A Web-Based Object-Relational Cadastral Model for Kenya: The Case for Spatially Enabled Land Governance.

Prof.Gordon Wayumba[1] ,Benard Odhiambo[2], Dr. Samson Ayugi[3],Martin Ondiwa[4], Boaz Nyakongo[5]

[1] Lecturer in Geospatial Engineering and Land Administration, The Technical University of Kenya, Nairobi. Tel: +254722775168, Email: gowayumba@gmail.com.

[2] Geospatial Engineer and Developer at Serian Geospatial Limited. Nairobi. Tel: +254717374902, Email: odhiambb@gmail.com.

[3] Lecturer in Geospatial Engineering and Geoinformatics, The Technical University of Kenya, Nairobi. Tel: +254722276544, Email: okothayugi@gmail.com.

[4] Geospatial Engineer, The Technical University of Kenya, Nairobi. Tel: +254798095529, Email: surekondiwa@gmail.com.

[5] Geospatial Engineer, The Technical University of Kenya, Nairobi. Tel: +254797073761, Email: boaznyakongo99@gmail.com.

1 The Technical University of Kenya, Nairobi

2 Private Consultant in Geospatial Engineering

ABSTRACT

Harmonizing data into comprehensive cadastral models, such as Land Information Management Systems, is a mandatory requirement in developed countries, where such systems support the implementation of sustainable development needs. For example, Europe already has the Land Administration Domain Model [Lemmen, 2012] and Infrastructure for Spatial Information in the European Community (INSPIRE, 2007). Both models were developed to facilitate efficient spatial data access and sharing between the European nations and for ease of doing business. Accessible information on tenure systems is crucial for various needs; it supports sustainable economic and infrastructural development and interrelated spatial planning needs. It also supports resource and environment management for climate change and the associated disaster risk reduction.

Contrastingly, in most developing countries, several aspects of spatial data management are still missing. In most cases, land parcels are not registered, recorded, or spatially referenced to a specific geodetic datum. In the cases where spatial/and attribute data exist, their quality is poor and incomplete, and the land records kept in the registries are not up-to-date and do not represent the actual situation on the ground. Most spatial data are analog format, and the few computerized data are often improperly maintained.

A research project was initiated between the Technical University of Kenya and a Cooperative Society in Nairobi to develop a prototype Web-Based cadastral model for Kenya based on the Object-Relational model to solve this problem. The prototype used a web-centric solution, with data stored and managed centrally from an Object-Relational database (PostgreSQL/PostGIS) through implementing the Django framework as a back-end framework. Further interactive pages

in the front end were developed using bootstrap4, HTML, JavaScript, and CSS. Consequently, it enabled users to view the land data in the system through their web browsers. The research, therefore, developed a Web-Based Application for Land Information Management where different users can log in and interact with the different land information attributes and spatial data for different needs.

Keywords.

Web_based Cadastre, Object-Relational database, Land Information Management, spatial data.

Wayumba Gordon, Odipiambo Bernard, Martin Sure Ondiwa, Nyakongo Boas and Samson Alungi (Kenya)

1. INTRODUCTION

The National Land Policy (NLP) [MoL, 2009] observed that the Kenya government currently lacks an up-to-date and comprehensive National Land Information System (NLIMS), and this compromises the efficient land administration processes in the country. The available land information system is currently held primarily in paper format and is managed manually. This is inefficient, time-consuming and does not support timely decision-making on land matters. Additionally, the current system is centralized in Nairobi; it is slow and involves cumbersome and bureaucratic procedures. De Soto [2000] observed that the bureaucratic Land Administration processes and lack of adequate documentation of resources in developing countries is why capitalism thrives best in the developed world but fails everywhere else.

The Ministry of Lands in Kenya, in its Strategic Plan (2008-2012) and performance contract (2010/2011), already identified that to ensure effective and efficient service delivery to the clients, processes, procedures and practices of handling land information need to be redesigned. It further noted that such redesign should include Business Process Re-Engineering and targeted four bench-marking activities towards achieving these objectives. These four bench-mark activities included reviewing and documenting current processes, procedures and practices; redesigning procedures and processes of land administration; making recommendations for quick-win projects; and implementing a digital National Land Information System [MoL, 2011: 17].

After promulgating the Kenya Constitution 2010 and establishing the National Land Commission (NLC), the government mandated the newly formed NLC to develop NIMS for the country. However, up-to-date, the NLC has not developed the NLIMS and Kenya still operates the manual LIMS. Kuria et al. [2016] attempted to develop a web-based workflow system for better Land Administration, but no web-based operational NLIMS exists. To solve this problem, the School of Surveying and Spatial Science of the Technical University of Kenya initiated a research project to develop and test a prototype Web-Based Object-Relational Cadastral Model for Kenya.

2. METHODOLOGY

2.1 System Analysis, Data Collection, and User Requirements.

The methodology adopted for the research is summarized in Figure 1.1 and consisted of the following approaches: literature review and research, systems analysis, data collection and user need assessment, systems design, coding and implementation, testing, deployment and maintenance. To elaborate on the methodology in more detail, it consisted first of (i) transformation of the coordinates of the cadastral map of the study area from Cassini-Soldner projection to the UTM (1960 Arc Datum) through a four-parameter affine transformation and further transformation from the UTM projection (1960 Arc Datum) to the WGS84 projection through the PROJ4 Software (Codes 21037 and 4326 for the UTM 1960 Arc Datum and WGS84 respectively). The transformed coordinates were used to digitize the cadastral maps into the WGS84 coordinates system for compatibility with the World Wide Web

(WWW). Finally, appropriate classes representing the main components of the Kenyan Cadastre were developed with the UML Diagrams (Fig.1.4). The various selected classes are presented below for more details and a better understanding of the Kenyan Cadastre.

