Projecte final per a la 15ª edició del Màster en Tecnologies de la Informació Geogràfica

Desenvolupament d'un Sistema d'Informació Geogràfica i d'una aplicació mòbil de consulta i recollida de dades

Autor

Víctor Pinell Milian

Tutor per part del MTIG

José Quirós Jiménez

Entitat col·laboradora

Museu de Ciències Naturals de Barcelona

Tutor per part de l'entitat col·laboradora

Francesc Uribe

UAB Universitat Autònoma de Barcelona
Departament de Geografia

15 mtig 2013

nat museu de ciències naturals de Barcelona

RESUM

El present projecte representa un conveni de col·laboració entre la Universitat Autònoma de Barcelona (UAB) i el Museu de Ciències Naturals de Barcelona(MCNB) per a realitzar el Projecte final del Màster en Tecnologies de la Informació Geogràfica, 15a edició, a través de pràctiques professionals.

L'objectiu per a la realització d'aquest projecte és crear un Sistema d'Informació Geogràfica (SIG) amb PostgreSQL per tal d'emmagatzemar les dades recollides a una finca, anomenada Can Catà, situada al Parc Natural de la Serra Collserola, ja que allà el museu desenvolupa un programa de recerca en ecologia evolutiva sobre aus de fa més de 15 anys. Per a la elaboració del SIG s'ha fet un disseny conceptual i lògic de la base de dades a crear. A continuació s'ha realitzat la implementació de les taules a PostgreSQL. I finalment , aquestes s'han omplert amb les dades desitjades mitjançant sentències SQL.

A més del SIG, s'ha de desenvolupar una aplicació per a un dispositiu mòbil per tal de poder consultar i recollir dades de camp. Aquesta aplicació constarà de tres parts ben diferenciades: la primera serà la que permetrà consultar dades en el mateix dispositiu mòbil; la segona permetrà recollir dades de camp; i la tercera serà una part cartogràfica on es trobarà un mapa amb la localització de les caixes niu que formen par de la recerca.

I per últim, s'haurà de dissenyar, mitjançant llenguatge Python, una petita aplicació que permetrà portar a terme una transmissió bidireccional de dades entre el SIG i l'aplicació mòbil.

RESUMEN

El presente proyecto representa un convenio de colaboración entre la Universidad Autónoma de Barcelona (UAB) i el Museo de Ciencias Naturales de Barcelona (MCNB) para realizar el Proyecto final del Máster en Tecnologías de la Información Geográfica, 15a edición, a través de prácticas profesionales.

El objetivo para la realización de este proyecto es crear un Sistema de Información Geográfica (SIG) con PostgreSQL para poder almacenar los datos recogidos en un finca, llamada Can Catá, situada al Parque Natural de la Sierra de Collserola, ya que allí el museo desarrolla un programa de investigación en ecología evolutiva sobre aves des de hace más de 15 años. Para la elaboración del SIG se ha hecho un diseño conceptual y lógico de la base de datos a crear. A continuación se ha realizado la implementación de las tablas a PostgreSQL. I finalmente, estas se han llenado con los datos deseados mediante sentencias SQL.

Además del SIG, se ha desarrollado una aplicación para un dispositivo móvil para poder consultar y recoger datos de campo. Esta aplicación constará de tres partes bien diferenciadas: la primera será la que permitirá consultar datos en el mismo dispositivo móvil; la segunda permitirá recoger datos de campo; y la tercera será la parte cartográfica dónde se hallará un mapa con la localización de las cajas nido que forman parte de la investigación.

I por último, se diseñará, mediante lenguaje Python, una pequeña aplicación que permitirá llevar a cabo una transmisión bidireccional de datos entre el SIG i la aplicación móvil.

ÍNDEX DE TAULES

ÍNDEX DE FIGURES

1. INTRODUCCIÓ

1.1. Presentació.

El següent informe presenta les tasques realitzades en el marc del projecte final del Máster en Tecnologies de la Informació Geogràfica (15a edició) organitzat pel Departament de Geografia de la Universitat Autònoma de Barcelona, en col·laboració amb el Museu de Ciències Naturals de Barcelona.

El projecte consta de tres fases ben diferenciades, la primera és la creació d'un Sistema de Informació Geogràfica per a l'estació biològica de Can Catà, la segona és la realització d'una aplicació mòbil de consulta i recollida de dades de camp, i la tercera és una aplicació de transmissió de dades entre el SIG i la aplicació mòbil. Al llarg d'aquest document, es detalla pas a pas con s'ha portat a terme la realització d'aquestes tres fases.

Es tracta d'un document tècnic que té dos objectius. Per una banda servir de resum de la feina realitzada dins del projecte esmentat. Per l'altra banda, pretén ser una guia metodològica que permeti al personal del Museu de Ciències Naturals de Barcelona entendre tots els passos realitzats.

1.2. Marc Institucional.

Aquest projecte final de Màster es realitza gràcies a un conveni de col·laboració entre el Museu de Ciències Naturals de Barcelona i el Máster en Tecnologies d'Informació Geogràfica (15a edició) organitzat pel Departament de Geografia de la Universitat Autònoma de Barcelona.

Dins de l'entitat col·laboradora que s'ha esmentat abans, s'ha treballar a l'àrea d'Ecologia evolutiva i conductual que es dedica a estudis d'Ecoetologia de vertebrats, especialment d'aus. En aquest marc, s'integren aspectes com la dinàmica de poblacions, l'organització social, la comunicació i l'ecologia, per tal de conèixer i aprofundir en les estratègies que duen a terme els vertebrats per sobreviure i maximitzar la producció de descendència.

La durada aproximada d'aquest conveni de col·laboració es de quatre mesos (setembre – desembre 2013), sent de mitja jornada.

1.3. Antecedents.

Al Museu de Ciències Naturals de Barcelona, MCNB, s'han activat diverses línies de treball relacionades amb la informació geogràfica. El primer àmbit a desenvolupar criteris, projectes i aplicacions específiques de la gestió de dades geogràfiques ha estat el de la gestió científica de les col·leccions del museu.

La georeferenciació retrospectiva, la publicació de dades georeferenciades i estandarditzades i la participació ciutadana en la millora de les localitzacions de mostres de les col·leccions han estat motiu de sengles projectes plenament vigents. Es tracta de projectes de llarg recorregut i prou consolidats ja després d'anys d'experiència en un àmbit com el de la custòdia de patrimoni que s'inspira a llarg termini.

En els plans de recerca del Museu, la situació és diferent: domina la diversitat de propòsits, els canvis metodològics en funció dels objectius, els canvis de l'equip d'investigadors, etc. En aquest escenari es fa evident la necessitat de disposar de plataformes de treball pels programes d'investigació que sistematitzin, garanteixin la conservació i incrementin l'eficiència d'ús de les dades científiques vinculades a l'àmbit de recerca.

El MCNB desenvolupa un programa de recerca en ecologia evolutiva des de fa més de 15 anys en una finca d'unes 80 Ha, Can Catà, situada al Parc Natural de la Serra Collserola, dins del municipi de Cerdanyola del Vallès. Aquest espai té la particularitat de contenir una alta heterogeneïtat ambiental, amb microhàbitats ben determinats.

Els objectius de recerca s'adrecen a qüestions relacionades amb la variabilitat de les condicions individuals de supervivència, estratègies d'implicació en l'organització social i d'explotació dels recursos naturals, etc. En poblacions d'aus, bàsicament.

Durant aquest temps s'han mantingut sèries temporals d'estudis que s'han repetit any rere any o s'han desplegat estudis puntuals sobre aspectes concrets del pla de recerca específic de cada període.

Com a resultat es disposen d'arxius d'informació de resultats diversos associats la seva majoria a punts concrets de la finca de treball: nius, menjadores, trampes de captura i recaptura, etc. Amb tota aquesta informació més altres capes descriptives del territori s'ha anat acumulat un Sistema d'Informació Geogràfica amb el suport de MiraMon.

1.4. Objectius.

1.4.1 Objectius generals.

Aconseguir un Sistema d'Informació Geogràfica robust, coherent i molt flexible per facilitar consultes en cada moment i preveure el seu creixement any a any. A més, aquest SIG a de saber gestionar les múltiples capes que intervenen en l'inventari de riquesa temàtica de l'estació biològica de Can Catà

Desenvolupar una aplicació que permeti capturar les dades recollides al camp en un dispositiu mòbil.

1.4.2. Objectius específics.

- Anàlisi dels programaris de SIG lliures més idonis i elecció del que permetrà desenvolupar el SIG de recerca a Can Catà.
- Desenvolupar i documentar el procés de creació del SIG.
- Enriquir les capes descriptives del territori de la finca de Can Catà.
- Dissenyar un *front end* que doni peu a una senzilla gestió de les capes d'informació disponibles, tant de dades de recerca com de fons.
- El SIG ha de permetre suportar operacions bàsiques de consulta i exportació, tant espacials com alfanumèriques.
- Ha de suportar l'inventari i actualitzacions de la informació, tant alfanumèrica com cartogràfica.
- Ha de permetre generar cartografia en funció dels atributs alfanumèrics.
- L'aplicació mòbil ha de tenir la capacitat de poder consultar l'històric, a més de la capacitat d'introducció de dades.
- L'aplicació hauria de tenir incorporada cartografia base amb les diferents localitzacions de les caixes niu, i a poder ser interactuar amb aquesta per obtenir informació.
- L'aplicació mòbil hauria de permetre donar avisos en funció de l'estat de la caixa niu i dates destacades.
- Desenvolupar una aplicació mòbil que accepti la creació de plantilles de dades, contingui la capacitat d'incorporar automàticament la georeferenciació de cada punt i possiblement el vincle amb altres arxius multimèdia(imatge, so...).
- Els arxius resultants de l'aplicació i les seves dades s'haurien de connectar de forma transparent amb el SIG.

1.5. Informació disponible.

Per poder complir els objectius indicat a l'apartat 1.4., la informació alfanumèrica i cartogràfica que ha de contenir el SIG és la següent:

Informació	Descripció	Cartogràfica	Alfanumèrica
Caixes niu	Representació espacial de les caixes niu, amb coordenades UTM i geogràfiques	X	X
Estat	Possibles estats de les caixes-niu		X
Eclosió	Dates d'eclosió estimada i real dels ous, sortida estimada i real dels polls, entre d'altres		X
Identificació	Numero de les anelles de mascle i femella, nom de l'espècie, altres		X
Captura	Data de les captures, hores de captura dels mascles i femelles, hores d'activació i desactivació, hores d'alliberament, altres		X
Experiments	Data i nom de l'experiment		X

Figura 1.1. Tipus d'informació

1.6. Plataforma tecnològica.

A continuació es detalla el programari utilitzat en les diferents etapes de desenvolupament del sistema.

Fase I: Sistema d'Informació Geogràfica

MS Office Excel

Microsoft Excel és un component del paquet Microsoft Office. És una aplicació per manipular fulles de càlcul, normalment utilitzada per a operacions contables i financeres.

S'ha fet servir per a la creació de les sentències SQL que posteriorment han servit per a omplir la base de dades alfanumèrica de PostgreSQL. A més, tota la informació alfanumèrica inicial es trobava en aquest format.

DeZign for Databases V7

Es tracta d'una eina de la firma Datanamic molt útil per a modelitzar i dissenyar bases de dades per a múltiples plataformes, con per exemple Oracle, MySQL, IBM DB2, PostgreSQL, SQL

Server o MS Access entre d'altres. El programa utilitza diagrames d'entitat – relació per a possibilitar un disseny intuïtiu basat en gràfics de la base de dades, traduint automàticament el seu contingut en llenguatge SQL mitjançant la creació d'un script de text.

DeZign ofereix un entorn gràfic per a modelitzar no només els elements que hauran de participar en la base de dades, sinó també les relacions que els vinculen. D'aquesta manera és possible determinar el tipus de relació que guarda una entitat respecte a la resta, la seva cardinalitat i el paper dels seus atributs. Permet definir els camps de relació entre les taules, atorgant claus primàries i foranies, possibilitant la identificació d'errors en el disseny, així con la generació automàtica dels camps que hereta una entitat pel fet d'estar unida amb una altra. També proporciona l'opció d'estipular con a obligatòria la presència de valors no nuls en els registres, o la naturalesa dels diferents camps(tipus de dades que emmagatzema, longitud, descripció, etc).

