] [dcterms:coverage (AdministrativeUnit au-komm)]] [(AdministrativeUnit au-komm) [rdf:a au:AdministrativeUnit] [au:country mdr-eu:NOR] [au:nationalCode (datatypes/convert-literal KOMMUNEID "string")] [au:nationalLevel inspire-hierarchy:3rdOrder]] [(RealRights rr-id) [rdf:a prodm-cad:RealRights] [dbo:type (datatypes/convert-literal ROLLE "string" :lang-tag "no")] [dbo:type (datatypes/convert-literal rolle-en "string" :lang-tag "en")] [dul:defines (CadastralParcel cad-ref-id)] [dul:defines (RightsHolderOrganization NR)] [dcterms:source (s "cadaster")] [prodm-cad:hasNumerator (datatypes/convert-literal TELLER "integer")] [prodm-cad:hasDenominator (datatypes/convert-literal NEVNER "integer")] [prodm-cad:hasStartDate DATOFRA] [prodm-cad:hasEndDate DATOTIL]] [prodm-cad:RealRights [rdfs:subClassOf dce:Rights]] [(RightsHolderOrganization NR) [rdf:a (s "prodm-cad:RightsHolderOrganization")] [schema:leiCode (datatypes/convert-literal NR "string")] [prodm-com:hasAlternativeName (AlternativeName NR)]] [(AlternativeName NR) [rdf:a prodm-com:AlternativeName] [prodm-com:hasAlternativeName (datatypes/convert-literal NAVN "string")] [dcterms:source (s "cadaster")]] [prodm-cad:RightsHolderOrganization [rdfs:subClassOf prodm-cad:RightsHolder] [rdfs:subClassOf schema:Organization]] [mdr-eu:NOR [owl:sameAs gn:3144096]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (mapc {:KOMMUNEID (fn [arg] (pad arg "0" "4")) :ROLLE rights-type :DATOFRA reformat-dates :DATOTIL reformat-dates}) (derive-column :cad-ref [:KOMMUNEID :GNR :BNR :FNR :SNR] cad-ref) (derive-column :cad-ref-id [:KOMMUNEID :GNR :BNR :FNR :SNR] cad-ref-id) (derive-column :rr-id [:cad-ref-id :NR] rr-id) (mapc {:ROLLE trim}) (derive-column :rolle-en [:ROLLE] rights-type-en) (derive-column :au-komm [:KOMMUNEID] au-komm))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
3	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def LandParcel (prefixer "http://www.datagraft.net/prodm/LandParcel/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:LandParcel (prodm-cad "LandParcel")) (def prodm-cad:hasLandParcelID (prodm-cad "hasLandParcelID")) (def dul (prefixer "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#")) (def dul:part (dul "part")) (def CadastralParcel (prefixer "http://www.datagraft.net/prodm/CadastralParcel/")) (def prodm-com (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Common#")) (def prodm-com:hasAreaMeasurement (prodm-com "hasAreaMeasurement")) (def LandParcelAreaMeasurement (prefixer "http://www.datagraft.net/prodm/LandParcelAreaMeasurement/")) (def gsp (prefixer "http://www.opengis.net/ont/geosparql#")) (def gsp:hasGeometry (gsp "hasGeometry")) (def LandParcelGeometry (prefixer "http://www.datagraft.net/prodm/LandParcelGeometry/")) (def prodm-com:AreaMeasurement (prodm-com "AreaMeasurement")) (def rdf:value (rdf "value")) (def dbr (prefixer "http://dbpedia.org/resource/")) (def dbr:Square_metre (dbr "Square_metre")) (def sf (prefixer "http://www.opengis.net/ont/sf#")) (def sf:MultiPolygon (sf "MultiPolygon")) (def gsp:asWKT (gsp "asWKT")) (def prodm-cad:CadastralParcel (prodm-cad "CadastralParcel")) (def prodm-cad:hasLandParcel (prodm-cad "hasLandParcel")) (def prodm-cad:hasCadastralParcelID (prodm-cad "hasCadastralParcelID")) (def prodm-cad:hasCadastralUnitNumber (prodm-cad "hasCadastralUnitNumber")) (def prodm-cad:hasPropertyUnitNumber (prodm-cad "hasPropertyUnitNumber"))

```
(def prodm-cad:hasLeaseholdNumber (prodm-cad "hasLeaseholdNumber"))
(def prodm-cad:hasCondominiumUnitNumber (prodm-cad "hasCondominiumUnitNumber"))
(def prodm-cad:hasCadastralID (prodm-cad "hasCadastralID"))
(def cp (prefixer "http://www.w3.org/2015/03/inspire/cp#"))
(def cp:nationalCadastralReference (cp "nationalCadastralReference"))
(def dcterms:coverage (dcterms "coverage"))
(def AdministrativeUnit (prefixer "http://www.datagraft.net/prodm/AdministrativeUnit/"))
(def prodm-com:hasIndicator (prodm-com "hasIndicator"))
(def ParcelCulturalHeritageIndicator (prefixer "http://www.datagraft.net/prodm/ParcelCulturalHeritageIndicator/"))
(def ParcelGroundPollutionIndicator (prefixer "http://www.datagraft.net/prodm/ParcelGroundPollutionIndicator/"))
(def prodm-com:Indicator (prodm-com "Indicator"))
(def dbo (prefixer "http://dbpedia.org/ontology/"))
(def dbo:type (dbo "type"))
(def au (prefixer "http://www.w3.org/2015/03/inspire/au#"))
(def au:AdministrativeUnit (au "AdministrativeUnit"))
(def au:country (au "country"))
(def au:nationalCode (au "nationalCode"))
(def au:nationalLevel (au "nationalLevel"))
(def inspire-hierarchy (prefixer "http://inspire.ec.europa.eu/codelist/AdministrativeHierarchyLevel/"))
(def inspire-hierarchy:3rdOrder (inspire-hierarchy "3rdOrder"))

(defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s)))
(defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s)))
(defn join [& strings] (clojure.string/join " " strings))
(defn join-with [sep] (fn [& strings] (clojure.string/join sep strings)))
(defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date
(read-string "#\"/\"") )] (apply str (interpose "-" [y m d])))))
(defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " "")))
(defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-
string "#\"[0-9]{4} \"") "")))
(def string-literal s)
(defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x))))
(defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c
(interpose " ") (apply str) clojure.string/trim))))
(def transform-gender {"f" (s "female") "m" (s "male")})
(defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (.
gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N")
"gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong
hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees
(.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords))))))
(defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply
clojure.string/replace arg1 arg2)) content replacement-list)))
(defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-
several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere
zoneNumber)) (partition 2 all-coords))))))
(defn fill-when [col] (grafter.sequences/fill-when col))
(def not-empty? (complement empty?))
(defn cad-ref "" [komm gnr bnr fnr snr] (str komm "/" gnr "/" bnr "/" fnr "/" snr))
(defn cad-ref-id "" [komm gnr bnr fnr snr] (str komm gnr bnr fnr snr))
(defn pad "Pad string istr with val upto length n" [istr val n] (str (apply str (take (- (Integer/parseInt n) (count istr)) (repeat val)))
istr))
(defn to-double "" [x] (cond (empty? x) (double 0) :else (-> x (Double/parseDouble) (double))))
(defn au-komm "" [komm] (str "NOR3-" komm))
(def make-graph (graph-fn [{:keys [au-komm nationalCadastralReference cad-ref-id TEIGID AREAL WKT ENUMMER GNR
BNR FNR SNR MATRIKKELI HARKULTURM HARGRUNNFO KOMMUNEID]}] (graph "http://www.datagraft.net/prodm/"
[(LandParcel TEIGID) [rdf:a prodm-cad:LandParcel] [prodm-cad:hasLandParcelID (datatypes/convert-literal TEIGID "string")]
[dul:part (CadastralParcel cad-ref-id)] [prodm-com:hasAreaMeasurement (LandParcelAreaMeasurement TEIGID)]
[gsp:hasGeometry (LandParcelGeometry TEIGID)]] [(LandParcelAreaMeasurement TEIGID) [rdf:a prodm-
com:AreaMeasurement] [rdf:value AREAL] [sdmx-attribute:unitMeasure dbr:Square_metre]] [(LandParcelGeometry TEIGID)
[rdf:a sf:MultiPolygon] [gsp:asWKT (s WKT (org.openrdf.model.impl.URIImpl.
"http://www.opengis.net/ont/geosparql#wktLiteral"))]] [(CadastralParcel cad-ref-id) [rdf:a prodm-cad:CadastralParcel] [prodm-
cad:hasLandParcel (LandParcel TEIGID)] [prodm-cad:hasCadastralParcelID (datatypes/convert-literal ENUMMER "string")]
[prodm-cad:hasCadastralUnitNumber (datatypes/convert-literal GNR "string")] [prodm-cad:hasPropertyUnitNumber
(datatypes/convert-literal BNR "string")] [prodm-cad:hasLeaseholdNumber (datatypes/convert-literal FNR "string")] [prodm-
cad:hasCondominiumUnitNumber (datatypes/convert-literal SNR "string")] [prodm-cad:hasCadastralID (datatypes/convert-literal
MATRIKKELI "string")] [cp:nationalCadastralReference (datatypes/convert-literal nationalCadastralReference "string")]
[dcterms:coverage (AdministrativeUnit au-komm)] [prodm-com:hasIndicator (ParcelCulturalHeritageIndicator cad-ref-id)] [prodm-
com:hasIndicator (ParcelGroundPollutionIndicator cad-ref-id)]] [(ParcelCulturalHeritageIndicator cad-ref-id) [rdf:a prodm-
com:Indicator] [dbo:type (s "CulturalHeritage")] [rdf:value HARKULTURM]] [(ParcelGroundPollutionIndicator cad-ref-id) [rdf:a
prodm-com:Indicator] [dbo:type (s "GroundPollution")] [rdf:value HARGRUNNFO]] [(AdministrativeUnit au-komm) [rdf:a
au:AdministrativeUnit] [au:country (s "NOR")] [au:nationalCode (datatypes/convert-literal KOMMUNEID "string")]
[au:nationalLevel inspire-hierarchy:3rdOrder]])))

(defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset
move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (mapc {:KOMMUNEID (fn [arg]
(pad arg "0" "4")) :AREAL to-double :HARKULTURM to-double :HARGRUNNFO to-double}) (derive-column :au-komm
```