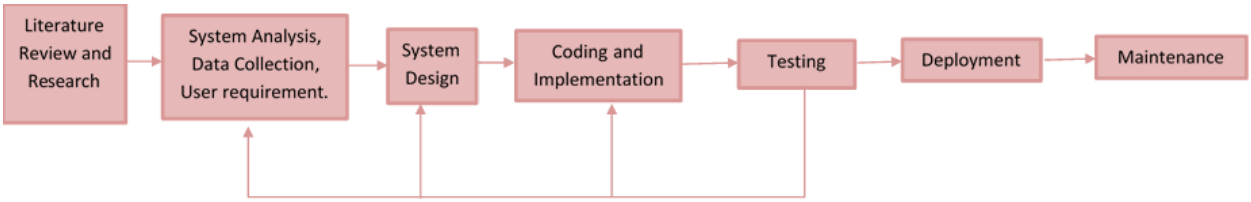


Fig 1. Flow-chart Diagram for the Research Methodology

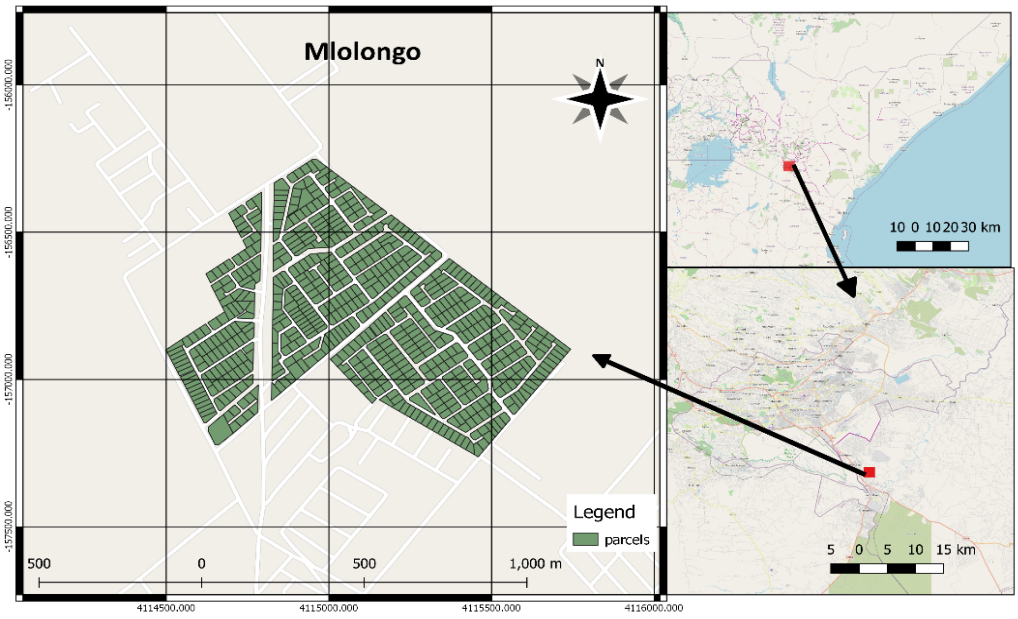


Fig. 2. Map showing the location of the study area.

2.2 The UML Classes used in this study were adopted from Nyadimo [2006] and Wayumba [2013]

2.2.1 Parcel Class

The scenario in the diagram below depicts the relationship between the Land Object, RightfulClaimant, and RightsOrRestrictions as required in the Cadastral Core Model. The LandObject can be modeled as a polygon feature (in most cases with coordinates) and presented in a class diagram, the ParcelClass, including its attributes and the operations that can be performed over the LandClass. The RightfulClaimant, in this case, is identified as the land owner, modeled in the OwnerClass.

This way, it is possible for the government to identify property/land parcel owners through the registration documents. Currently, there is no connection between the registrar of the person's details and the land information kept by the government in the Ministry of Lands. The main attributes include the registered person's name, postal address, RegistrationID, OwnerID, passport ID, the owner's location by coordinates, and the date of issue of the identity card.

Other classes, such as taxation class, physical planning class, adjudication class etc, discussed in (Wayumba, 2013) thesis, were also considered in developing this system.

2.3 System Design.

After identifying the appropriate classes constituting the cadastral system in Kenya, appropriate UML Diagrams were designed to represent each class using the Creatly.com software from the Web, Microsoft Visio, or Django extensions. Figure 1.3 shows the UML diagrams and the association between the various classes. In the UML diagrams, the ParcelClass is related directly to the survey class but relates to the Owners class through the Registration class. The parcel class is further related to the Survey Beacons class through the Survey class. The relationship between the building class and the registration class is indicated in the UML diagrams. The EncumbrancesClass is related directly to the Registration class as the encumbrances are registered against the properties at the land registries.

The EncumbranceType identifies the type of encumbrance while the operations include registration of encumbrance in the Lands office, demarcation on the ground, and removal of encumbrance when the need expires. The relationship between the EncumbranceClass is given as many-to-many as one land parcel can have many encumbrances registered upon it at any one time. The BoundaryBeaconsClass is related to the RegistrationClass through the SurveyClass, and the Parcel Class of the SurveyClass contains the Folio Registry Plan Numbers (FR No)., which contain the property beacons. This way, it was found possible to access the coordinates defining the parcel boundary through the Survey Plans, folio registry maps (FRs)

The RegistrationClass is most central in the cadastral modeling as all aspects of the Cadastre relate to registration. The relationship between the RegistrationClass and the other classes is indicated in the UML diagrams. The ValuationClass is related to the ParcelClass through the RegistrationClass. In the UML model, the SerialNo gives the serial number of the valuation file, while ValuationBookNo refers to the book used for valuation. The attribute Value gives the unimproved site value of the land parcel. The ParcelID serves to link the ValuationClass to the ParcelClass. The relationship between the ValuationClass and the ParcelClass is one-to-one (1:1), meaning that one land parcel can only have one value. The SurveyClass is related to the RegistrationClass through the ParcelsClass. The relationship is many-to-many as one survey plan can carry many parcels, and one parcel cannot support many survey plans. The WaterBillClass with attributes consisting of Meter Number, Name of Owner, Address of Owner, amount of bill, date of last payment, Building ID, and location of meter by coordinates.