Aquest programa s'ha utilitzat per a elaborar el disseny conceptual del sistema d'informació i el seu posterior desenvolupament lògic, així con per a la generació de la base de dades alfanumèrica.

PostgreSQL

PostgreSQL es un sistema de gestió de bases de dades objecte-relacional, distribuït sota la llicencia BSD i amb un codi font disponible lliurement. És el sistema de gestió de base de dades de codi obert més potent del mercat.

Utilitza un model client/servidor i utilitza multiprocessos per tal de garantir l'estabilitat del sistema. Un error en un dels processos no afectarà a la resta i el sistema continuarà funcionant. També, funciona molt bé amb gran quantitats de dades i una alta concurrència d'usuaris accedint alhora al sistema.

PostgreSQL s'ha utilitzat con a sistema de gestió de la base de dades del projecte. És a dir, per a la creació i implementació de la base de dades alfanumèrica. A més s'ha creat un sèrie de vistes amb sentències SQL per a consultar-les o analitzar-les al SIG d'escriptori de Quantum GIS.

PostGIS

PostGIS és un mòdul que afegeix suport d'objectes geogràfics a la base de dades objecte-relacional de PostgreSQL, convertint-se en una base de dades espacial per a la seva utilització en un Sistema d'Informació Geogràfica. Es publica sota la Llicència Pública General de GNU.

Ha estat desenvolupat per la empresa canadenca Refraction Research, especialitzada en productes Open Source. Un aspecte a tenir en compte, es que PostGIS garanteix la interoperabilitat amb altres sistemes.

Aquesta eina s'ha utilitzat com a base de dades espacial, la qual té relació amb la base de dades alfanumèrica de PostgreSQL i des de la que s'accedeix a partir de Quantum GIS.

Quantum GIS 1.8.0

QGIS és un Sistema d'Informació Geogràfica de Codi Obert llicenciat sota GNU (General Public License). QGIS és un projecte oficial d' *Open Source Geospatial Foundation* (OSGeo). Suporta nombrosos formats I funcionalitat de dades vectorials, dades ràster I bases de dades.

QGIS proporciona un continu creixement fluït de les capacitats proporcionades per les funcions bàsiques i complements. Es pot visualitzar, administrar, editar, analitzar dades i composar mapes imprimibles.

Aquest programa s'ha utilitzat per a poder realitzar la explotació del sistema d'informació geogràfica, tant per consultar com per analitzar les dades. També ha permès poder efectuar la connexió a PostGIS, per tal d'obtenir tota la informació desitjada.

Fase II: Aplicació mòbil

Eclipse – Android Development Tools(ADT)

Eclipse es una plataforma de desenvolupament obert formada per marcs extensibles, eines i temps d'execució per a la construcció, desplegament i gestió de programari a través del cicle de vida.

Android Development Tools (ADT) és un plugin per l'IDE Eclipse que està dissenyat per donar-li un entorn integrat per a la construcció d'aplicacions d'Android. ADT amplia les capacitats d'Eclipse per permetre configurar nous projectes per Android, crear una interfície d' usuari de l'aplicació, afegir els paquets basats en l'API d'Android Framework, depurar les seves aplicacions utilitzant les eines del SDK d'Android.

S'ha utilitzat com a entorn de treball per a la programació de tota la aplicació mòbil, amb el llenguatge Java.

Java

Java és un llenguatge de programació de propòsit general, concurrent, orientat a objectes i basat en classe que va ser dissenyat específicament per a tenir tan poques dependencies d'implementació com a fos possible.

Aquest ha estat el llenguatge utilitzat per a la programació mòbil del projecte final de màster, ja que és un llenguatge bastant senzill.

Fase III: Aplicació de transmissió de dades

Python

Python és un llenguatge de programació dinàmic i potent que s'utilitza en una àmplia varietat de dominis d'aplicació. Python té les següents característiques distintives: sintaxi molt clar i llegible, una forta capacitat d'introspecció, orientació a objectes intuïtiva, expressió natural del codi de procediment, maneig d'errors basat en excepcions, tipus de dades dinàmiques de molt alt nivell, entre d'altres.

Aquest llenguatge de programació s'ha utilitzat per al desenvolupament de l'aplicació de transmissió de dades.

2. FASE I: SISTEMA D'INFORMACIÓ GEOGRÀFICA

El següent esquema mostra les fases necessàries per al desenvolupament del Sistema d'Informació Geogràfica, què més endavant s'explicaran:

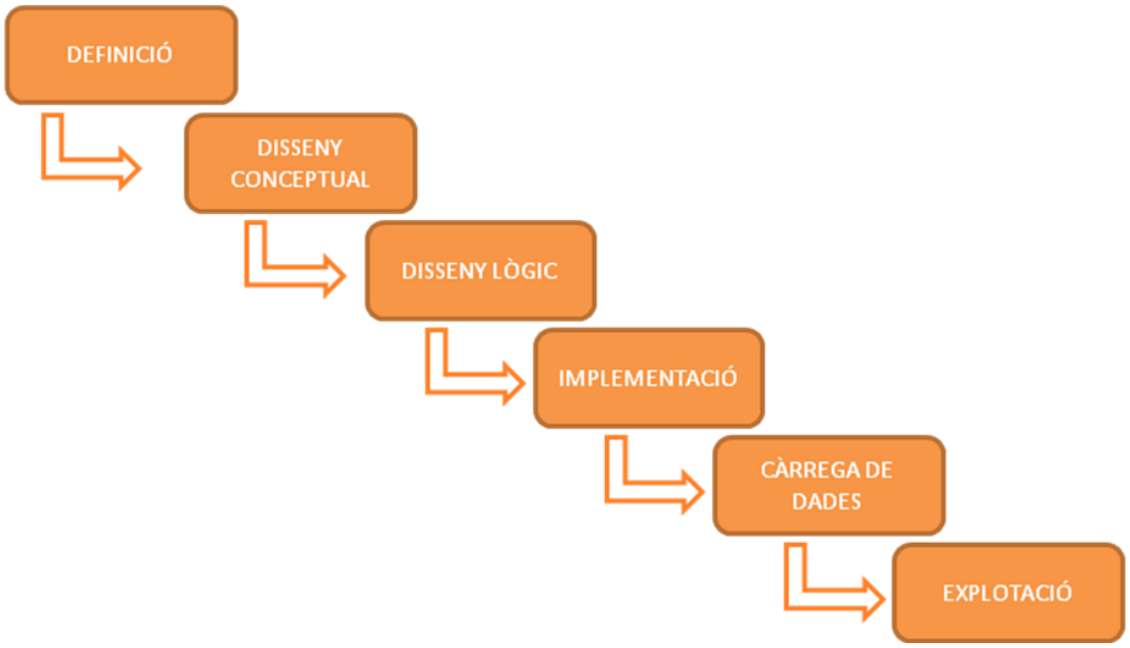

Figura 2.1. Fases per a la elaboració del SIG

2.1.Definició

Els Sistemes d'informació Geogràfica (SIG) són el resultat de l'aplicació de les anomenades Tecnologies de la Informació (TI) a la gestió de la Informació Geogràfica (IG).

El terme Sistema d'Informació Geogràfica té tres accepcions: el SIG com a disciplina; el SIG com a projecte, cadascuna de les realitzacions pràctiques, de les implementacions existents; el SIG com a software, és a dir els programes i aplicacions d'un projecte SIG.

L'accepció principal és la de SIG com a projecte, Sistema d'Informació que gestiona Informació Geogràfica, és a dir, informació georeferenciada. La definició més estesa de SIG, amb petites variacions, és la establerta pel Departament de Medi Ambient, Burrough, Goodchild, Rhin i altres. La qual es pot sintetitzar dient que un SIG es un :

« Conjunt integrat de medis i mètodes informàtics capaç de recollir, verificar, emmagatzemar, gestionar, actualitzar, manipular, recuperar, transformar, analitzar, mostrar i transferir dades espacialment referits a la Terra»

No obstant, es pensa que, tal i com mantenen Burrough i Bouillé, un SIG s'ha de veure també com un model del món real, pel que es podria definir com:

« Model informatitzat del món real, en un sistema de referència lligat a la Terra per a satisfer unes necessitats d'informació concretes»

El Museu de Ciències Naturals de Barcelona està desenvolupant un programa de recerca en ecologia evolutiva d'unes espècies concretes d'aus en una finca d'unes 80 Ha, Can Catà, on han col·locat unes 180 caixes niu georeferenciades. En aquesta finca, s'estudia el comportament, la conducta i l'evolució de les aus.

Per tant, el Sistema d'Informació proposat té com a finalitat general servir d'instrument per a la gestió de la informació geogràfica que s'ha obtingut durant aquestes sèries temporals d'estudis que s'han repetit any a any.

El SIG ha de ser coherent i molt flexible per facilitar tot tipus de consulta, és a dir, poder combinar tota la informació d'una manera clara.

A continuació, s'inclou l'esquema on es representen les diferents fases que s'han necessitat per a la realització del Sistema d'Informació Geogràfica.

2.2. Disseny Conceptual

El primer pas per a desenvolupar el Sistema d'Informació Geogràfica és elaborar el disseny conceptual. Aquest disseny és essencial, ja que si es cometen errors en aquest pas, s'aniran arrossegant durant tot el projecte.

En el disseny conceptual s'han de col·locar tots els actors que intervindran en la base de dades. És el pas previ a la creació del disseny lògic, què serà el que ens permetrà crear la nova BD.

Per a l'elaboració d'aquest model, s'han d'estudiar les dades que s'inclouran a la base de dades, per tal d'entendre com s'han de relacionar entre sí, ja que depèn com s'emmagatzemin les dades permetrà actualitzar-les o modificar-les en el futur.

Per dissenyar aquest model conceptual s'utilitza el model entitat-relació. Per tant, alhora de generar el disseny conceptual s'han d'identificar totes les entitats que conformaran las taules, i les relaciones que hi ha entre aquestes. Un cop identificades les entitats, s'han de definir els atributs, que són les característiques de cada entitat.

Les diferents entitats i atributs que s'han definit al dissenys conceptual són les següents:

- Caixes niu: Aquesta entitat ens permet saber quantes caixes niu hi ha, i quines coordenades té. Els atributs són: número de la caixa niu, codi del camí on es troben, coordenades UTM X, coordenades UTM Y, coordenades Z, Latitud i longitud.

- Estat: Són els diferents estats en que es pot trobar una caixa niu. Els seus atributs són: nom de l'estat i descripció del mateix.

- Captura: Són les dades que es recullen alhora de capturar les aus. Els atributs són els següents: número de polls, tipus de la trampa, hora d'activació, hora de captura del mascle, número de l'anella del mascle, hora de captura de la femella, número de l'anella de la femella, hora de desactivació, hora d'alliberament i observacions que es puguin obtenir durant la captura.

- Espècie: Són el nom de les diferents espècies que s'estudien. Els atributs són: nom científic, nom català i nom castellà de les diferents espècies.

- Identificació: com el seu propi nom indica, és la entitat que defineix la espècie i les anelles d'identificació de les dades obtingudes al camp. El seus atributs són: número de l'anella d'identificació del mascle i de la femella, l'any del qual s'han identificat, polls mesurats, l'alçada del niu fresc i observacions. Aquesta entitat està relacionada amb la d'espècie.

- Eclosió: equival a les dates de eclosions i sortides de l'ou. Té els següents atributs: número d'ous genètics, data del primer ou genètic, eclosió estimada, eclosió real, sortida estimada, sortida real, data d'anellar, si la neteja del niu s'ha produït o no, el número de polls adoptius que surten del niu, número d'ous buits i l'any de les dades referides a aquesta entitat.

- Experiment: són els diferents experiments que es porten a terme. Els atributs són: data i nom de l'experiment.