	[:KOMMUNEID] au-komm) (derive-column :nationalCadastralReference [:KOMMUNEID :GNR :BNR :FNR :SNR] cad-ref) (derive-column :cad-ref-id [:KOMMUNEID :GNR :BNR :FNR :SNR] cad-ref-id))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
4	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def Building (prefixer "http://www.datagraft.net/prodm/Building/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:Building (prodm-cad "Building")) (def prodm-cad:hasCadastralBuildingNumber (prodm-cad "hasCadastralBuildingNumber")) (def prodm-cad:isBuiltOn (prodm-cad "isBuiltOn")) (def CadastralParcel (prefixer "http://www.datagraft.net/prodm/CadastralParcel/")) (def dbo (prefixer "http://dbpedia.org/ontology/")) (def dbo:type (dbo "type")) (def dbo:status (dbo "status")) (def prodm-com (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Common#")) (def prodm-com:hasAreaMeasurement (prodm-com "hasAreaMeasurement")) (def BuildingAreaMeasurement (prefixer "http://www.datagraft.net/prodm/BuildingAreaMeasurement/")) (def prodm-com:hasIndicator (prodm-com "hasIndicator")) (def BuildingCulturalHeritageIndicator (prefixer "http://www.datagraft.net/prodm/BuildingCulturalHeritageIndicator/")) (def SEFRAKIndicator (prefixer "http://www.datagraft.net/prodm/SEFRAKIndicator/")) (def prodm-com:AreaMeasurement (prodm-com "AreaMeasurement")) (def rdf:value (rdf "value")) (def dbr (prefixer "http://dbpedia.org/resource/")) (def dbr:Square_metre (dbr "Square_metre")) (def prodm-com:Indicator (prodm-com "Indicator")) (def prodm-cad:CadastralParcel (prodm-cad "CadastralParcel")) (def prodm-cad:hasBuilding (prodm-cad "hasBuilding")) (def dul (prefixer "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#")) (def dul:part (dul "part")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (defn cad-ref "" [komm gnr bnr fnr snr] (str komm "/" gnr "/" bnr "/" fnr "/" snr)) (defn pad "Pad string istr with val upto length n" [istr val n] (str (apply str (take (- (Integer/parseInt n) (count istr)) (repeat val))) istr)) (defn to-double "" [x] (cond (empty? x) (double 0) :else (-> x (clojure.string/replace "," ".") (Double/parseDouble) (double)))) (defn au-komm "" [komm] (str "NOR3-" komm)) (defn cad-ref-id "" [komm gnr bnr fnr snr] (str komm gnr bnr fnr snr)) (defn get-cad-ref-id "" [gid snr] (let [matrikkelid (if (= (-> gid (clojure.string/split (read-string "#\"/\"")) (first) (count)) 4) gid (str "0" gid))] (clojure.string/replace (str matrikkelid snr) "/" ""))) (defn get-cad-ref-id-snr0 "" [gid] (let [matrikkelid (if (= (-> gid (clojure.string/split (read-string "#\"/\"")) (first) (count)) 4) gid (str "0" gid))] (clojure.string/replace (str matrikkelid "0") "/" ""))) (def make-graph (graph-fn [{:keys [cad-ref-id cad-ref-id-snr0 BYGNINGSNR BYGGTYPE STATUS TOTALAREAL HARKULTURMINNE HARSEFRAKMINNE ENUMMERSNR0]}] (graph "http://www.datagraft.net/prodm/" [(Building BYGNINGSNR) [rdf:a prodm-cad:Building] [prodm-cad:hasCadastralBuildingNumber (datatypes/convert-literal BYGNINGSNR "string")] [prodm-cad:isBuiltOn (CadastralParcel cad-ref-id)] [dbo:type (datatypes/convert-literal BYGGTYPE "string")] [dbo:status (datatypes/convert-literal STATUS "string")] [prodm-com:hasAreaMeasurement (BuildingAreaMeasurement BYGNINGSNR)] [prodm-com:hasIndicator (BuildingCulturalHeritageIndicator BYGNINGSNR)] [prodm-com:hasIndicator (SEFRAKIndicator BYGNINGSNR)]] [(BuildingAreaMeasurement BYGNINGSNR) [rdf:a prodm-com:AreaMeasurement] [rdf:value TOTALAREAL] [sdmx-attribute:unitMeasure dbr:Square_metre]] [(BuildingCulturalHeritageIndicator BYGNINGSNR)