The WaterBillClass was modeled as an aggregate of the BuildingClass as one building may have many Water Bills and is connected to the RegistrationClass through the Building Class. The association is many-to-many, meaning one parcel can have many buildings with different water bills. The ElectricityBillClass is therefore modeled as an aggregate of the BuildingClass as a Building may have many units and hence many power bills to pay. The classclass provides all the necessary attributes for location and efficient readings of the meters. The main attributes include the name of the bill's owner, the owner's address, the bill amount, the meter number, the date of the last payment, and the location by coordinates. The association is many-to-many, and Figure 1.1 shows that the ElectricityClass is related to the RegistrationClass through the BuildingClass.

Therefore, It was necessary to include PersonsClass in the cadastral database to provide the registered persons' location and profile. This way, it would be possible for the government to have adequate data for locating the clients and be able to relate the identity cards with other social attributes. In the UML Diagrams in Fig. 3. The PersonsClass is connected to the parcel RegistrationClass through the OwnershipClass. The PhysicalPlanningClass is directly related to the LandAdmnistrationClass, the SurveyClass, where the Parcel Number is allocated to the planned and surveyed land parcel. The classclass is then further related to the RegistrationClass through the parcel class. Land Admin.

The Land Administration Class is first related to SurveyClass, the ParcelsClass and then to the RegistrationClass, where land titles are prepared. During the user needs assessment, it was observed that the demand for information in this classclass was the highest. Including the LandAdjudicationClass in the UML diagram gives the government an overview of the attributes and how the system can be modernized. The classclass is directly related to SurveyClass, then to the ParcelsClass and RegistrationClass. The ParkingClass is connected directly to the ParcelsClass and indirectly to the RegistrationClass, as the parking spaces are usually created out of the parcels to be registered and are allocated the Land Registry Number as any other parcels.