- Usuari: Dades de les persones que recullen les dades de camp. Els seus atributs són: nom i cognoms, NIF, telèfon de contacte, correu electrònic, nom d'usuari i contrasenya.

- Cita: Aquesta entitat és la via per la qual s'introdueixen les dades, és a dir, aquesta entitat està relacionada amb tota la resta d'entitats. En definitiva, es com si fos la taula que es portés al camp, i s'omplís amb totes les dades recollides. Els atributs són: tots els identificadors de les altres entitats, la data de la recollida de dades, informació addicional relativa a l'estat, el número total d'aranyes i de nius d'aranya que es troben a les caixes niu.

Les relacions entre entitats s'han intentat explicar, però s'entendran millor al següent punt, on es representarà mitjançant un diagrama el disseny lògic.

2.3. Disseny Lògic

Un cop dissenyat el disseny conceptual, s'ha de passar a la següent fase: elaborar el disseny lògic. Aquest model lògic s'ha creat a partir del programa Dezign for Databases. S'ha escollit aquest programa perquè permet definir el tipus de dades que es cada camp, a més de permetre escollir a quin Sistema Gestor de Bases de Dades anirà destinat. En aquest cas, a PostgreSQL.

Un cop creat el disseny lògic, Dezign ens permet crear automàticament les sentències SQL per a la posterior implementació de les taules a PostgreSQL.

Aquest és el resultat del disseny conceptual:

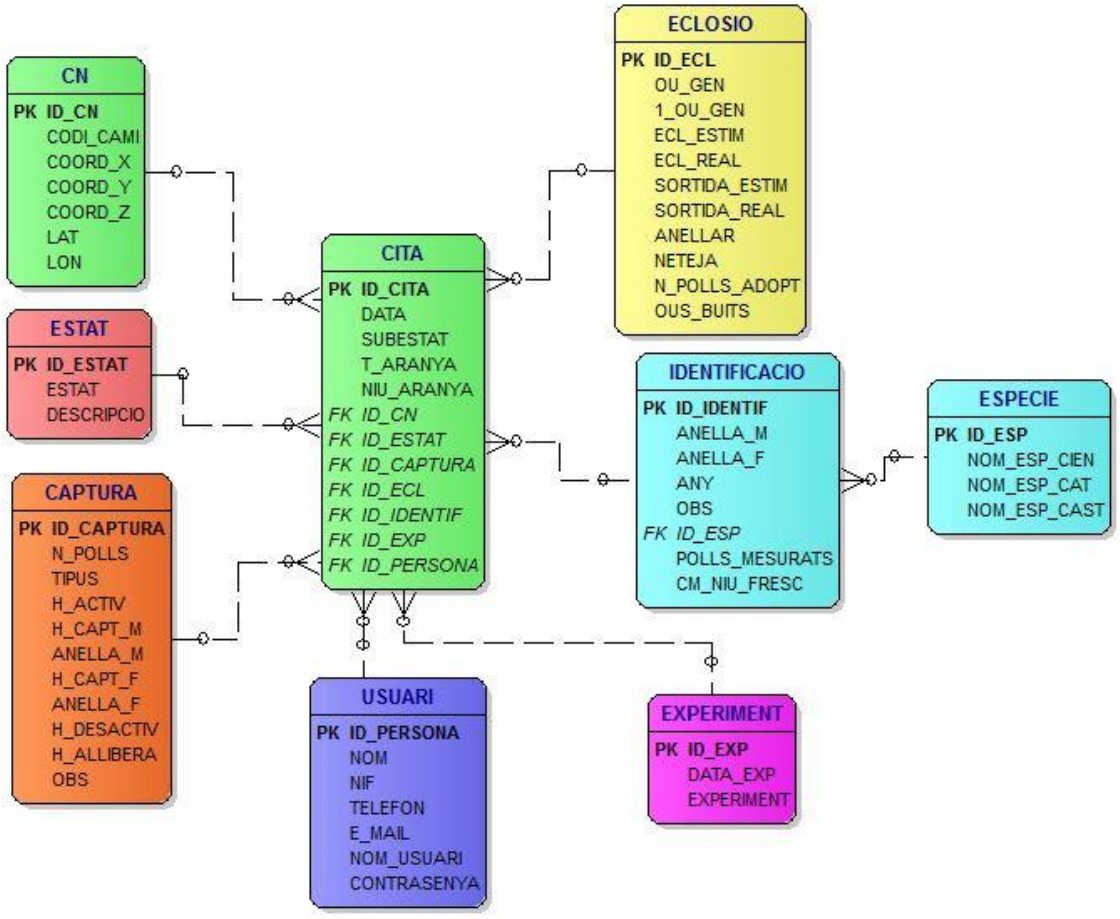

Figura 2.2. Disseny lògic de la base de dades

2.3.1. Relacions entre entitats

A continuació es descriuran les relacions que hi ha entre les diferents entitats per tal justificar el model lògic.

La primera relació és la que hi ha entre les entitats espècie i identificació. La cardinalitat d'aquesta relació és de un a molts, ja que cada espècie pot tenir més d'una identificació, però per cada identificació només hi pot haver una espècie.

Tota la resta d'entitats (cn, estat, captura, usuari, experiment, identificació, eclosió) es relacionen directament amb la entitat cita, ja que com s'ha explicat anteriorment, l'entitat cita serveix per recollir i relacionar totes les dades. Per tant, la relació és de un a molts, és a dir, qualsevol entitat pot tenir moltes cites, però una cita no pot tenir més d'una caixa niu, estat, captura, usuari, experiment,identificació, eclosió.

2.3.2. Definició de les taules

Seguidament es mostren de forma més detallada l'estructura de les taules definides pel disseny lògic:

CAPTURA					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_CAPTURA (not null)	BIGINT(100)	Identificador captura	PK_CAPTURA		
N_POLLS	INTEGER(10)	Número de polls			
TIPUS_TRAMPA	CHARACTER (20)	Tipus de trampa			
H_ACTIV	TIME (00:00)	Hora d'activació			
H_CAPT_M	TIME (00:00)	Hora de captura del mascle			
ANELLA_M	CHARACTER (20)	Número de l'anella del mascle			
H_CAPT_F	TIME (00:00)	Hora de captura de la femella			
ANELLA_F	CHARACTER (20)	Número de l'anella de la femella			
H_DESACTIV	TIME (00:00)	Hora de desactivació			
H_ALLIBERA	TIME (00:00)	Hora d' alliberament			
OBS_CAPT	CHARACTER (50)	Observacions			

CITA					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_CITA (not null)	BIGINT(100)	Identificador cita	PK_CITA		
ID_CN	INTEGER(10)	Identificador caixes-niu		PK_CN	CN
DATA	DATA (00/00/00)	Data			
ID_ESTAT	INTEGER(10)	Identificador del estat		PK_ESTAT	ESTAT
SUBESTAT	CHARACTER (20)	Subestat			
T_ARANYA	INTEGER(10)	Número total d'aranyes a la caixa-niu			
NIU-ARANYA	INTEGER(10)	Número de nius d'aranya a la caixa-niu			
ID_ECL	BIGINT(100)	Identificador eclosió		PK_ECL	ECLOSIO
ID_CAPTURA	BIGINT(100)	Identificador de la captura		PK_CAPTURA	CAPTURA
ID_IDENTIF	BIGINT(100)	Identificador identificació		PK_IDENTIF	IDENTIFICACIO
ID_EXP	BIGINT(100)	Identificador experiments		PK_EXP	EXPERIMENT
ID_USUARI	BIGINT(100)	Identificador usuari		PK_USUARI	USUARI

CN					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_CN (not null)	INTEGER(10)	Identificador caixes-niu	PK_CN		
CODI_CAMI	VARCHAR (20)	Codi del camí			
COORD_X	DOUBLE (20,5)	Coordenades UTM X			
COORD_Y	DOUBLE (20,5)	Coordenades UTM Y			
COORD_Z	DOUBLE (20,5)	Coordenades Z			
LAT	DOUBLE (20,5)	Latitud			
LON	DOUBLE (20,5)	Longitud			

ECLOSIO					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_ECL (not null)	BIGINT(100)	Identificador eclosió	PK_ECL		
OUS_GEN	INTEGER(10)	Número d'ous genètics			
PR_OU_GEN	DATA (00/00/00)	Data del primer ou genètic			
ECL_ESTIM	DATA (00/00/00)	Eclosió estimada			
ECL_REAL	DATA (00/00/00)	Eclosió real			
SORTIDA_ESTIM	DATA (00/00/00)	Sortida estimada			
SORTIDA_REAL	DATA (00/00/00)	Sortida real			
ANELLAR	DATA (00/00/00)	Data d'anellar			
NETEJA	CHARACTER(20)	Neteja realitzada			
N_POLLS_ADOPT	INTEGER(10)	Número de polls adoptius que surten del niu			
OUS_BUITS	INTEGER(10)	Ous buits.			
ANY_ECLOSIO	INTEGER(4)	Any de l'eclosió			

ESPECIE					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_ESP (not null)	INTEGER(10)	Identificador espècie	PK_ESP		
NOM_ESP_CIEN	CHARACTER (30)	Nom científic de l'espècie			
NOM_ESP_CAT	CHARACTER (30)	Nom català de l'espècie			
NOM_ESP_CAST	CHARACTER (30)	Nom castellà de l'espècie			

ESTATS_CN					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_ESTAT (not null)	INTEGER(10)	Identificador estat	PK_ESTAT		
ESTAT	CHARACTER (20)	Estat			
DESCRIPCIO	CHARACTER (200)	Descripció de l'estat			

EXPERIMENT					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_EXP (not null)	BIGINT(100)	Identificador experiment	PK_EXP		
DATA_EXP	DATA (00/00/00)	Data de realització de l'experiment			
NOM_EXP	CHARACTER (50)	Nom de l'experiment			

IDENTIFICACIO					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_IDENTIF (not null)	BIGINT(100)	Identificador identificació	PK_IDENTIF		
ANELLA_M_ID	VARCHAR (20)	Número de l'anella del mascle			
ANELLA_F_ID	VARCHAR (20)	Número de l'anella de la femella			
OBS_ID	VARCHAR(50)	Observacions			
POLLS MESURATS	CHARACTER (20)	Polls mesurats			
CM_NIU_FRESC	DOUBLE (5,2)	Alçada del niu fresc			
ID_ESP	INTEGER(10)	Identificador espècie		FK_ESP	ESPECIE

PERSONA					
Camp	Tipus	Descripció	Clau primària	Claus foràanies	Taula referida
ID_PERSONA (not null)	BIGINT(100)	Identificador persona	PK_USUARI		
NOM	VARCHAR (50)	Nom i cognoms de la persona que introdueix les dades			
NIF	VARCHAR (20)	NIF de la persona que introdueix les dades			
TELEFON	NUMERIC(9)	Número de telèfon de la persona que introdueix les dades			
E_MAIL	CHARACTER (50)	E-mail de la persona que introdueix les dades			
NOM_USUARI	CHARACTER (20)	Nom d'usuari de la persona que introdueix les dades			
CONTRASENYA	CHARACTER (20)	Contrasenya de la persona que introdueix les dades			

Taula 2.1. Definició de les taules de la base de dades

2.4. Implementació de la base de dades

Un cop realitzat el model lògic de la base de dades, el següent pas és la implementació d'aquesta al Sistema de Gestió de Base de Dades escollit, el qual és PostgreSQL.

El primer pas és generar els scripts per tal de crear les taules i les relacions entre aquestes. Aquest pas es realitza mitjançant el programa Dezign, ja que aquest permet crear automàticament el codi per a portar a terme la implementació al SGBD desitjat. Per tant, és un pas senzill i ràpid, en que només cal clicar la pestanya *Database* de la barra d'eines, i a continuació prémer a *Generate database*. El resultat d'aquest pas és la generació de dos arxius, un per crear la base de dades i un altre per esborrar-la.