	[rdf:a prodm-com:Indicator] [dbo:type (s "CulturalHeritage")] [rdf:value HARKULTURMINNE]] [(SEFRAKIndicator BYGNINGSNR) [rdf:a prodm-com:Indicator] [dbo:type (s "SEFRAK")] [rdf:value HARSEFRAKMINNE]] (if (not-empty? ENUMMERSNR0) [(CadastralParcel cad-ref-id) [dul:part (CadastralParcel cad-ref-id-snr0)]]) [(CadastralParcel cad-ref-id) [rdf:a prodm-cad:CadastralParcel] [prodm-cad:hasBuilding (Building BYGNINGSNR)]] (if (not-empty? ENUMMERSNR0) [(CadastralParcel cad-ref-id-snr0) [rdf:a prodm-cad:CadastralParcel] [prodm-cad:hasBuilding (Building BYGNINGSNR)]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (columns (range 0 15)) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (derive-column :cad-ref-id [:MATRIKKELID :SNR] get-cad-ref-id) (derive-column :cad-ref-id-snr0 [:MATRIKKELID] get-cad-ref-id-snr0) (mapc {:TOTALAREAL to-double :HARKULTURMINNE to-double :HARSEFRAKMINNE to-double}))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
5	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def Building (prefixer "http://www.datagraft.net/prodm/Building/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:Building (prodm-cad "Building")) (def prodm-cad:hasCadastralBuildingNumber (prodm-cad "hasCadastralBuildingNumber")) (def gsp (prefixer "http://www.opengis.net/ont/geosparql#")) (def gsp:hasGeometry (gsp "hasGeometry")) (def BuildingGeometry (prefixer "http://www.datagraft.net/prodm/BuildingGeometry/")) (def sf (prefixer "http://www.opengis.net/ont/sf#")) (def sf:Point (sf "Point")) (def gsp:asWKT (gsp "asWKT")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (def make-graph (graph-fn [{:keys [BYGNINGSNR WKT]}] (graph "http://www.datagraft.net/prodm/" [(Building BYGNINGSNR) [rdf:a prodm-cad:Building] [prodm-cad:hasCadastralBuildingNumber (datatypes/convert-literal BYGNINGSNR "string")] [gsp:hasGeometry (BuildingGeometry BYGNINGSNR)]] [(BuildingGeometry BYGNINGSNR) [rdf:a sf:Point] [gsp:asWKT (s WKT (org.openrdf.model.impl.URIImpl. "http://www.opengis.net/ont/geosparql#wktLiteral"))]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
6	(def graph0 (prefixer "http://www.example.no/#/")) (def BuildingSoE (prefixer "http://www.datagraft.net/prodm/Building/Soe/")) (def prodm-soe (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/SoE#")) (def prodm-soe:Building (prodm-soe "Building")) (def dcterms:identifier (dcterms "identifier")) (def Building (prefixer "http://www.datagraft.net/prodm/Building/")) (def prodm-cad (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#")) (def prodm-cad:Building (prodm-cad "Building")) (def prodm-cad:hasCadastralBuildingNumber (prodm-cad "hasCadastralBuildingNumber")) (def prodm-com (prefixer "http://vocabs.datagraft.net/proDataMarket/0.1/Common#")) (def prodm-com:hasIndicator (prodm-com "hasIndicator")) (def BuildingNumberOfFloorsIndicator (prefixer "http://www.datagraft.net/prodm/BuildingNumberOfFloorsIndicator/")) (def BuildingStepFreeMainEntranceIndicator (prefixer "http://www.datagraft.net/prodm/BuildingStepFreeMainEntranceIndicator/"))