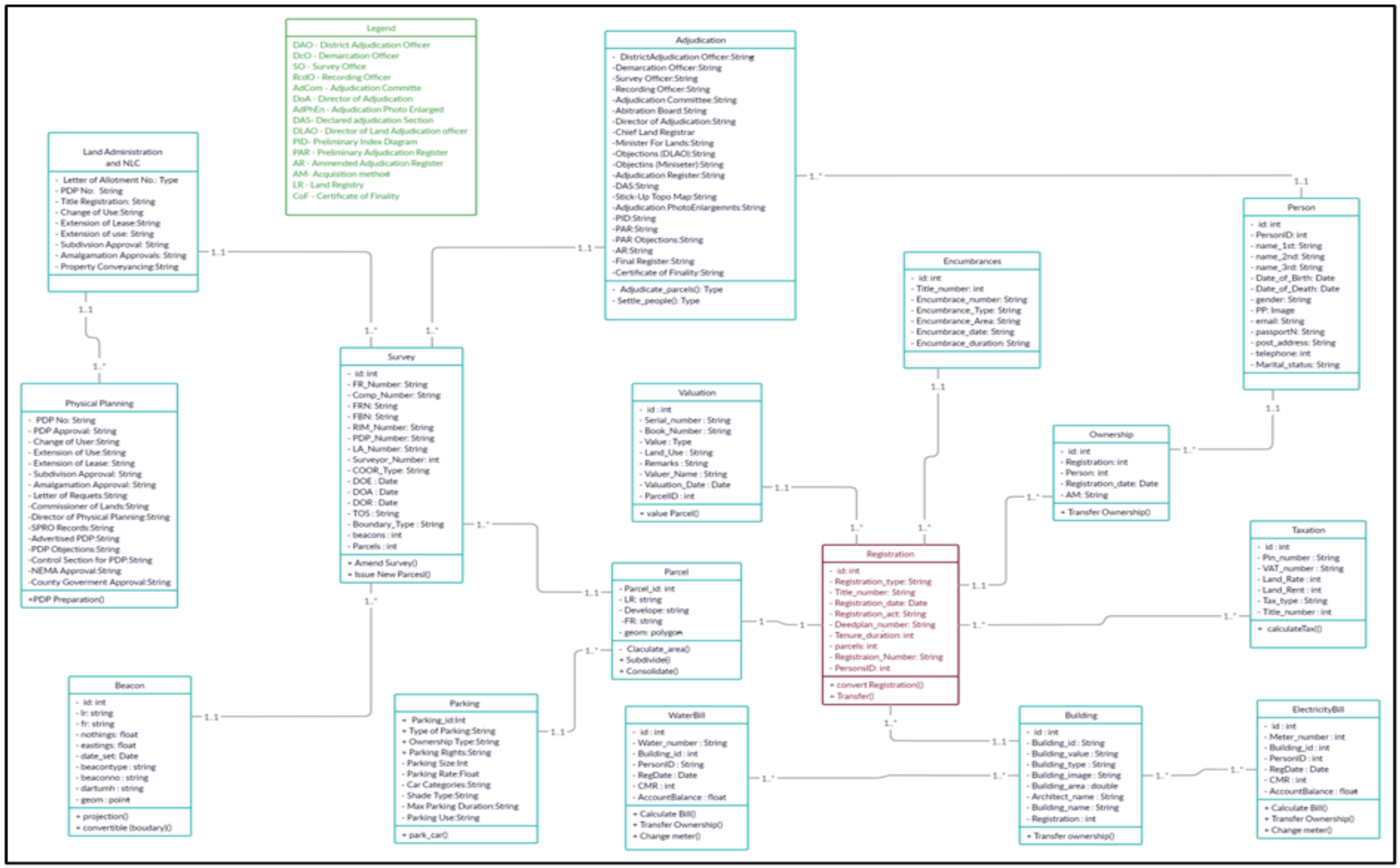


Fig. 3. UML Diagrams for the cadastral classes in Kenya

2.4 Use case diagrams

For the system development, only four users were considered: the visitor, the land owner, the land administration staff/officer, the land professionals, and the land administrators. Fig. 4 shows one of the use case diagrams for landowners. The use case diagrams assist the systems design team identify users and their needs. In this research, the users of the system and their needs were carried out with structured questionnaires implemented by Wayumba (2013). The primary users identified consisted of the visitors to the Land Registry who are interested in various aspects of the Cadastre, land owners who usually want to assess the status of their property, the land administrators involved in the operations of the Cadastre in the land registry, and the land professionals. The use case diagrams were coded in **views.py,** while the user privileges were coded in the decorators, as discussed later in this paper. Flow chart Diagrams were prepared for a few selected use case diagrams to demonstrate the directions of the database operation.

After designing the appropriate UML diagrams and the Use Case Diagrams, the authors designed an empty PostgreSQL (PostGIS) database as it is an Open-Source system capable of handling spatial data better than other available database systems. The Django (Geodjango) was selected as the best back-end framework for data interaction locally and in the Web interface, while the bootstrap4 was selected for the front-end operations.

Figure 4: Use case diagram for Land admin super-admin. Figure 5: Use Case diagram for Land administration staff

Fig 6: Use Case diagram for land owners. Fig 7: Use case diagram for visitors.

2.5 Coding and Implementation.

In order to convert the UML classes into computer models, a Geodjango application was created to handle the coding processes and import the spatial data into the already-created PostgreSQL/PostGIS database. The spatial data was prepared from survey plans through coordinate transformation and digitization. Consequently, the resulting shape files are populated in QGIS software as fields identified as id, FR, LR, and Development status. After completion of the process, the final data was saved as Esri shapefiles (.shp). To code spatial class data in Django, the GDAL libraries (ogrinfo or inspect) were used to inspect and confirm that the coded data corresponded with the class attributes in the UML diagrams. The Ogrinfo in the GDAL library is generally used to inspect spatial data formats like shapefiles (.shp) or .kml before they can be coded into the Django model.

Sometimes, the same results can be achieved using an ogrinspect modified for Django. This or inspect library inspects the attributes of the spatial data versus the original spatial UML class. It automatically generates the model definition (i.e., the model definitions Fig. 5) and Layer Mapping dictionary for the spatial classes automatically. These auto-generated models' definitions can be copied into the Django class models (i.e., Django models.py). This study used the latter version to verify the generated spatial classes before uploading the model classes into the database. Layer Mapping dictionary was used later to load the two spatial data classes into their respective tables in the database (Westra, 2016).

Finally, from the GeoDjango App, the non-spatial attribute classes in the UML diagrams were scripted (coded) as models and, together with the spatial classes models (from or inspect auto-generated model definitions classes), were all combined to form the class models in Django models.py. This model.py is the representation of the UML in Django language.

```
class beacons(models.Model):
    id = models.IntegerField(primary_key=True)
    lr = models.CharField(max_length=15)
    developed = models.CharField(max_length=2,null=True)
    fr = models.ForeignKey('Parcels',
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='parcels')
    nothings = models.FloatField()
    eastings = models.FloatField()
    date_set = models.DateField(null=True)
    beacontype = models.CharField(max_length=10, null=True)
    beaconno = models.BigIntegerField(null=True)
    dartumh = models.CharField(max_length=21, null=True)
    geom = models.MultiPointField(srid=4326)

    def __unicode__(self):
        return self.beacontype
```

Fig. 5. An example of a beacon spatial Django Class Model

```
class parcels(models.Model):
    id = models.IntegerField(primary_key=True)
    lr = models.CharField(max_length=15)
    developed = models.CharField(max_length=2, null=True)
    fr = models.CharField(max_length=15, null=True)
    geom = models.MultiPolygonField(srid=32737)

    def __unicode__(self):
        return self.lr
```

Fig An example of a Parcel spatial Django Class Model

Fig. An example of a PersonClass a Non-Spatial Attributes Django Model Class

```
class Ownership(models.Model):
    Method= [
        ('A','Allotement'),
        ('I','Inheritance'),
        ('P','Purchase'),
        ('G','Gift'),
    ]
    Registration = models.ForeignKey('Registration', on_delete=models.CASCADE,
null=True )
    Person = models.ForeignKey('Person', on_delete=models.CASCADE, null=True )
    Registration_date = models.DateField( max_length=50, null=True, blank=True )
    AM = models.CharField(verbose_name='Acquisition Method',max_length=1,
null=True, choices=Method, blank=True)
    def __unicode__(self):
        return self.AM
```