Un cop generats els scripts, és el moment de crear la base de dades de PostgreSQL. En primer lloc s'ha d'engegar el programa d'administració de PostgreSQL anomenat pgAdmin III. En segon lloc, s'ha de realitzar la connexió al servidor de PostgreSQL 9.3. Un cop feta la connexió i creat l'usuari, ja es porta a terme la creació d'una nova base de dades. Per tal de crear-la, amb el botó dret s'ha de prémer a *Database* i a continuació a *New database*. Fets aquests petits passos, sorgirà una nova finestra on s'hauran de definir els paràmetres que es vulguin per a la base de dades. Els paràmetres introduïts en aquest cas són els següents:

- Nom: bd_sigcancata
- Owner: postgres
- Encoding: UTF8
- Template: template_postgis
- Tablespace: pg_default

La resta de paràmetres quedaran per defecte.

Un tercer pas, es crear una sèrie de sentències SQL per tal de convertir als identificadors de les taules en auto numèrics. Aquestes sentències s'executaran a PostgreSQL, on quedaran emmagatzemades. Es crearà una seqüència per cada taula que necessiti que el seu identificador sigui auto numèric, ja que totes no ho necessiten, com per exemple les caixes niu, on el seu identificador és el número de la caixa niu, per tant no ha de ser auto numèric.

Es crearan aquestes seqüències per a les següents taules: captura, cita, eclosió, experiment, identificació i usuari. Un exemple de sentència:

```
CREATE SEQUENCE autonum
INCREMENT 1
MINVALUE 1
MAXVALUE 999999999999999
START 1
```

Un cop creades les sentències, queda afegir-les al codi generat pel Dezign, per tal que alhora de crear les taules, ja tinguin incorporat les sentències per tal de que l'identificador sigui auto numèric. Un cop modificat el codi de creació de taules, és el moment de executar el codi per tal d'implementar les taules a la base de dades de PostgreSQL.

Un exemple de creació d'una taula:

```
CREATE TABLE IDENTIFICACIO (
   ID_IDENTIF BIGINT  NOT NULL DEFAULT NEXTVAL('autonum_identif'),
   ANELLA_M CHARACTER(20),
   ANELLA_F CHARACTER(20),
   ANY_IDENTIF NUMERIC(4),
   OBS CHARACTER(50),
   POLLS_MESURATS CHARACTER(20),
   CM_NIU_FRESC DOUBLE PRECISION,
   ID_ESP INTEGER,
   CONSTRAINT PK_IDENTIFICACIO PRIMARY KEY (ID_IDENTIF)
);
COMMENT ON COLUMN IDENTIFICACIO.ID_IDENTIF IS 'Identificador identificació';
COMMENT ON COLUMN IDENTIFICACIO.ANELLA_M IS 'Número de l''anella del mascle';
COMMENT ON COLUMN IDENTIFICACIO.ANELLA_F IS 'Número de l''anella de la femella';
COMMENT ON COLUMN IDENTIFICACIO.ANY_IDENTIF IS 'Any';
COMMENT ON COLUMN IDENTIFICACIO.OBS IS 'Observacions';
COMMENT ON COLUMN IDENTIFICACIO.POLLS_MESURATS IS 'Polls mesurats';
COMMENT ON COLUMN IDENTIFICACIO.CM_NIU_FRESC IS 'Alçada del niu fresc (cm)';
```

2.5. Càrrega de dades

Ja implementada la base de dades, ara és el torn de la càrrega de dades, és a dir, omplir les taules amb tota la informació.

El Museu de Ciències Naturals de Barcelona, degut a la seva trajectòria d'estudis d'aquest àmbit, tenia dades en format Excel des de l'any 1998 fins al 2013, ininterrumpidament. Per aquest motiu es decideix crear les sentències SQL a partir d'Excel. Generar els scripts ha estat una taca relativament llarga, ja que en un primer moment s'han hagut d'homogeneïtzar totes les taules per tal de poder portar a terme una inserció de dades correcta. Un cop realitzada la homogeneïtzació, mitjançant la concatenació dels camps desitjats per a la inserció de dades, es creen les diverses sentències SQL que després s'executaran a PostgreSQL.

Aquesta concatenació anirà precedida de la seqüència INSERT en llenguatge SQL. EL resultat és el següent:

INSERT INTO IDENTIFICACIO(ANELLA_M,ANELLA_F,ANY_IDENTIF,OBS,ID_ESP) VALUES ('L743307','L743288','2010',null,'1');

Una vegada s'hagi realitzat aquest procés amb totes les dades s'obtindran el conjunt de sentències SQL, què executant-les a PostgreSQL ompliran totes les taules. Però, alhora de executar els scripts generals s'han de vigilar l'ordre en que s'executen ja que l'ordre haurà de ser l'adequat per a complir les restriccions de les claus foranies.

Un ordre correcte seria el següent:

CN → ESTAT → CAPTURA → ECLOSIO → USUARI → EXPERIMENT → ESPECIE → IDENTIFICACIO → CITA

Feta la càrrega de dades de la BBDD alfanumèrica, ara és el torn de la càrrega de dades espacial. Per portar a terme aquest procés s'ha d'obrir Quantum GIS. Un cop obert s'ha de crear una connexió amb la base de dades alfanumèrica. Per a realitzar la connexió s'ha d'anar a la barra d'eines i clicar en *Añadir capas PostGIS*, i a continuació prémer a *Nueva* per a crear una nova connexió. Seguidament, s'obrirà una nova finestra on s'haurà d'omplir amb els següents paràmetres:

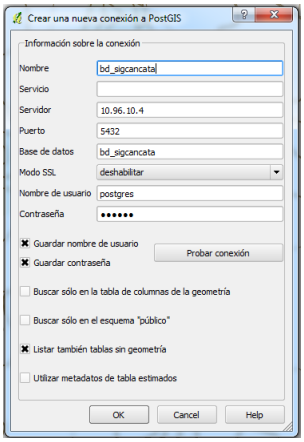

Figura 2.3. Connexió a PostGIS

En aquest moment, es quan s'ha de definir el nom de la connexió, el servei, el servidor, el port, la base de dades a la que es vol fer la connexió, el nom d'usuari i contrasenya i altres. Fet això, ja està creada la connexió a la base de dades.

El següent pas a seguir és obrir PostGIS Manager, un plugin de Quantum GIS. Anant a la barra d'eines de Quantum GIS i clicant a *Base de datos > PostGIS Manager*. Un cop realitzada aquesta operació, ja es tindrà accés a la base de dades de PostgreSQL mitjançant PostGIS.

Ara per portar a terme la càrrega de dades espacial, que en aquest cas només és la càrrega de dades de un shapefile, el de les caixes-niu georeferenciades. Per realitzar aquesta operació s'ha de polsar el boto *Data > Load Data From Shapefile*. I a la finestra emergent s'hauran d'omplir amb diferents paràmetres:

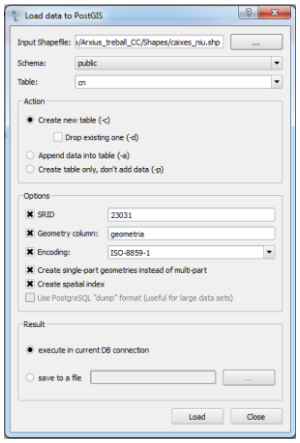

Figura 2.4. Carregar capa a PostGIS

Es defineix el shapefile, l'esquema on està la base de dades i la taula a la qual té relació. També es concreta el sistema de referència del shapefile entre altres paràmetres.

Amb aquests últims passos ja s'ha realitzat la càrrega de dades de la base de dades alfanumèrica i espacial, és a dir, s'ha finalitzat la creació del Sistema d'Informació Geogràfica.

2.6. Explotació del Sistema d'Informació Geogràfica

El Sistema d'Informació Geogràfica elaborat servirà merament com un mitjà de consulta o com un mitjà per exportar les taules en altres format per tal de poder analitzar-les posteriorment amb profunditat. La informació de la base de dades es podrà consultar directament des de PostgreSQL o des de el SIG d'escriptori Quantum GIS.

Per a que el client pugui realitzar consultes i analitzar-les correctament, s'ha portar a terme l'elaboració de diverses vistes de dos tipus diferents:

1. Vistes segons l'any d'obtenció de dades i el concepte, és a dir, segons identificació, eclosió, experiment, estat o captura.

2. Vistes històriques, és a dir, engloba tots els anys en sola una taula, però segons la identificació, eclosió, experiment, estat o captura. D'aquesta manera es pot observar més eficientment la evolució d'aquestes aus, el qual és el tema d'estudi.

Un exemple del codi de la creació d'una vista es el següent:

```
CREATE VIEW v_2012_identif AS SELECT cita.id_cita, cn.id_cn, cn.coord_x, cn.coord_y,
cn.coord_z, identificacio.anella_m_id, identificacio.anella_f_id, especie.nom_esp_cien,
identificacio.any_identif, identificacio.polls_mesurats, identificacio.cm_niu_fresc,
identificacio.obs_id
FROM cn, cita, identificacio, especie
WHERE cn.id_cn = cita.id_cn AND cita.id_identif = identificacio.id_identif AND
identificacio.id_esp = especie.id_esp AND identificacio.any_identif = 2012::numeric
ORDER BY cn.id_cn;
```

Un cop creades aquestes vistes, és poden visualitzar, combinar i analitzar des de Quantum GIS. Per afegir les vistes al QGIS, només cal clicar sobre *Añadir capas PostGIS*, connectar-se a la connexió creada anteriorment, i en un primer moment només es veuran les capes cartogràfiques, però activant la pestanya *Listar también tablas sin geometria*, apareixeran totes les taules, tant les inicials, com les vistes creades. Per afegir les taules només cal definir quin és la seva clau primària, la qual és id_cita. Un cop definides ja es poden afegir.

A continuació, per combinar diferents taules o combinar-les amb la capa de caixes niu, només cal anar a les propietats de la taula, clicar la pestanya *Uniones*. En aquest moment s'hauran de definir els camps comuns entre les dues taules a combinar per a portar a terme la unió de taules. El camp comú que sempre servirà per fer les unions és el número de la caixa niu, és a dir, el camp id_cn.

A continuació es mostra un exemple de dues taules combinades, en aquests cas la taula de caixes niu amb la d'identificació de l'any 2013:

Taula resultant de la combinació entre les dues taules:

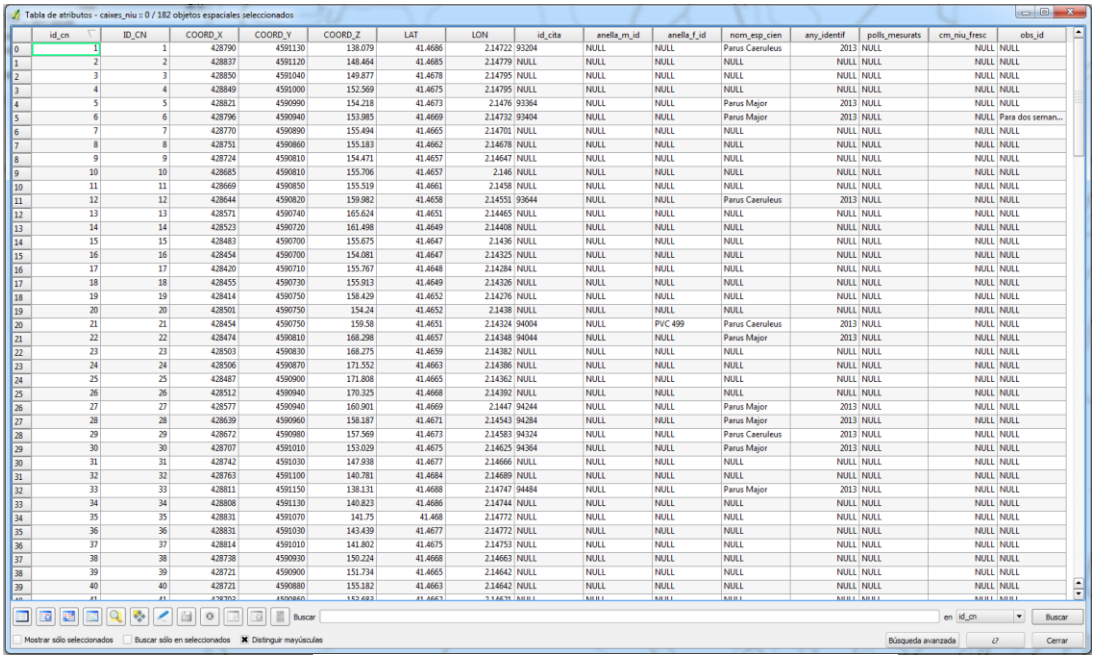

Figura 2.5. Taula de combinació entre taules

Resultat de la consulta sobre una caixa niu on s'observen els atributs d'ambues taules degut a la unió d'aquestes:

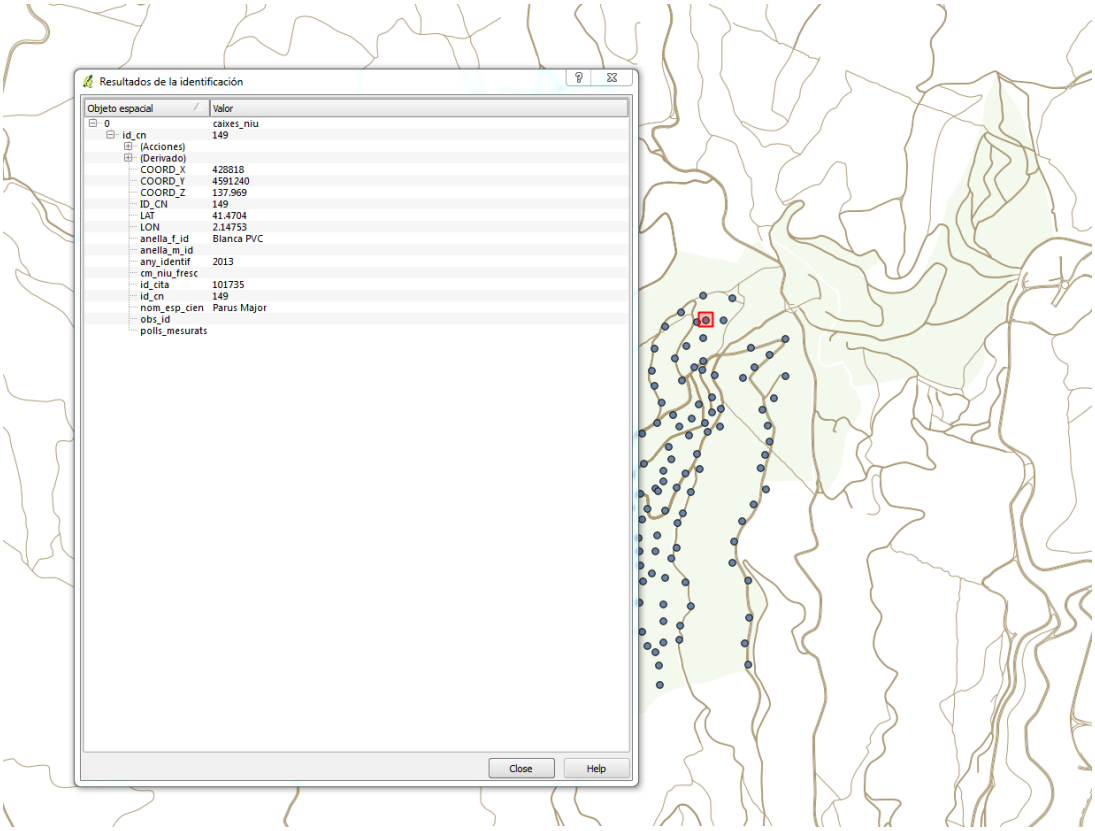

Figura 2.6. Identificació Quantum GIS

Per acabar, el client desitja poder exportar les taules a altres formats. Hi ha dues maneres de fer-ho, des de PostgreSQL o des de Quantum GIS.

1. Des de PostgreSQL. Aquest mètode d'exportació és més complicat que el següent. Per exportar les taules des de PostgreSQL s'ha d'executar l'script de la taula que es vol exportar prement el botó *Execute Query, write result to write*. Fent aquest procés les taules es poden exportar en format csv, el qual es pot llegir amb Microsoft Excel.

2. Des de Quantum GIS. L'exportació des de QGIS és molt més fàcil i intuïtiva, ja que el que sa de fer és clicar amb el botó dret sobre la taula que es desitja exportar i prémer a Guardar como. Un cop feta aquesta operació, només cal decidir en quin format es vol exportar, ja que hi ha uns quants. Els formats més destacats d'exportació són els següents: Fitxer shapefile d'ESRI, AutoCAD DXF, GeoJSON, kml, gml, csv, entre d'altres.

3. FASE II: APLICACIÓ MÒBIL

Abans de començar la explicació sobre l'aplicació i la seva programació, cal esmentar que és Android i perquè s'ha escollit aquest sistema operatiu.

Android és un sistema operatiu inicialment pensat per telèfons mòbils, tabletes i altres dispositius, igual que iOS, Symbian i Blackberry OS. El que el fa diferent és que està basat en Linux, un nucli de sistema operatiu lliure, gratuït i multi plataforma

El desenvolupament de programes per a Android és fa habitualment amb el llenguatge de programació Java i el conjunt d'eines de desenvolupament(SDK, Software Development Kit). Aquest últim comprèn un depurador de codi, biblioteca, un simulador de telèfon basat en QEMU, documentació, exemples de codi i tutorials.

El sistema permet programar aplicacions en una variació de Java anomenada Dalvik. El sistema operatiu proporciona totes les interfícies necessàries per a desenvolupar aplicacions que accedeixin a les funcions del telèfon d'una forma molt senzilla en un llenguatge de programació mol conegut com és Java.

Una de les millors característiques d'aquest sistema operatiu és que es completament lliure, és a dir, ni per a programar en aquest sistema ni per a incloure al telèfon s'ha de pagar res. I el que el fa molt popular entre fabricants i desenvolupadors, ja que els costos per a llençar una aplicació són molt baixos.

Qualsevol pot descarregar-se el codi font, inspeccionar-lo, compilar-lo o inclús modificar-lo. Això dóna seguretat als usuaris, ja que quelcom és obert permet detectar errors més ràpidament.

Per tant, s'ha escollit aquest sistema operatiu, sobretot, perquè és lliure i de relativa fàcil programació.

3.1. Requeriments

Els requeriments de l'aplicació mòbil de l'empresa, i per tant els objectius a assolir són els següents:

- L'aplicació mòbil ha de tenir la capacitat de poder consultar l'històric, a més de la capacitat d'introducció de dades.
- L'aplicació hauria de tenir incorporada cartografia base amb les diferents localitzacions de les caixes niu, i a poder ser interactuar amb aquesta per obtenir informació.
- L'aplicació mòbil hauria de permetre donar avisos en funció de l'estat de la caixa niu i dates destacades.

3.2. Estructura de l'aplicació

L'aplicació mòbil consta d'activitats (requadres taronges), les quals són la component principal encarregada de mostra a l'usuari la interfície gràfic, és a dir, una activitat seria l'equivalent a una finestra. Es defineix una activitat per cada interfície del projecte. Els elements que és mostra en aquesta han de ser definits en el fitxer xml que porten associat (que es guarda a *./res/layout*). Dins del fitxer xml associat a l'activitat, es defineixen els elements con la ubicació dels elements de la pantalla (*layouts*), botons, textos, *spinners*, etc.

Con el llenguatge utilitzat durant la programació de l'aplicació mòbil és Java, un llenguatge orientat a objectes, l'aplicació també comptarà amb classes (requadres blancs). Una classe no es més que un concepte que alhora d'implementar-lo es composa de variable juntament amb un conjunt de funcions que s'apliquen sobre aquestes.

L'estructura de l'aplicació mòbil de recollida de dades és la següent:

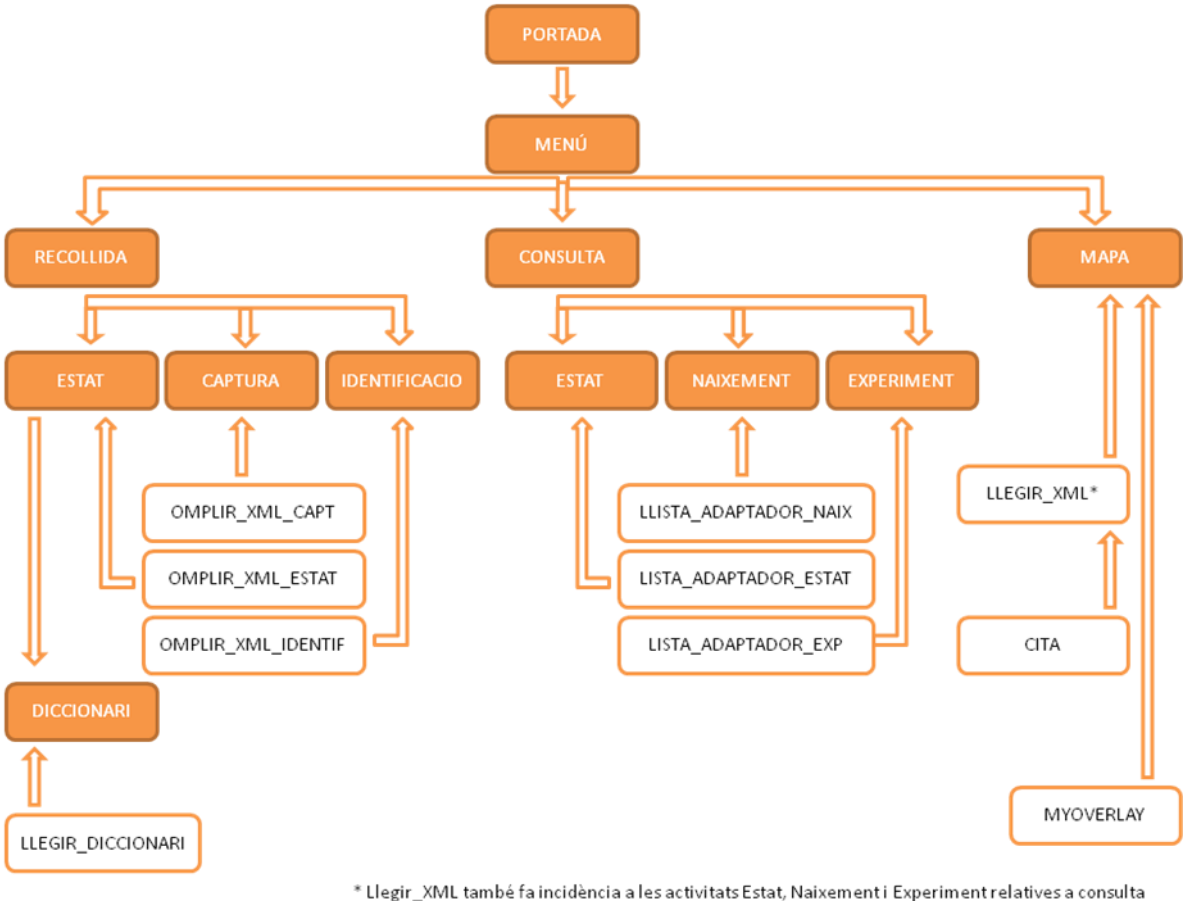

Figura 3.1. Estructura de l'aplicació mòbil

L'aplicació mòbil es divideix en tres parts ben diferenciades: recollida de dades, consulta i mapa. Pel que fa a la part de recollida, se centrarà en la recol·lecció de dades relatives a estat, captura i identificació, mentre que pel que respecta a consulta, se centrarà en estat, naixement i experiment. Alhora de recollir dades referents a l'estat, es comptarà amb una taula diccionari per poder veure els significat dels diferents estats possibles. A més, hi ha una tercera part, la cartogràfica, on és mostrarà el mapa de la zona d'estudi amb la localització de les caixes niu.