	(def BuildingStepFreeSideEntranceIndicator (prefixer "http://www.datagraft.net/prodm/BuildingStepFreeSideEntranceIndicator/")) (def BuildingElevatorIndicator (prefixer "http://www.datagraft.net/prodm/BuildingElevatorIndicator/")) (def BuildingHCToiletIndicator (prefixer "http://www.datagraft.net/prodm/BuildingHCToiletIndicator/")) (def BuildingHCParkingIndicator (prefixer "http://www.datagraft.net/prodm/BuildingHCParkingIndicator/")) (def prodm-com:Indicator (prodm-com "Indicator")) (def dbo (prefixer "http://dbpedia.org/ontology/")) (def dbo:type (dbo "type")) (def rdf:value (rdf "value")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (defn map-indicators "" [ind] (if (= ind "Nei") 0 1)) (def make-graph (graph-fn [{:keys [Byggnr Byggnavn GAB-nummer Antall_etasjer Trinnfri_hovedatkomst Trinnfri_sideatkomst Heis HC-toalett HC-parkering]}] (graph "http://www.example.no/#/" [(BuildingSoE Byggnr) [rdf:a prodm-soe:Building] [dcterms:identifier (datatypes/convert-literal Byggnr "string")] [rdfs:label (datatypes/convert-literal Byggnavn "string")]] [(Building GAB-nummer) [rdf:a prodm-cad:Building] [prodm-cad:hasCadastralBuildingNumber (datatypes/convert-literal GAB-nummer "string")] [prodm-com:hasIndicator (BuildingNumberOfFloorsIndicator GAB-nummer)] [prodm-com:hasIndicator (BuildingStepFreeMainEntranceIndicator GAB-nummer)] [prodm-com:hasIndicator (BuildingStepFreeSideEntranceIndicator GAB-nummer)] [prodm-com:hasIndicator (BuildingElevatorIndicator GAB-nummer)] [prodm-com:hasIndicator (BuildingHCToiletIndicator GAB-nummer)] [prodm-com:hasIndicator (BuildingHCParkingIndicator GAB-nummer)]] [(BuildingNumberOfFloorsIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "NumberOfFloors")] [rdf:value (datatypes/convert-literal Antall_etasjer "double")]] [(BuildingStepFreeMainEntranceIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "StepFreeMainEntrance")] [rdf:value (datatypes/convert-literal Trinnfri_hovedatkomst "double")]] [(BuildingStepFreeSideEntranceIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "StepFreeSideEntrance")] [rdf:value (datatypes/convert-literal Trinnfri_sideatkomst "double")]] [(BuildingElevatorIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "Elevator")] [rdf:value (datatypes/convert-literal Heis "double")]] [(BuildingHCToiletIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "HCToilet")] [rdf:value (datatypes/convert-literal HC-toalett "double")]] [(BuildingHCParkingIndicator GAB-nummer) [rdf:a prodm-com:Indicator] [dbo:type (s "HCParking")] [rdf:value (datatypes/convert-literal HC-parkering "double")]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (mapc {:Trinnfri_hovedatkomst map-indicators :Trinnfri_sideatkomst map-indicators :Heis map-indicators :HC-toalett map-indicators :HC-parkering map-indicators}))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
7	(defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (.

	gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (defn shift-soe-recs [dataset] (letfn [(is-numeric [x] (not (nil? (re-matches (read-string "#\"^((-*[0-9]+)|((-*[0-9]+).([0-9]+)))$\"") (str x))))) (drop-nth [n coll] (concat (take n coll) (drop (inc n) coll))) (fix-row [row] (let [colnames (column-names dataset)] (->> (map row colnames) (drop-nth 4) (reverse) (apply conj ' ("")) (zipmap colnames))))] (let [colnames (column-names dataset) wrong-rows (-> (grep dataset (complement is-numeric) [:EIENDOMSNR]) (:rows)) correct-rows (-> (grep dataset is-numeric [:EIENDOMSNR]) (:rows)) fixed-rows (loop [fixed-rows () cnt 0] (if (= cnt (count wrong-rows)) fixed-rows (recur (conj fixed-rows (fix-row (nth wrong-rows cnt))) (inc cnt))))] (-> (make-dataset (:rows (incanter.core/conj-rows correct-rows fixed-rows)) (column-names dataset)) (with-meta (meta dataset)))))) (defn transform-text "Transforms text" [s] (-> s (clojure.string/replace "," "."))) (defn pad "Pad string istr with val upto length n" [istr val n] (str (apply str (take (- (Integer/parseInt n) (count istr)) (repeat val))) istr)) (defn empty-to-zero "" [x] (if (empty? x) "0" x)) (def rolle {"ANNET" "ANNET" "EID" "HJEMMELSHAVER" "FESTET" "FESTER" "FORVALTET" "FORVALTET" "GRUNNRETT" "GRUNNRETT" "IKKE EID" "IKKE EID" "KLAUSUL" "KLAUSUL" "LEID" "LEID" "SOLGT" "SOLGT"}) (defn reformat-dates [d] (let [arg (if-not (= (count d) 8) "17530101" d)] (.toDate (clj-time.format/parse (clj-time.format/formatter (clj-time.core/time-zone-for-offset 0) "dd.MM.yyyy" "yyyyMMdd") arg)))) (defn to-double "" [x] (cond (empty? x) (double 0) :else (-> x (Double/parseDouble) (double)))) (defn cad-ref "" [komm gnr bnr fnr snr] (str komm "/" gnr "/" bnr "/" fnr "/" snr)) (def make-graph (graph-fn [{:keys [cad-ref]}])) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (shift-soe-recs) (mapc {:BRUTTO_BTA_SUM transform-text :BYGGEIEFORHOLD upper-case}) (mapc {:KOMMUNE (fn [arg] (pad arg "0" "4")) :GNR empty-to-zero :BNR empty-to-zero :FNR empty-to-zero :SNR empty-to-zero :EIEFORHOLD rolle :ERVERVAAR reformat-dates :TOMTEAREAL to-double :BRUTTO_BTA_SUM to-double :BYGGEIEFORHOLD rolle}) (derive-column :cad-ref [:KOMMUNE :GNR :BNR :FNR :SNR] cad-ref))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)
8	(def graph0 (prefixer "http://www.datagraft.net/prodm/")) (def AdministrativeUnit (prefixer "http://www.datagraft.net/prodm/AdministrativeUnit/")) (def au (prefixer "http://www.w3.org/2015/03/inspire/au#")) (def au:AdministrativeUnit (au "AdministrativeUnit")) (def au:country (au "country")) (def au:nationalCode (au "nationalCode")) (def au:nationalLevel (au "nationalLevel")) (def inspire-hierarchy (prefixer "http://inspire.ec.europa.eu/codelist/AdministrativeHierarchyLevel/")) (def inspire-hierarchy:3rdOrder (inspire-hierarchy "3rdOrder")) (def gsp (prefixer "http://www.opengis.net/ont/geosparql#")) (def gsp:hasGeometry (gsp "hasGeometry")) (def MunicipalityGeometry (prefixer "http://www.datagraft.net/prodm/MunicipalityGeometry/")) (def sf (prefixer "http://www.opengis.net/ont/sf#")) (def sf:MultiPolygon (sf "MultiPolygon")) (def gsp:asWKT (gsp "asWKT")) (defn double-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Double/parseDouble s))) (defn integer-literal [s] (if (nil? (re-matches (read-string "#\"[0-9.]+\"") s)) 0 (Integer/parseInt s))) (defn join [& strings] (clojure.string/join " " strings)) (defn join-with [sep] (fn [& strings] (clojure.string/join sep strings))) (defn organize-date "Transform date dd/mm/yyyy ~> yyyy-mm-dd" [date] (when (seq date) (let [[d m y] (clojure.string/split date (read-string "#\"/\""))] (apply str (interpose "-" [y m d]))))) (defn remove-blanks [s] (when (seq s) (clojure.string/replace s " " ""))) (defn replace-varible-string [cell] (-> cell (clojure.string/replace (read-string "#\".* #\"") "number") (clojure.string/replace (read-string "#\"[0-9]{4} \"") ""))) (def string-literal s) (defn stringToNumeric [x] (if (= "" x) nil (if (.contains x ".") (Double/parseDouble x) (Integer/parseInt x)))) (defn titleize [st] (when (seq st) (let [a (clojure.string/split st (read-string "#\" \"")) c (map clojure.string/capitalize a)] (->> c (interpose " ") (apply str) clojure.string/trim)))) (def transform-gender {"f" (s "female") "m" (s "male")}) (defn get-lat-long-strings-replacement [easting northing hemisphere zoneNumber] (let [utmCoords (. gov.nasa.worldwind.geom.coords.UTMCoord fromUTM (Integer/parseInt zoneNumber) (if (= hemisphere "N") "gov.nasa.worldwind.avkey.North" (if (= hemisphere "S") "gov.nasa.worldwind.avkey.East" (throw (Exception. "Wrong hemisphere input")))) (Double/parseDouble easting) (Double/parseDouble northing))] (vector (re-pattern easting) (str (.getDegrees (.getLongitude utmCoords))) (re-pattern northing) (str (.getDegrees (.getLatitude utmCoords)))))) (defn replace-several [content replacements] (let [replacement-list (partition 2 replacements)] (reduce (fn [arg1 arg2] (apply