Fig. An example of a PersonClss Non-Spatial Attributes Django Model Class

```
68  class Registration(models.Model):
69      Registration_type = models.CharField( max_length=50, null=True )
70      Title_number = models.IntegerField( null=True, blank=True )
71      Title_deed = models.ImageField(verbose_name='Upload title deed image', null=True, blank=True)
72      Deedplan = models.ImageField(verbose_name='Upload Deedplan image', null=True, blank=True)
73      Registration_date = models.DateField( null=True, blank=True )
74      Registration_act = models.CharField( max_length=50, null=True )
75      Tenure_type = models.CharField( max_length=50, null=True )
76      Deedplan_number = models.IntegerField( null=True, blank=True )
77      Tenure_duration = models.IntegerField( null=True )
78      parcels = models.OneToOneField('Parcels',on_delete=models.CASCADE,null=True , related_name='parco')
79      Registraion_Number = models.IntegerField( null=True, blank=True )
80      PersonsID = models.ManyToManyField('Person',
81          through='Ownership',
82          through_fields=('Registration','Person'),
83          related_name='Persona')      You, seconds ago • Uncommitted changes
84
85      def __str__(self):
86          return '%d: %s %s' %( self.Deedplan_number,self.Registration_act, self.Tenure_type)
```

Fig An example of an OwnershipClass Non-Spatial Attributes Django Model Class

Wayumba Gordon, Odhiambo Bernard, Martin Suter Ondiwa, Nyakongo Boaz and Samson Awuor (Kenya)

Fig. 7. Diagram of Object Relational Mapping of Django generated classes into the PostgreSQL database.

2.5.2 Display of Cadastral Data from the Database.

The data stored in the PostgreSQL database is retrieved through Django views to retrieve cadastral data from the database and display them on a webpage as maps.API. The Django Views.API enables one to retrieve or query data from the database and keep them ready for any other use in the format chosen by the researcher, under views.py or views.API functions are written to retrieve the data required for the function (e.g., object, filtered object, or all objects) and convert them into a useable ready format, i.e., GeoJayson for cadastral data since these data will be loaded on a web page as maps.

To load cadastral data onto the web page, write a function in Django Views. API is the app for retrieving all data from the PostgreSQL database and presenting them as Geojson. The Geojson cadastral data are kept in virtual storage, ready to be used with any map-loading Library. For example, Leaflet.Js can load Geojson cadastral data onto the web page. Leaflet.Js provides ways to view and interact with maps on a web page for any spatial data to be displayed and /or interacted with in the form of web maps. JavaScript library, e.g.leaflet.js, is used. In this case, the OpenStreetMap (OSM) data is utilized as a base map to show the location of the cadastral information. The Leaflet.Js library can generate attribute information in a parcel as **pop-up views.** Fig. 14. shows the pop-up information in a specific parcel in the study site. This tool can, therefore, be used to display many-to-many relations or any special tenure information in the parcel, such as customary/indigenous/or special rights in a parcel. Currently, there has not been any study showing how other special tenure types can be incorporated into the formal land administration system. This facility provides a means of mapping and displaying these special tenure rights as pop-ups.

After the display of spatial data on the webpage, the users may perform queries. **Django views.API** is used to perform queries from the database. For example, Leaflet.Json is a front-end operator that helps to display spatial data onto the database. Other front-end languages include HTML, CSS, and JavaScript. The Django **views.API** is responsible for retrieving both spatial and non-spatial data from the database and keeping them in a virtual format.

For images, **Django Views.API** does not keep any image in the database but keeps only the location information of the image columns files in the static folders in the database. The folder can then be secured in a Dropbox or Google Drive. **Django Views API** picks the image url and stores it in the database. When there is a need to retrieve the image from the database, Django Views.API calls the image url from the database by following the URL path and retrieving the image.

2.5.3 Serializing the Database Data.

Serialization converts the database data so that it is readable on the Web. The Web only recognizes GeoJson, Json or Text languages. GeoJson is the standard language for presenting geospatial data on the Web, while Json deals with other formatted text data, and Text deals with standard Text data. Serializers Classes are then coded in the serializers.py file and imported into the views.py file for queried datasets.

The Serialize classes help to serialize the queries. It helps add related model attributes to the main model class being serialized. For example, a father may have a child from the typical family circle. Only the names of the family members in the immediate father class will be shown without serializing. However, with the serialization of the class data, the names of other children kept in other files will pop up and be added to the father's family. Sometimes, it is necessary to design nested serializers for special query attributes such as the family issue discussed. Serialization can also help to filter out people who are not supposed to access certain information in the database; therefore, it operates as a security check.

Serializing converts data to be displayed on the Web and Retrieved from the Web. When Django **Views. API** is converting the class for Geojson, using the Geojson Serializers. The Django serializers can have limitations, including following a relationship between the classes. For example, connecting the attributes of one class with another, e.g., bringing taxation class attributes to the spatial class attribute and displaying them on the map. This is for geometry-based classes such as cadastral information; otherwise, it is for the Django Views' non-spatial attribute**s. API** uses the Jason serializers.

The default serialization of Django can only deal with the attributes from the immediate class. If this situation occurs such that the Django serialization cannot recognize the class relationships, the researcher has to build his serializer class using django-rest-framework(drf) for **nested serialization**, e.g., **RegistrySerializer** class, and for spatial class serialization, we use Django-rest-framework-gis to use GeoFeatureModel serializers specifying field containing geometry e.g. a **ParcelSerializer** shown in Fig. 8.

```
class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = person
        fields = ['PersonID','name_1st','name_2nd','telephone']


class RegistrySerializer(serializers.ModelSerializer):
    persons = PersonSerializer( source='PersonsID', many=True, read_only=True, allow_null=True)
    class Meta:
        model = Registration
        fields = ['id','Tenure_type','persons']


class ParcelSerializer(GeoFeatureModelSerializer):
    Registrations = RegistrySerializer( source='parco',allow_null=True)
    class Meta:
        model = parcels
        geo_field = 'geom'
        fields = ['id','lr','geom','Registrations']
```

Fig. 8. A special Serializer.py for a land parcel

```
def owners(request):
    queryset = parcels.objects.all()
    queryset1 = queryset.exclude(parce__isnull=True)
    parce1 = ParcelSerializer(data=queryset1, many=True)
    parce1.is_valid()
    parce1.data
    return JsonResponse(parce1.data, content_type='json', safe=False)
```

Fig. 9. Using the Serializer in Views.py

```
urlpatterns = [
    path('owner/', views.owners, name='owner')
]
```

Fig. 10. The Url.py

2.5.4 Decorators.

The decorators are used to obstruct access to different views and web pages to display data from the database for non-authorized persons. Generally, access to the database data and specific web pages used to display then requires authorization, and the user privileges are coded in the decorators and used to decorate view functions. For example, from the Use Case Diagrams, the people authorized to access the database and what they should access are known. The users who want to access information on parcel ownership have to be authorized as either owners of the parcels or land-administrator. Those who are not authorized are redirected to where to log in and shown empty ownership information through decorator functions.