Cadascuna d'aquestes tres parts tindrà relacionada una sèrie de classes per tal de poder portar a terme la programació d'una manera satisfactòria.

3.3. Programació de l'aplicació

Per a la elaboració del projecte s'ha utilitzat eclipse com a entorn de treball ja que és una plataforma de desenvolupament obert formada per marcs extensibles, eines i temps d'execució per la construcció, desplegament i gestió de programari a través del cicle de vida. És un bon entorn de treball ja que permet depurar codi, el qual facilita la feina alhora de saber on està l'error. A més el llenguatge de programació que s'ha fet servir ha estat Java.

Per començar la programació el primer que s'ha de fer és crear un nou projecte, on per defecte es crearà una primera activitat, amb el seu xml associat.

A continuació s'explica el procediment que s'ha seguit per al desenvolupament de l'aplicació mòbil. Explicant l'Android Manifest, el menú i les tres parts de l'aplicació: recollida, consulta i mapa

3.3.1. Android Manifest

Android Manifest és un arxiu xml, el qual és un dels més importants de qualsevol aplicació Android. Es genera automàticament alhora de crear el projecte, i en aquest es troba definida la configuració del projecte en xml (activitats, intents, permisos, etc).

En aquest fitxer també es defineix quina és el nivell d'API mínima requerida per a que l'aplicació funcioni. El nivell d'API que s'ha definit com a la mínima requerida és la 8. Aquesta API anomenada per Android, Froyo, equival a la versió 2.2. També és pot definir el nivell de l'API objectiu a la que va dirigida l'aplicació. Si no es declara, agafarà per defecte la versió mínima establerta. S'ha concretat l'API 18 com a versió destí. Aquesta API equival a la versió 4.3. d'Android, anomenada Jelly Bean.

També, a Android Manifest s'han de concretar permisos per al bon funcionament de l'aplicació. En el cas d'aquest projecte només se sol·licita un permís, el qual demana l'accés d'escriptura a l'emmagatzematge extern.

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="18" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

3.3.2. Menú

Com s'ha dit amb anterioritat, al crear un projecte nou, genera una activitat nova. És en aquesta activitat on es crearà el menú que donarà pas a les tres parts de l'aplicació. Però abans d'això és crearà una nova activitat per tal de que faci la funció de portada, és a dir, serà la primera activitat que pareixerà a l'iniciar l'aplicació.

Per a crear una nova activitat, s'ha de prémer amb el botó dret sobre el projecte, i clicar *New > Other > Android Activity*. És en aquest moment quan és donarà nom a l'activitat i al xml associat que serveix per dissenyar la interfície. El nom que se li donarà és *SplashActivity*. Com l'objectiu d'aquesta activitat és que faci de capçalera de l'aplicació, s'haurà de modificar l'Android Manifest, per tal de que sigui la primera activitat en aparèixer.

Per poder realitzar aquesta operació s'haurà de definir un *intent-filter* dins de l'activitat *SplashActivity* on s'indicarà com a acció, *main*. I com a categoria, *launcher*. Això farà que sigui l'activitat d'inici.

```
<activity
    android:name="com.example.can_cata.SplashActivity"
    android:label="@string/title_activity_splash" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

A continuació, és dissenyarà la interfície d'aquesta activitat. Per tant, s'haurà de treballar amb l'arxiu xml associat a aquesta activitat. Aquest s'anomena *activity_splash.xml*. En aquest es treballarà sobre un *RelativeLayout* on s'incorporarà a l'interior d'aquest dues *ImageView* i un *TextView*. Les *ImageView* s'utilitzaran per mostrar una fotografia i el logo de l'entitat col·laboradora, en aquest cas el logo del Museu de Ciències Naturals de Barcelona. Però prèviament aquestes imatges s'hauran d'introduir a la carpeta *./res/drawable* dins del projecte. Un cop introduïdes s'hauran de definir dins de cada *ImageView* de la següent manera: `android:src="@drawable/nom_imatge"`. I el *TextView* s'utilitzarà per al títol.

El resultat és el següent:

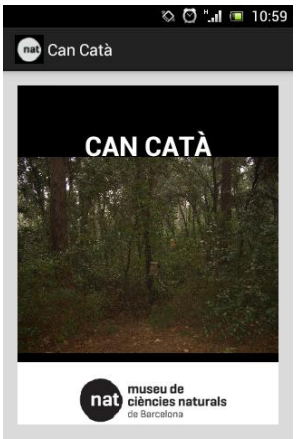

Figura 3.2. Activitat inicial de l'aplicació mòbil

Dissenyada la interfície, ara és el moment de crear un mètode per a aquesta activitat. Donat que l'objectiu és que al tocar a qualsevol part de la pantalla del dispositiu mòbil passi a la següent activitat, el mètode que s'usarà serà *onTouchEvent*. Aquest mètode té com a objecte *MotionEvent*, el qual tancarà la activitat *SplashActivity* i obrirà *MainActivity*, on s'implementarà el menú.

```
public boolean onTouchEvent (MotionEvent event) {
    startActivity(new Intent(SplashActivity.this, MainActivity.class));
    SplashActivity.this.finish();
    return true;
}
```

Un cop realitzada la activitat inicial de l'aplicació és crearà el menú a l'activitat que s'ha creat per defecte al crear un nou projecte. Aquesta activitat és *MainActivity*. Primer de tot s'inclouran quatre botons al xml associat a aquesta activitat. El primer botó serà per a consulta, el segon per a la recollida de dades, el tercer per a la part cartogràfica i el quart per a sortir de l'aplicació.

Creats els botons, ara és el torn de crear les activitats que s'obriran al ser clicats. És crearan tres activitats noves, una per a consulta, una altra per a recollida i una tercera per al mapa. Fet aquest pas, al *MainActivity* es crearan els mètodes que realitzaran la funció de passar d'una activitat a una altra. Aquest mètode serà el següent:

```
public void verConsulta(View v) {
    //Clicar el botó i portar-te a l'activity: ConsultaActivity
    Intent act = new Intent(this, ConsultaActivity.class);
 startActivity(act);
}
```

Seguidament aquest mètode s'ha d'instanciar d'aquesta manera al xml associat dins del botó que correspongui: `android:onClick="nom_mètode"`. Això farà que al prémer el botó s'obri la activitat desitjada.

I per acabar amb el menú, es programarà el botó creat que correspon a sortir i el botó d'anar enrere del dispositiu mòbil. L'objectiu es que al clicar qualsevol d'aquest botó s'obri un quadre de diàleg que pregunti si desitja sortir. Una vegada creat el mètode s'instanciarà de la mateixa manera que els anteriors.

El codi és el següent:

```
public void sortir (View v) {
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
        // Titol
        alertDialogBuilder.setTitle("Segur que vols sortir?");
        // missatge de diàleg
        alertDialogBuilder
            .setCancelable(false)
            .setPositiveButton("Si",new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,int id) {
                    // Si cliques aquest botó, es tanca la app
                    MainActivity.this.finish();
                }
              })
            .setNegativeButton("No",new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,int id) {
                    // Si cliques aquest botó es tanca el quadre de diàleg i no es tanca la app
                    dialog.cancel();
                }
            });

            // Crea l'alerta de diàleg
            AlertDialog alertDialog = alertDialogBuilder.create();

            // Es visualitza el diàleg
            alertDialog.show();
    };
```

Fetes aquestes operacions ja està programat el menú.

Figura 3.3. Menú de l'aplicació mòbil

3.3.3. Recollida de dades

L'objectiu d'aquesta part és crear uns formularis que emmagatzemin les dades recollides al camp, i a partir d'aquests generin un fitxer xml, que mitjançant una petita aplicació programada amb Python (veure 4.2.1.) es transmetrà al Sistema d'Informació Geogràfica.

Alhora de crear el menú, també s'havia creat una activitat per a la recollida de dades. La funció d'aquesta activitat serà la de derivar cap als tres tipus de recollida de dades: Estat, Captura i Identificació. Per a portar a terme aquest pas, es necessitaran tres botons, que obriran tres noves activitats amb els formularis corresponents. Per a portar a terme aquesta operació se seguiran els mateixos passos que s'han dut a terme durant l'elaboració del menú.

Figura 3.4. Menú de recollida de l'aplicació mòbil

Anteriorment s'ha creat tres activitats relatives a la recollida de dades referent a l'estat, captura i identificació.. El primer que s'ha de realitzar és dissenyar la interfície d'aquestes activitats, és a dir, elaborar els formularis que s'hauran d'omplir al camp durant la recollida de dades. Per tant, s'hauran de modificar els fitxers xml associats a aquestes activitats. En aquests arxius s'incorporaran *TextViews*, E*ditText*, algun que altre *spinner* i botons per tal de confirmar la introducció de les dades.

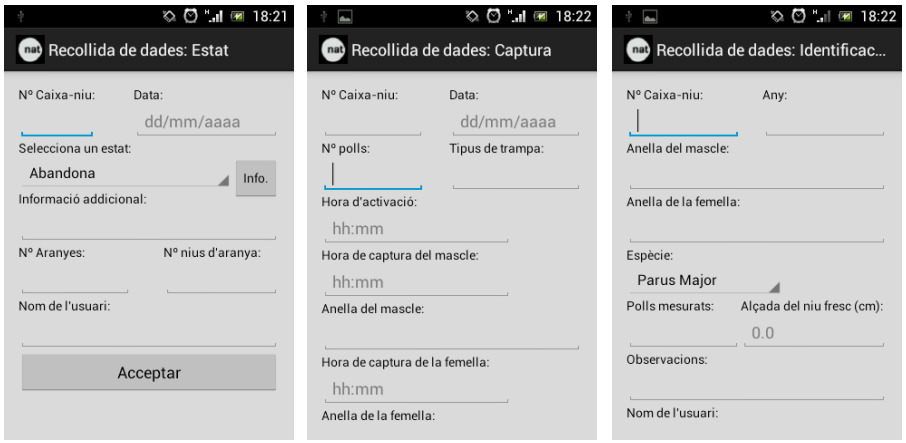

Figura 3.5. Formularis de recollida de dades de l'aplicació mòbil

Els *TextViews* serviran per indicar què s'ha d'introduir a cada *EditText*. Els *EditText* serviran per a introduir les dades recollides i el *spinner* s'utilitzarà per a escollir l'estat en què es troba el niu i la espècie dels ocells que ocupen aquestes caixes niu. Els botons s'usaran per acceptar i confirmar la introducció de dades. I en el cas del formulari de recollida de dades relativa a estat, comptarà amb un botó extra, el qual serà per mostrar la descripció dels estats.

Per a omplir els *spinner* que inclouran els estats i les espècies, s'hauran de crear dos *arrays*, un pels estats i un altre per les espècies, en un fitxer xml que es col·locarà a la carpeta *./res/values* del projecte. Un cop creats els *arrays*, a les activitats on aniran, s'hauran de generar uns *arrayAdapter* per a cada *spinner* al *onCreate* de l'activitat corresponent per a poder omplir el *spinner* amb els *arrays* creats anteriorment.

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this,R.array.estat, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

spn_estat.setAdapter(adapter);
```

Un cop dissenyats el formularis d'introducció de dades d'aquestes activitats, s'hauran de crear una classe per a cada activitat, per tal de poder generar un xml a partir de les dades recollides al camp. L'objectiu d'aquestes classes és omplir un xml buit on s'aniran afegint totes les dades recollides, i què després aquestes classes es cridaran des de cada activitat. Abans de programar aquestes classes s'haurà de crear un xml buit, és a dir, un xml amb un únic node, on al seu interior s'aniran afegint tots els registres recollits. Aquest xml buit es col·locarà a una carpeta creada dins de la memòria externa del dispositiu mòbil, anomenada CanCata.