	clojure.string/replace arg1 arg2)) content replacement-list))) (defn convert-col-lat-long [col hemisphere zoneNumber] (let [all-coords (re-seq (re-pattern "-?[0-9]{1,13}.[0-9]+") col)] (replace-several col (flatten (map (fn [coord-pair] (get-lat-long-strings-replacement (nth coord-pair 0) (nth coord-pair 1) hemisphere zoneNumber)) (partition 2 all-coords)))))) (defn fill-when [col] (grafter.sequences/fill-when col)) (def not-empty? (complement empty?)) (defn pad "Pad string istr with val upto length n" [istr val n] (str (apply str (take (- (Integer/parseInt n) (count istr)) (repeat val))) istr)) (defn au-komm "" [komm] (str "NOR3-" komm)) (def make-graph (graph-fn [{:keys [au-komm Kom428 Navn WKT]}] (graph "http://www.datagraft.net/prodm/" [(AdministrativeUnit au-komm) [rdf:a au:AdministrativeUnit] [au:country (s "NOR")] [au:nationalCode (datatypes/convert-literal Kom428 "string")] [au:nationalLevel inspire-hierarchy:3rdOrder] [gsp:hasGeometry (MunicipalityGeometry au-komm)] [rdfs:label (datatypes/convert-literal Navn "string")]] [(MmunicipalityGeometry au-komm) [rdf:a sf:MultiPolygon] [gsp:asWKT (s WKT (org.openrdf.model.impl.URIImpl. "http://www.opengis.net/ont/geosparql#wktLiteral"))]]))) (defpipe my-pipe "Grafter pipeline for data clean-up and preparation." [data-file] (-> (read-dataset data-file) (-> (make-dataset move-first-row-to-header) (rename-columns (comp keyword new-tabular/string-as-keyword))) (mapc {:Kom428 (fn [arg] (pad arg "0" "4"))}) (derive-column :au-komm [:Kom428] au-komm))) (defgraft my-graft "Transformation that converts input CSV data into RDF graph data." my-pipe make-graph)

4. Details of the augmentation queries

Nr	Augmentation queries
1	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#> PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#> PREFIX schema: <http://schema.org/> PREFIX gsp: <http://www.opengis.net/ont/geosparql#> PREFIX sf: <http://www.opengis.net/ont/sf#> PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#> PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX dcterms: <http://purl.org/dc/terms/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> CONSTRUCT {?cp prodm-com:isStateOwned true .} WHERE { ?cp a prodm-cad:CadastralParcel; prodm-cad:hasLandParcel ?teig. ?rr a prodm-cad:RealRights; dul:defines ?cp; dul:defines ?org. ?org a prodm-cad:RightsHolderOrganization; schema:legalName ?legalName . }
2	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#>

```
PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#>
PREFIX schema: <http://schema.org/>
PREFIX gsp: <http://www.opengis.net/ont/geosparql#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#>
PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX au: <http://www.w3.org/2015/03/inspire/au#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
?cp a prodm-cad:CadastralParcel;
prodm-cad:hasCadastralParcelID ?id ;
prodm-cad:hasCadastralUnitNumber ?gnr ;
prodm-cad:hasPropertyUnitNumber ?bnr ;
prodm-cad:hasLeaseholdNumber ?fnr ;
prodm-cad:hasCondominiumUnitNumber ?snr ;
prodm-cad:hasCadastralID ?cadId ;
cp:nationalCadastralReference ?nationalCadastralReference ;
dcterms:coverage ?au .

?au a au:AdministrativeUnit;
au:country "NOR" ;
au:nationalCode ?code;
au:nationalLevel ?level .

?rr a prodm-cad:RealRights ;
dbo:type ?role ;
dul:defines ?cp ;
dul:defines ?org ;
dcterms:source "cadaster" ;
prodm-cad:hasNumerator ?teller ;
prodm-cad:hasDenominator ?nevner ;
prodm-cad:hasStartDate ?startDate ;
prodm-cad:hasEndDate ?endDate .
```

```
?org a prodm-cad:RightsHolderOrganization ;
schema:leiCode ?leiCode ;
prodm-com:hasAlternativeName ?altName .

?altName a prodm-com:AlternativeName;
prodm-com:hasAlternativeName ?altNavn ;
dcterms:source "cadaster" .
}
WHERE
{?cp a prodm-cad:CadastralParcel;
prodm-com:isStateOwned true ;
prodm-cad:hasCadastralParcelID ?id ;
prodm-cad:hasCadastralUnitNumber ?gnr ;
prodm-cad:hasPropertyUnitNumber ?bnr ;
prodm-cad:hasLeaseholdNumber ?fnr ;
prodm-cad:hasCondominiumUnitNumber ?snr ;
prodm-cad:hasCadastralID ?cadId ;
cp:nationalCadastralReference ?nationalCadastralReference ;
dcterms:coverage ?au .

?au a au:AdministrativeUnit;
au:country "NOR" ;
au:nationalCode ?code;
au:nationalLevel ?level .

?rr a prodm-cad:RealRights ;
dbo:type ?role ;
dul:defines ?cp ;
dul:defines ?org ;
dcterms:source "cadaster" ;
prodm-cad:hasNumerator ?teller ;
prodm-cad:hasDenominator ?nevner ;
prodm-cad:hasStartDate ?startDate ;
prodm-cad:hasEndDate ?endDate .

?org a prodm-cad:RightsHolderOrganization ;
schema:leiCode ?leiCode ;
prodm-com:hasAlternativeName ?altName .
```