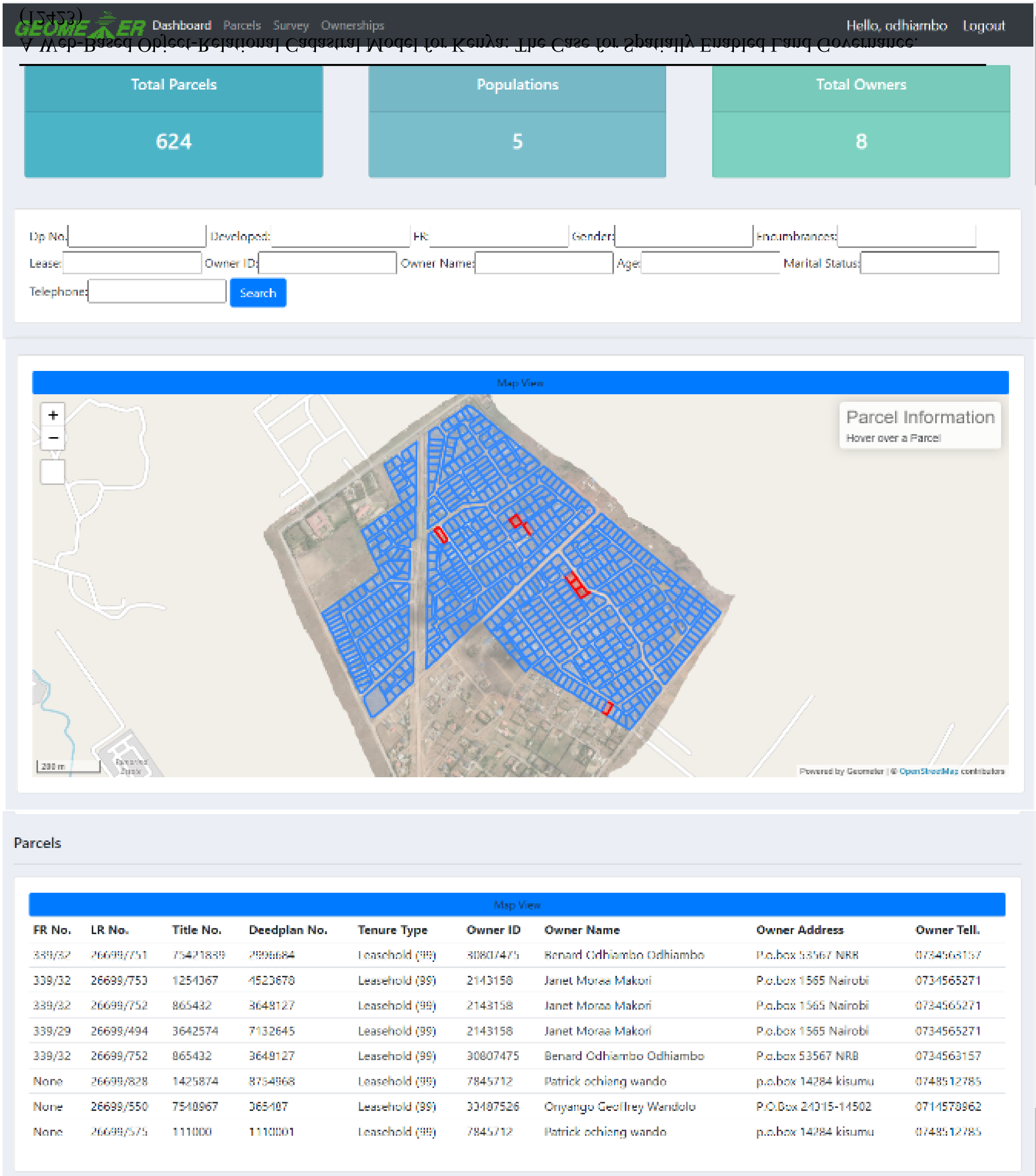
```
@login_required(login_url='login')
@allowed_users(allowed_roles=['Admin'])
def survey(request):
    survey1 = Survey.objects.all()
    context = {'survey':survey1}
    return render(request,'Cadastre/survey.html', context)
```

Fig. 11. Using the Decorators

Fig. 17: Leaflets use the geom in the code to draw the map and the attribute data to display as a pop-up, for example, the information displayed in the pop-up below. The results of these operations are displayed in HTML for visualization. A pop-up can present several property owners in a parcel, displaying many-to-many relationships. It can also display multi-tenure systems or any other multi-user in the Land Administration Systems.

3.1 Dashboard View

Parcels, owners, ownership and survey data were all queried from the database through views in Django. Parcel data were serialized using Django inbuilt geojson serializer since all the properties we used to label the parcels, i.e., LR, were already among the parcels attribute. Retrieved and serialized data were then passed to a template. In the template, the data layout was then designed. A table was used to display the parcels and their respective attributes, and a leaflet was used to display the parcel data on a map.