El primer pas per crear les classes que ompliran el xml és generar un DocumentBuilderFactory, un DocumentBuilder i un Document, per tal d'accedir posteriorment al xml buit creat anteriorment o al fitxer que és generarà a l'insertar la primera cita. Per accedir a aquest fitxer s'ha de fer el següent:

```
try {
    doc = docBuilder.parse(new InputSource(new FileInputStream(
            Environment.getExternalStorageDirectory().getPath() + "/CanCata/recollida_CanCata.xml")));

} catch (FileNotFoundException fileError) {
    doc = docBuilder.parse(new InputSource(new FileInputStream(
            Environment.getExternalStorageDirectory().getPath() + "/CanCata/buit_CanCata.xml")));
}
```

El resultat és què si es troba el primer fitxer, recollida_CanCata.xml, accedirà a aquest i si no el troba accedirà al xml buit. Tots dos es trobaran a la mateixa carpeta de la memòria externa del dispositiu mòbil. En definitiva, a la primera cita que s'introdueixi s'accedirà al fitxer buit,

mentre que a la resta de cites successives accediran al xml recollida_CanCata.xml, el qual es crearà alhora d'insertar la primera cita a partir del fitxer buit.

A continuació es generarà un node nou, anomenat id_cita, que serà fill del node principal creat al fitxer buit. Cada node d'aquests significarà una cita nova. Immediatament després, dins d'aquest nou node creat es generaran els nodes fills d'id_cita que faran referència a cada dada registrada al camp, juntament amb els seus valors corresponents.

```
Element cita= doc.createElement("id_cita");
dataroot.appendChild(cita);

Element cn = doc.createElement("id_cn");
if (RecollidaEstatActivity.txt_cn.getText().toString().compareTo("")!=0) {
    cn.appendChild(doc.createTextNode(RecollidaEstatActivity.txt_cn.getText().toString()));
} else {
    cn.appendChild(doc.createTextNode(" "));
}
cita.appendChild(cn);
```

I per últim, es programarà l'escriptura del xml amb tota la informació obtinguda dels *EditText*. S'accedirà al document i crearà un nou arxiu a partir del xml buit en la introducció de la primera cita, i se sobreescriurà aquest fitxer amb la resta de dades introduïdes sense solapar-se unes amb altres. Aquest és el procediment:

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(
        Environment.getExternalStorageDirectory().getPath() + "/CanCata/recollida_CanCata.xml");
transformer.transform(source, result);
```

Per últim, a les activitats on es troben els formularis s'instanciaran aquestes classes què s'activaran alhora de prémer el botó acceptar. Cada cop que es premi aquest botó es crearà una nova cita al xml amb totes les dades introduïdes al formulari.

A l'activitat de recollida relativa a estat també es troba un botó d'informació. Aquest botó obre una nova activitat, anomenada *DiccionariEstat* on es descriuran tots els estats possibles de les caixes niu. El primer pas és elaborar un fitxer de text on es redacti les descripcions d'aquests estats. Un cop generar aquest fitxer, es col·locarà dins la carpeta abans creada a la memòria externa del dispositiu mòbil, anomenada *CanCata*.

Es crearà una classe que servirà per portar a terme la lectura del fitxer de text. El primer pas serà obrir el fitxer de text accedint a la memòria externa. El segon pas serà crear l'objecte d'entrada i un *buffer* de lectura. El tercer pas serà generar un bucle per tal de que vagi llegint el fitxer línia a línia, on s'aniran concatenant totes les línies. I finalment es tanca l'arxiu i es mostra a un *TextView* abans introduït al *Layout* de l'activitat *DiccionariEstat*.

```
// Obre l'arxiu
FileInputStream fstream = new FileInputStream(
        Environment.getExternalStorageDirectory().getPath() + "/CanCata/estat_dic_CanCata.txt");
// Crea l'objecte d'entrada
DataInputStream entrada = new DataInputStream(fstream);
// Crea el Buffer de Lectura
BufferedReader buffer = new BufferedReader(new InputStreamReader(entrada));
// Llegeix l'arxiu linia per linia
while ((strLinea = buffer.readLine()) != null)   {
    //Concatena totes les linies
    strLineaResult = strLineaResult + "\n\n" + strLinea;
}
//DiccionariEstat.txt_diccestat.setText(strLineaResult);
// Tanca l'arxiu
entrada.close();
//Mostra el text
txtInfo.setText(strLineaResult);
```

El resultat d'aquesta activitat és el següent:

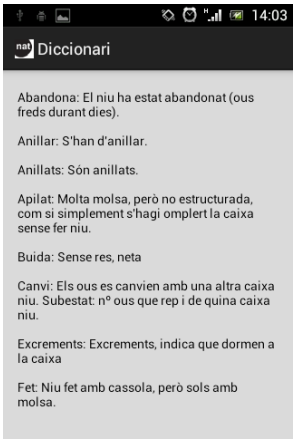

Figura 3.6. Activitat diccionari de l'aplicació mòbil

3.3.4. Consulta de dades

L'objectiu d'aquesta part es poder consultar dades exportades en un fitxer xml des del Sistema d'Informació Geogràfica creat amb informació relativa a estat, identificació, naixement i experiment (Veure 4.2.2.), mitjançant *ListViews*.

Durant la creació del menú principal és va crear una activitat referent a la consulta. Aquesta activitat, com en el cas de l'activitat de recollida de dades, servirà per a derivar cap a altres activitats mitjançant botons. Es crearan tres activitats noves, una per a estat/identificació, una altra per a naixement i una última per a experiment.

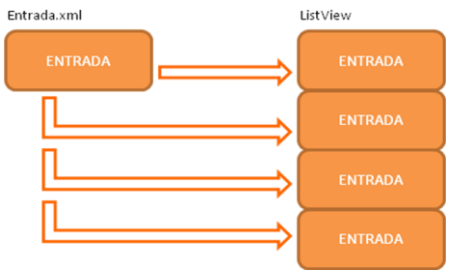

Figura 3.7. Esquema del ListView

El primer pas és afegir un *ListView* al *Layout* de cada activitat, i crear un altre xml independent on es dissenyarà només una entrada per al *ListView*, que després es relacionarà a partir d'un adaptador. En aquesta entrada s'afegiran tants *TextViews* com a camps per consultar hi hagi.

El segon pas és elaborar dues classes, una que s'anomenarà Cita i una altra que serà *LlegirXML*. La primera servirà per emmagatzemar en variables tots els valors que es vagin llegint i la segona, com el seu propi nom indica, serà la que vagi recorrent i llegint tot el xml.

En la classe Cita, es crearà una variable per a cada node del xml que és vulgui mostrar a la *ListView*. Es farà de següent manera:

```
private Integer id_cn;

 public Integer getId_cn() {
  return id_cn;
 }
 public void setId_cn(Integer id_cn) {
  this.id_cn = id_cn;
 }
```

Un cop creada aquesta classe, s'ha de crear la classe *LlegirXML*. El primer que és farà es crear un *ArrayList* per a anar omplint la classe Cita, que com s'ha dit són variables que es van omplint a mesura que es vagi llegint el fitxer xml.

A continuació, es genera un *DocumentBuilderFactory*, un *DocumentBuilder* i un *Document* per tal d'obrir el xml que es generi a partir de la exportació des del SIG. El fitxer es col·locarà a la memòria externa del dispositiu mòbil dins la carpeta abans creada anomenada CanCatà. Un

cop obert aquest fitxer es comença a llegir mitjançant un bucle que anirà recorrent tots els nodes i emmagatzemant els valors dins de les variables de la classe cita de la següent manera:

```
NodeList nList = doc.getElementsByTagName("cita");

for (int temp = 0; temp < nList.getLength(); temp++) {
    Cita cita = new Cita();

        Node nNode = nList.item(temp);

        if (nNode.getNodeType() == Node.ELEMENT_NODE) {

            Element eElement = (Element) nNode;

            if (eElement.getElementsByTagName("data").item(0).getTextContent()
                    .toString().toLowerCase().compareTo("none") == 0){
                cita.setData("-");
            }else{
                cita.setData(eElement.getElementsByTagName("data").item(0).getTextContent());
            }

            datos.add(cita);
```

Finalitzada aquesta operació, l'aplicació mòbil ja llegeix el xml, però no el mostra. Per tant, ara és el moment de crear l'adaptador per tal de visualitzar i poder consultar tota la informació. Aquest adaptador servirà per relacionar el *ListView* de les activitats amb el xml de les entrades, a més d'omplir els *TextViews* de les entrades amb les dades emmagatzemades a la classe Cita.

Es crearà un contenidor per a cada *TextView*, el qual després s'omplirà amb els valors emmagatzemats a la taula Cita.

```
Cita cita = (Cita) getItem(arg0);

contenedor.num_cn.setText("Caixa-niu : " + cita.getId_cn().toString());

contenedor.data_estat.setText("Data: " + cita.getData().toString());

contenedor.estat.setText("Estat: " + cita.getEstat().toString());

contenedor.subestat.setText("Informació addicional: " + cita.getSubestat().toString());

contenedor.t_aranya.setText("Nº d'aranyes: " + cita.getT_aranya().toString());

contenedor.niu_aranya.setText("Nº de nius d'aranya: " + cita.getNiu_aranya().toString());

contenedor.anella_m_id.setText("Anella del mascle: " + cita.getAnella_m_id().toString());

contenedor.anella_f_id.setText("Anella de la femella: " + cita.getAnella_f_id().toString());

contenedor.nom_esp_cien.setText("Espècie: " + cita.getNom_esp_cien().toString());
```

Ara ja només cal instanciar la classe *LlegirXML* i l'adaptador a les activitats on es troba el *ListView*. És en aquest moment quant és relaciona l'adaptador amb el *ListView*, per tal d'omplir-lo amb totes les entrades que necessiti.

Figura 3.8. Activitats de consulta de dades

3.3.5. Mapa

L'objectiu és mostrar una base cartogràfica offline de l'àmbit de recerca amb les localitzacions de les caixes niu. A més s'haurà de poder interactuar amb aquestes per tal de poder obtenir informació de consulta. I per últim diferenciar les caixes niu segons l'estat en que es troben.

Per a la realització d'aquesta part, s'ha obtat per utilitzar la llibreria MapsForge basada en OpenStreetMaps. Els motius de l'elecció d'aquesta llibreria es perquè té una representació gràfica ràpida de les dades d'OpenStreetMpas, el seu format d'arxius és compacte, és 100% lliure i de codi obert, es pot executar en qualsevol dispositiu mòbil amb Android 1.5 o superior, entre d'altres. Per tant s'ha d'incluir la llibreria mapsforge-map-0.3.0-jar-with-dependencies.jar al projecte.

Un cop inclosa la llibreria, s'ha de descarregar el mapa de l'àmbit de recerca del museu. Aquest mapa ha d'esta en format .map. Descarregat el mapa, s'ha d'incloure a la memòria externa del dispositiu mòbil dins de la carpeta creada anteriorment, anomenada CanCata. Ara ja està tot preparat per a començar amb la programació de visualització del mapa.

El mapa es visualitzarà en una activitat que derivarà del menú d'inici. En el layout d'aquesta activitat es crearà un MapView a partir de la llibreria de MapsForge. Serà en aquest MapView on es visualitzarà el mapa.

Per portar a terme la visualització del mapa es crearà la variable de MapView on es recuperarà l'arxiu .map col·locat a la memòria externa del dispositiu mòbil. També s'establiran altres paràmetres com el zoom inicial, el punt central, els controls de zoom i finalment que el mapa sigui visible.

```
MapView mapView = new MapView (this);
mapView.setClickable(true);;
mapView.setBuiltInZoomControls (true); //Controls de zoom
mapView.getController().setZoom(15); //zoom inicial
mapView.getController().setCenter(new GeoPoint(41.467, 2.144)); //Geopoint d'inici.