	?altName a prodm-com:AlternativeName; prodm-com:hasAlternativeName ?altNavn ; dcterms:source "cadaster" . }
3	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#> PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#> PREFIX schema: <http://schema.org/> PREFIX gsp: <http://www.opengis.net/ont/geosparql#> PREFIX sf: <http://www.opengis.net/ont/sf#> PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#> PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX dcterms: <http://purl.org/dc/terms/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#> PREFIX dbr: <http://dbpedia.org/resource/> PREFIX au: <http://www.w3.org/2015/03/inspire/au#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> CONSTRUCT {?lp prodm-cad:hasLandParcelID ?lpID ; a prodm-cad:LandParcel ; dul:part ?cp ; prodm-com:hasAreaMeasurement ?lpareameas ; gsp:hasGeometry ?lpGeom . ?lpareameas a prodm-com:AreaMeasurement ; rdf:value ?area ; sdmx-attribute:unitMeasure dbr:Square_metre . ?lpGeom a sf:MultiPolygon ; gsp:asWKT ?coords . ?cp a prodm-cad:CadastralParcel; prodm-cad:hasLandParcel ?lp ; prodm-cad:hasCadastralParcelID ?id ; prodm-cad:hasCadastralUnitNumber ?gnr ;

```
prodm-cad:hasPropertyUnitNumber ?bnr ;
prodm-cad:hasLeaseholdNumber ?fnr ;
prodm-cad:hasCondominiumUnitNumber ?snr ;
prodm-cad:hasCadastralID ?cadId ;
cp:nationalCadastralReference ?nationalCadastralReference ;
dcterms:coverage ?au ;
prodm-com:hasIndicator ?chInd ;
prodm-com:hasIndicator ?gpInd .

?chInd a prodm-com:Indicator ;
dbo:type "CulturalHeritage" ;
rdf:value ?chValue .

?gpInd a prodm-com:Indicator ;
dbo:type "GroundPollution" ;
rdf:value ?gpValue .

?au a au:AdministrativeUnit;
au:country "NOR" ;
au:nationalCode ?code;
au:nationalLevel ?level ;
}
WHERE
{?lp prodm-cad:hasLandParcelID ?lpID ;
a prodm-cad:LandParcel ;
dul:part ?cp ;
prodm-com:hasAreaMeasurement ?lpareameas ;
gsp:hasGeometry ?lpGeom .

?lpareameas a prodm-com:AreaMeasurement ;
rdf:value ?area ;
sdmx-attribute:unitMeasure dbr:Square_metre .

?lpGeom a sf:MultiPolygon ;
gsp:asWKT ?coords .

?cp a prodm-cad:CadastralParcel;
prodm-com:isStateOwned true ;
prodm-cad:hasLandParcel ?lp ;
prodm-cad:hasCadastralParcelID ?id ;
```

	prodm-cad:hasCadastralUnitNumber ?gnr ; prodm-cad:hasPropertyUnitNumber ?bnr ; prodm-cad:hasLeaseholdNumber ?fnr ; prodm-cad:hasCondominiumUnitNumber ?snr ; prodm-cad:hasCadastralID ?cadId ; cp:nationalCadastralReference ?nationalCadastralReference ; dcterms:coverage ?au ; prodm-com:hasIndicator ?chInd ; prodm-com:hasIndicator ?gpInd . ?chInd a prodm-com:Indicator ; dbo:type "CulturalHeritage" ; rdf:value ?chValue . ?gpInd a prodm-com:Indicator ; dbo:type "GroundPollution" ; rdf:value ?gpValue . ?au a au:AdministrativeUnit; au:country "NOR" ; au:nationalCode ?code; au:nationalLevel ?level ; }
4	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#> PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#> PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#> PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX dbr: <http://dbpedia.org/resource/> PREFIX dcterms: <http://purl.org/dc/terms/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX au: <http://www.w3.org/2015/03/inspire/au#> PREFIX gsp: <http://www.opengis.net/ont/geosparql#> PREFIX sf: <http://www.opengis.net/ont/sf#> PREFIX prodm-nh: <http://vocabs.datagraft.net/proDataMarket/0.1/NaturalHazard#> PREFIX geof: <http://www.opengis.net/def/function/geosparql/> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>

	PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
	
	CONSTRUCT {?cp prodm-com:hasAreaMeasurement ?cparea .
	?cparea a prodm-com:AreaMeasurement;
	sdmx-attribute:unitMeasure dbr:Square_metre;
	rdf:value ?arealsum;}
	WHERE
	{
	
	SELECT ?cp ?cparea (sum (?areal) as ?arealsum)
	where
	{
	?lp a prodm-cad:LandParcel;
	dul:part ?cp;
	prodm-com:hasAreaMeasurement ?areameasurement.
	?areameasurement a prodm-com:AreaMeasurement;
	sdmx-attribute:unitMeasure dbr:Square_metre;
	rdf:value ?areal .
	