FR No.	LR No.	Title No.	Deedplan No.	Tenure Type	Owner ID	Owner Name	Owner Address	Owner Tell.
339/32	26699/751	75421839	2956684	Leasehold (99)	30807475	Benard Odhiambo Odhiambo	P.o.box 53567 NRB	0734563157
339/32	26699/753	1254367	4523678	Leasehold (99)	2143158	Janet Moraa Makori	P.o.box 1565 Nairobi	0734565271
339/32	26699/752	865432	3648127	Leasehold (99)	2143158	Janet Moraa Makori	P.o.box 1565 Nairobi	0734565271
339/29	26699/494	3642574	7132645	Leasehold (99)	2143158	Janet Moraa Makori	P.o.box 1565 Nairobi	0734565271
339/32	26699/752	865432	3648127	Leasehold (99)	30807475	Benard Odhiambo Odhiambo	P.o.box 53567 NRB	0734563157
None	26699/828	1425874	8754868	Leasehold (99)	7845712	Patrick ochieng wando	p.o.box 14284 kisumu	0748512785
None	26699/550	7548967	365407	Leasehold (99)	33407526	Onyango Geoffrey Wandolo	P.O.Box 24315-14502	0714578962
None	26699/575	111000	1110001	Leasehold (99)	7845712	Patrick ochieng wando	p.o.box 14284 kisumu	0748512785

Fig. 13. Presents various Dashboard views containing:

3.2 Ownership Map View

Fig. 14. shows the retrieval of ownership from the parcels class in the database as ManyToMany ownerships in a pop-up fashion.

The Pop-Up shows the Parcel L.R.No. Under reference, the names of the multiple owners, the tenure of the property, and their Identification card numbers. The method involved in querying the ownership view involves the following steps: (i)A nested-serializer was used to pass the parcel ownership details to the parcel class so that the parcels could be viewed on the digital map of the study site. The map was already loaded onto the PostGIS database through the leaflet operation. (ii) the parcel data was queried from the database and serialized using the developed serializer to display the multiple ownership details. (iii) the data was then served as geojson to the template, (iv) Leaflet was then used to display the geojson data and pop-up ownership details on the click of an open street map (OSM) and satellite image.

3.3 Survey view

Survey data was queried from the database and passed to the survey template for viewing and interaction. In the template, the data were displayed in a table format showing different attributes of the survey. A button was provided for viewing the survey map as an image in a different template, as shown in Fig. 16.

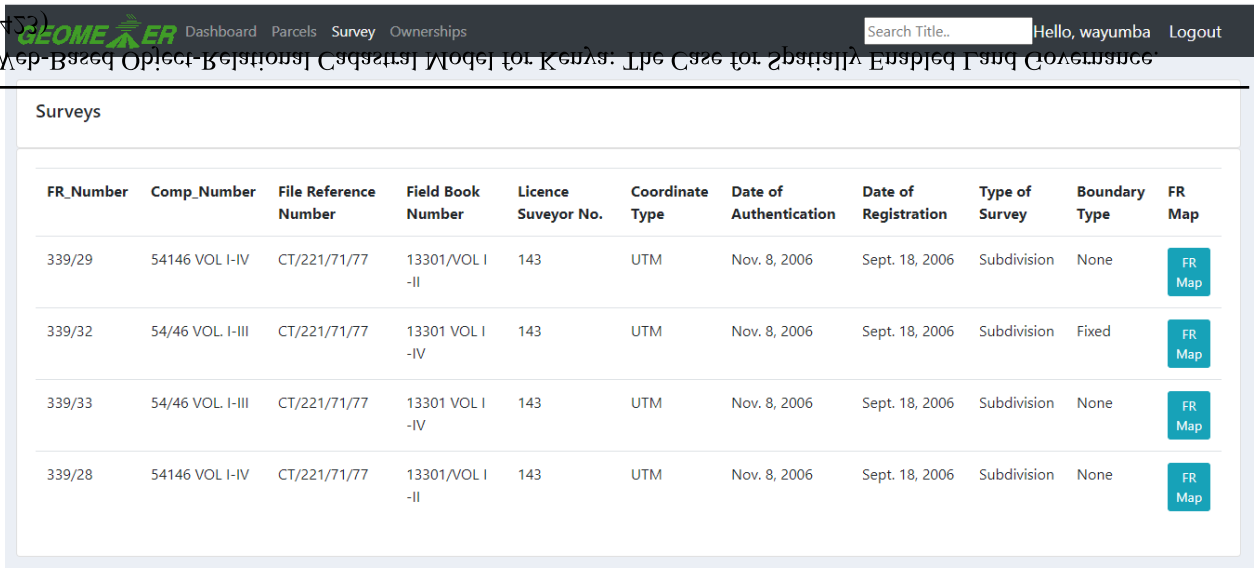


Fig. 15. shows a survey details views in the admin account.