	BIND(IRI(concat(str(?cp),"/AreaMeasurement")) As ?cparea)
	}
	GROUP BY ?cp ?cparea
	}

5	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#> PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#> PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#> PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX dbr: <http://dbpedia.org/resource/> PREFIX dcterms: <http://purl.org/dc/terms/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX au: <http://www.w3.org/2015/03/inspire/au#> PREFIX gsp: <http://www.opengis.net/ont/geosparql#> PREFIX sf: <http://www.opengis.net/ont/sf#> PREFIX prodm-nh: <http://vocabs.datagraft.net/proDataMarket/0.1/NaturalHazard#> PREFIX geof: <http://www.opengis.net/def/function/geosparql/> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

CONSTRUCT {
?bygg a prodm-cad:Building ;
prodm-cad:hasCadastralBuildingNumber ?byggnr ;
prodm-cad:isBuiltOn ?cp ;
dbo:type ?byggtype ;
dbo:status ?status ;
prodm-com:hasAreaMeasurement ?areameasurement;
prodm-com:hasIndicator ?chInd ;
prodm-com:hasIndicator ?sefrakInd .

?areameasurement a prodm-com:AreaMeasurement;
sdmx-attribute:unitMeasure dbr:Square_metre;
rdf:value ?areal .

?chInd a prodm-com:Indicator ;
dbo:type "CulturalHeritage" ;
rdf:value ?chValue .

?sefrakInd a prodm-com:Indicator ;
dbo:type "SEFRAK" ;
rdf:value ?sefrakValue .

?cp a prodm-cad:CadastralParcel;
prodm-cad:hasBuilding ?bygg .

}

WHERE {
?bygg a prodm-cad:Building ;
prodm-cad:hasCadastralBuildingNumber ?byggnr ;
prodm-cad:isBuiltOn ?cp ;
dbo:type ?byggtype ;
dbo:status ?status ;
prodm-com:hasAreaMeasurement ?areameasurement;
prodm-com:hasIndicator ?chInd ;
prodm-com:hasIndicator ?sefrakInd .
```

	?areameasurement a prodm-com:AreaMeasurement; sdmx-attribute:unitMeasure dbr:Square_metre; rdf:value ?areal . ?chInd a prodm-com:Indicator ; dbo:type "CulturalHeritage" ; rdf:value ?chValue . ?sefrakInd a prodm-com:Indicator ; dbo:type "SEFRAK" ; rdf:value ?sefrakValue . ?cp a prodm-cad:CadastralParcel; prodm-com:isStateOwned true ; prodm-cad:hasBuilding ?bygg . }
6	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#> PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#> PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#> PREFIX schema: <http://schema.org/> PREFIX gsp: <http://www.opengis.net/ont/geosparql#> PREFIX sf: <http://www.opengis.net/ont/sf#> PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#> PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> PREFIX dbo: <http://dbpedia.org/ontology/> PREFIX dcterms: <http://purl.org/dc/terms/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> CONSTRUCT { ?rr dul:defines ?building} WHERE { ?cp a prodm-cad:CadastralParcel; prodm-cad:hasBuilding ?building; cp:nationalCadastralReference ?nationalCadastralReference. ?rr a prodm-cad:RealRights; dul:defines ?cp. ?building a prodm-cad:Building; prodm-cad:isBuiltOn ?cp.

	}

7	PREFIX prodm-cad: <http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#>
	PREFIX prodm-com: <http://vocabs.datagraft.net/proDataMarket/0.1/Common#>
	PREFIX prodm-soe: <http://vocabs.datagraft.net/proDataMarket/0.1/SoE#>
	PREFIX cp: <http://www.w3.org/2015/03/inspire/cp#>
	PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#>
	PREFIX dbo: <http://dbpedia.org/ontology/>
	PREFIX dbr: <http://dbpedia.org/resource/>
	PREFIX dcterms: <http://purl.org/dc/terms/>
	PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
	PREFIX au: <http://www.w3.org/2015/03/inspire/au#>
	PREFIX gsp: <http://www.opengis.net/ont/geosparql#>
	PREFIX sf: <http://www.opengis.net/ont/sf#>
	PREFIX prodm-nh: <http://vocabs.datagraft.net/proDataMarket/0.1/NaturalHazard#>
	PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>
	PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
	
	CONSTRUCT {
	?bygg a prodm-cad:Building ;
	gsp:hasGeometry ?byggGeom .
	
	?byggGeom a sf:Point ;
	gsp:asWKT ?coords ;
	}
	WHERE {
	?bygg a prodm-cad:Building ;
	prodm-cad:isBuiltOn ?cp ;
	gsp:hasGeometry ?byggGeom .
	
	?byggGeom a sf:Point ;
	gsp:asWKT ?coords .
	
	?cp a prodm-cad:CadastralParcel;
	prodm-com:isStateOwned true ;
	prodm-cad:hasBuilding ?bygg .

	}