3.4 Owner views

Profile view

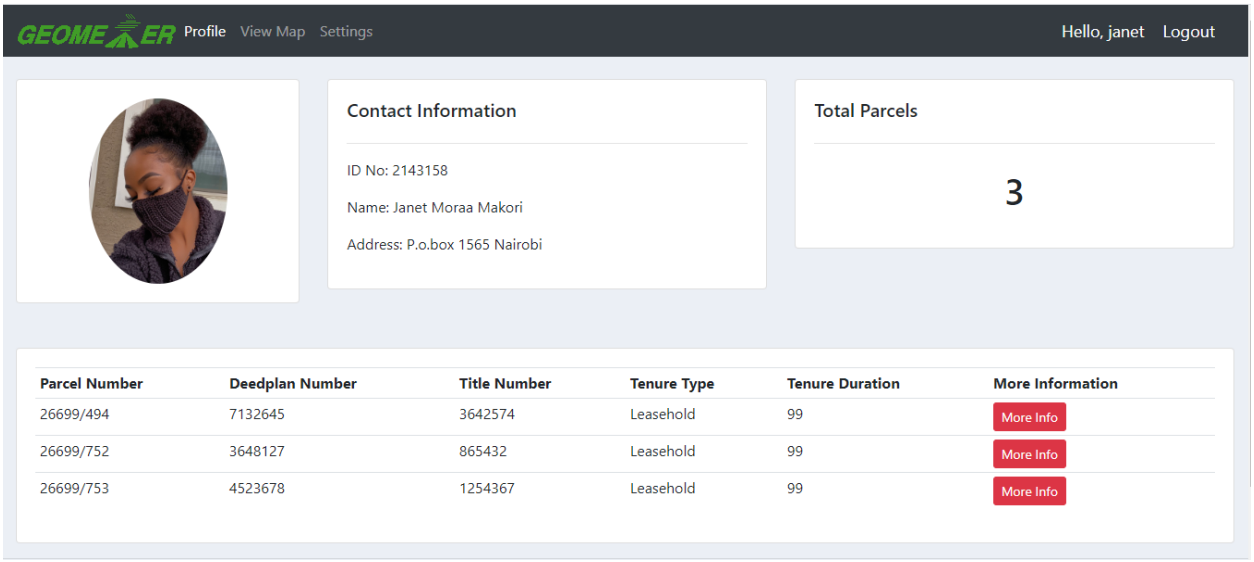


Fig. 17. Land owner profile page with the plots owned and their attributes.

Ownership data was retrieved from the database using a query based on the logged-in user's primary key to get only data specific to the user. Owner data were also retrieved from the database. Counting is done on owned properties. Data was passed to the profile template, whose layout was designed as shown in Fig. 17. above with the help of bootstrap4, HTML, CSS and JavaScript. A button is provided for each owned property to display more details of the specific property on a different template, as shown in Fig. 18.

Fig. 18. Land parcel registration details views.

Owned parcel map view.

Fig. 19. Map showing owned parcels highlighted in red overlaid on a satellite image.

All parcel objects were queried and serialized as geojson and passed to the owner-map template using a leaflet. The geojson were displayed on a map, and owned parcels were highlighted based on their LR number from the owned properties list.

4. CONCLUSIONS

Kenyan LADM country profile has been extended and used to develop a Web-based Object-Relational Cadastral model prototype that supports access to cadastral information through the Web anywhere in the world. This system can be adapted to other developing countries facing similar constraints to Kenya and, through this, enhances the transparency of land information to the world, increasing security. This system can be extended to a 3D cadastral system to complete it. The authors believe that the Ministry of Lands in Kenya can adopt and implement this prototype database at the government's National and County levels.

5. ACKNOWLEDGMENT

The authors acknowledge the Technical University of Kenya for availing some of the laboratory facilities to prepare the paper.

6. FUNDING

The paper was prepared purely from the funds contributed by the authors themselves.

7. AUTHOR CONTRIBUTIONS:

Benard O. Odhiambo investigator, field task, web-application development (60% of the paper)

Gordon O. Wayumba, reviewer, consultant, and document (40% of the paper)

8. REFERENCES

ISO 19152 (2012). Geographic information - Land Administration Domain Model (LADM), version 1 December 2012.

Lemmen, C. (2012). A Domain Model for Land Administration. PhD Thesis. DelftUniversity of Technology, Delft, the Netherlands.

De Soto, H., (2000). The Mystery of Capital. Why Capitalism Triumphs in the West and Fails Everywhere Else, Basic Books, New York, USA.

Greenfield, D. R., & Audrey Roy G(2021). Two Scoops of Django 3. x: Best Practices of Django Web Framework. Two scoops press, 2021.

Kuria, D.N., Moses, M.N., Caroline, G., Charles N. M., (2016). A Web-Based Pilot Implementation of the Africanized Land Administration Domain Model for Kenya. A Case Study of Nyeri County. Journal of Geographic Information Systems, 2016, pp. 8, 171–183. http://www.scirp.org/journal/jgis http://dx.doi.org/10.4236/jgis.2016.82016 (Accessed on 10th June 2021).

Ministry of Lands, 2009: Sessional Paper on National Land Policy, August, 2009. Government Printer, Nairobi, Kenya.

Ministry of Lands, 2011: A Report on Re-Engineering of the Ministry's Business Processes. Ministry of Lands, Nairobi.

Nyadimo, E., (2006). Cadastral Data and Process Modelling Using the Unified Modelling Language A

Case Study of Kenya. MSc Thesis University of Munich, Germany.

Wayumba, G. O. (2013). An evaluation of the cadastral system in Kenya and a strategy for its modernization. PhD Thesis, 08 2013. UoN Digital Repository, University of Nairobi, http://erepository.uonbi.ac.ke/handle/11295/56366?show=full.

Westra, E. (2016). Python Geospatial Development. Third Edition ed., Birmingham, packt publishing, 2016.

Amondi, Grace (2019). "Deploy GeoDjango application to Heroku." Medium, 2019. https://medium.com/the-geospatials/deploy-geodjango-application-to-heroku-in-2019-part-2-f553812d7b4c/. Accessed 5th october 2021.

Django Documentation 3.2 (2021). Geodjango Retrieved from GeoDjango. 23rd September 2021

9. ADDITIONAL READING

In this section, please provide a list of 3-6 additional readings (e.g. journal articles, book chapters, case studies, etc.). As the contributing author(s), you are the best source for suggestions on additional readings in your field.

10. KEY TERMS AND DEFINITIONS

Equity: A system of jurisdictions ensuring the quality of fairness within a society or a territory.
Web-Based: System which can be assessed through a web browser.
Object Relational: Relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and the query language.
Land parcel: The land parcel is identified as the building block of each land administration system.