String filepath=Environment.getExternalStorageDirectory().getPath() + "/CanCata/CanCata.map";
mapView.setMapFile (new File(filepath));
setContentView(mapView); //visualització del mapa
```

Per mostrar les localitzacions de les caixes niu s'haurà d'instanciar la classe LlegirXML, ja que a l'exportació també s'exporten les coordenades d'aquestes. Els passos a seguir són col·locar els símbols que representaran les caixes niu que es dibuixaran al mapa ala carpeta ./res/drawable, crear una nova classe per tal que faci la funció de dibuixar els punts al mapa, què serà la classe *MyOverlay extends ItemizedOverlay*. Un cop creada aquesta classe, es programarà que segons l'estat de la caixa niu agafi un símbol o un altre per tal de poder diferenciar a simple vista quins nius tenen un estat o un altre.

Visualitzats els punts de les caixes niu, ara es farà que al tocar sobre la pantalla una caixa niu del mapa ens retorni informació. Per poder portar a terme aquesta operació, a la classe creada per visualitzar els punts es crearà el mètode onTap el qual generarà un quadre de diàleg el qual mostrarà tota la informació que és llegeix quan s'executa la classe LlegirXML.

Figura 3.9. Mapa de l'aplicació mòbil

4. FASE III: APLICACIÓ DE TRANSMISSIÓ DE DADES

4.1.Requeriments

Els requeriment i per tant els objectius principals per al desenvolupament d'aquesta petita aplicació programada amb Python són el següents:

- L'aplicació ha de permetre transmetre les dades del dispositiu mòbil al Sistema d'informació Geogràfica mitjançant sentències SQL.
- L'aplicació ha de poder exportar dades del Sistema d'informació Geogràfica en un fitxer xml per a que després el pugui llegir l'aplicació mòbil, i per tant, poder consultar la informació exportada.
- Aquesta aplicació ha de tenir un fitxer de configuració on apareguin les dades de connexió a la base de dades de PostgreSQL, i amb les rutes de importació i exportació dels xml.
- I per últim, l'aplicació ha d'estar compilada, és a dir, han d'haver dos fitxers compilats, és a dir, un per a la importació i un altre per a la exportació.

4.2. Programació de l'aplicació

Abans de començar la programació s'ha de crear un fitxer Python de configuració on s'incorporaran les variable que equivaldran als paràmetres necessaris per a la connexió a PostgreSQL i les rutes d'importació i exportació dels fitxers xml. El codi és el següent:

```
#dades per a la connexio de la base de dades de postgresql
localhost = 'xx.xx.xx.x'
db = 'nom_de_la_base_de_dades'
usuari = 'nom_usuari'
pword = 'contrasenya'

#Ruta per a la exportacio de dades del SIG cap a aplicacio mobil
ruta_export_sig = r'D:\xxxxxxx\consulta_CanCata.xml'
#Ruta per a la importacio al SIG de dades recollides al camp
ruta_import_sig = r'D:\xxxxxxx\recollida_CanCata.xml'
```

Aquest arxiu s'haurà d'incorporar a una carpeta que Python sigui capaç d'interpretar, per tal de poder importar-lo als fitxers que es crearan a continuació.

4.2.1. Transmissió de dades del dispositiu mòbil al Sistema d'Informació Geogràfica

L'objectiu d'aquest pas es poder importar la informació recollida al camp i emmagatzemada en un xml al Sistema d'Informació Geogràfica. Per a portar a terme això, es generarà una fitxer Pyrhon, el qual llegirà el xml i crearà sentències SQL per tal d'introduir les dades al SIG.

Primer de tot s'ha de realitzar la connexió a PostgreSQL, per tan s'ha d'importar l'adaptador psycopg2, que permet connectar-se a una base de dades de PostgreSQL. A més, s'ha d'importar el fitxer de configuració abans creat, els qual emmagatzema els paràmetres de connexió. Un cop realitzades aquestes importacions és fa el següent:

```
try:

    con = psycopg2.connect(host=localhost, database=db, user=usuari, password=pword)
    cur = con.cursor()

except psycopg2.DatabaseError, e:
    if con:
        con.rollback()

    print 'Error %s' % e
    sys.exit(1)
```

S'haurà de realitzar una tercera importació per tal de poder portar a terme la lectura del xml generat per l'aplicació mòbil. És la següent:

```
from xml.dom import minidom
```

A continuació, és crearan quatre funcions, una per tal d'obtenir una llista amb els identificadors i estats, nom d'espècie i nom d'usuari, i una altra per tal de que retorni l'identificador dels estats, noms d'espècie i nom d'usuari. I una tercera per a obtenir l'últim identificador de les taules d'eclosió i d'identificació. I una última que obtingui el nom dels nodes del xml.

Seguidament, s'instanciarà la ruta on es trobi l'arxiu a llegir, el qual abans s'haurà col·locat a la ruta indicada des del dispositiu mòbil. Un cop indicat on es troba el fitxer que es llegirà per tal de realitzar les sentències SQL, es definirà quin tipus d'INSERTS s'han d'efectuar, ja que hi trobarem tres tipus: INSERT INTO CITA; INSERT INTO CAPTURA; INSERT INTO IDENTIFICACIO. Per tant, es generaran bucles que aniran recorrent el xml per tal de saber quin tipus d'insert són. Serà de la següent manera:

```
for n in  xmldoc.childNodes :
 for id_cita in n.childNodes:
  textInsert = ""
  typeConsulta = ""
  sqlString = ""
  textInsertCita = "insert into cita (id_cn, data, id_estat, subestat, t_aranya,"
  "niu_aranya, id_persona" + ") values ("
  textInsertCaptura = "insert into captura (n_polls, tipus_trampa, h_activ, h_capt_m,"
  "anella_m, h_capt_f, anella_f, h_desactiv, h_allibera, obs_capt" + ") values ("
  textInsertIdentificacio = "insert into identificacio (any, anella_m_id, anella_f_id,"
  "especie, polls_mesurats, cm_niu_fresc, obs_id" + ") values ("
  listaInserts = []
  campos = []
  campoval = []

  for registro in id_cita.childNodes:
   if registro.nodeType == minidom.Node.ELEMENT_NODE:
    if ((returnField(registro.nodeName) == "estat")):
     typeConsulta = "cita"
     textInsert = textInsertCita
     break
    elif ((returnField(registro.nodeName) == "n_polls")):
     typeConsulta = "captura"
     textInsert = textInsertCaptura
     break
    elif ((returnField(registro.nodeName) == "any")):
     typeConsulta = "identificacio"
     textInsert = textInsertIdentificacio
     break
```

Es generaran les sentències SQL que s'executaran a PostgreSQL per tal d'introduir les dades al SIG. Aquest procediment es portarà a terme de la següent forma. Depenent de quin tipus d'insert sigui, s'escriurà una sentència o una altra. També es contemplarà el camps que estiguin buit, és a dir, en el cas que no s'omplin dades, les dades seran introduïdes con a null.

```
textInsertCita = "insert into cita (id_captura, id_identif, id_cn,"
"data, id_estat, subestat, t_aranya, niu_aranya, id_persona" + ") values ("
textInsertCaptura = "insert into captura (n_polls, tipus_trampa, h_activ,"
"h_capt_m, anella_m, h_capt_f, anella_f, h_desactiv, h_allibera, obs_capt"
+ ") values ("
textInsertIdentificacio = "insert into identificacio (any_identif, anella_m_id,"
"anella_f_id, id_esp, polls_mesurats, cm_niu_fresc, obs_id" + ") values ("

if listaInserts[0] == "cita":
 textInsertCita = textInsertCita + "null, null, " + parseTipusCamp(listaInserts[2][0],'num') +
 ", " + parseTipusCamp(listaInserts[2][1],'text') + ", " +
 parseTipusCamp(str(returnId(estats, listaInserts[2][2])),'num') + ", " +
 parseTipusCamp(listaInserts[2][3],'text') + ", " + parseTipusCamp(listaInserts[2][4],'num') +
 ", " + parseTipusCamp(listaInserts[2][5],'num') + ", " +
 parseTipusCamp(str(returnId(personas, listaInserts[2][6])),'num') + ");"
 print textInsertCita
 executeInsert(cur,textInsertCita)
 con.commit()
```

I per acabar, un cop insertada la informació recollida al camp, es canviarà el nom del fitxer xml amb la data del dia de la importació.

4.2.2. Transmissió de dades del Sistema d'Informació Geogràfica al dispositiu mòbil

L'objectiu es poder exportar informació de la base de dades de PostgreSQL en un fitxer xml per poder després consultar-lo al dispositiu mòbil.

El primer pas és realitzar la connexió a la base de dades de PostgreSQL. Primer s'ha d'importar l'adaptador psycopg2, per a que permeti connectar-se a PostgreSQL, i l'arxiu de configuració, ja que emmagatzema els paràmetres de connexió a la base de dades. Un cop fet això és realitza la connexió establint el host, nom de la base de dades, usuari i contrasenya d'aquesta. És a dir, la connexió s'estableix de la mateixa manera que quan la importació.

Un cop realitzada la connexió a la base de dades, es crearan tres variables, una pels camps que es volen exportar de la base de dades, una altra per al nom que portaran els nodes del xml, i una tercera que retornarà una llista amb tots els tags d'una cadena de text. A continuació és realitzarà la consulta sql que permetrà obtenir tota la informació que es vol consultar al camp.

Per a poder crear el xml a partir de la canulta SQL realitzada es farà la següent importació:

```
from xml.etree.ElementTree import Element, SubElement, tostring
```

Feta aquesta importació, ara només queda crear un bucle per a que es vagi generant el xml de manera correcta. Aquest bucle anirà recorrent tots els registres i recuperant els noms dels nodes. Un cop realitzat el bucle, es crearà un fitxer xml el qual es col·locarà a la ruta desitjada, i s'omplirà amb totes les dades exportades.

```
for i in range(len(rows)):
    cita = SubElement(dataroot, "cita")
    for l in range(len(listCampos)):
        var = SubElement(cita, listCampos[l])
        var.text = str(rows[i][l])

f = open(ruta_export_sig', 'w')
f.write(tostring(dataroot))
f.close()
```

Una vegada obtingut aquest fitxer, es mourà a la carpeta CanCata de la memòria externa del dispositiu mòbil per que sigui llegit per l'aplicació de recollida i consulta de dades.

5. Conclusions

Tot i que la consecució de les diferents fases no s'ha donat dins del calendari previst es pot dir a grans trets, que els objectius s'han complert de manera satisfactòria.

La primera fase del projecte final de màster era crear un Sistema d'Informació Geogràfica per tal d'emmagatzemar, consultar i interpretar dades recollides a la finca de Can Catà, al Parc Natural de Collserola, per part d'un grup de recerca del Museu de Ciències Naturals de Barcelona. Aquesta fase s'ha desenvolupat sense cap inconvenient greu, ja que totes les parts que s'han necessitat per a dur a terme el desenvolupament del SIG s'han produït de manera satisfactòria. Per tant, els objectius relatius a aquesta fase s'han complert.

La segona fase era desenvolupar una aplicació mòbil Android per tal de poder consultar i recollir dades a la zona d'estudi, en aquest cas, Can Catà. Els objectius generals d'aquesta fase s'han aconseguit, ja que l'aplicació mòbil permet consultar dades exportades del Sistema d'Informació Geogràfica, a més de poder recollir les dades necessàries per al programa de recerca en ecologia evolutiva del Museu. L'aplicació també permet visualitzar les caixes niu en una base cartogràfica, el qual també era un objectiu. Per tant a grans trets s'han complert tots els objectius, però hi ha un que no s'ha pogut completar en la seva totalitat. Aquest és el que l'aplicació hauria de permetre rebre avisos segons estats, dates importants, etc. Respecte aquest objectiu, l'únic que s'ha pogut dur a terme és simbolitzar el mapa segons determinats estats. Per tant, no s'ha complert aquest objectiu en la seva totalitat, ja que per exemple no rep avisos sobre la data en que eclosiona un ou. En definitiva, s'ha dissenyat una aplicació senzilla i funcional.

La tercera fase, programar una aplicació de transmissió de dades, s'ha desenvolupat complint tots els objectius, ja que aquesta permet exportar informació des del Sistema d'informació Geogràfica per després poder consultar-la al dispositiu mòbil. I també permet importar al SIG les dades recollides amb l'aplicació mòbil. Per tant, aquesta fase s'ha aconseguit.

La realització d'aquest projecte ha estat un autèntic repte, ja que mai havia desenvolupat un Sistema d'informació Geogràfica des de zero pels meus propis medis. I, a més, les aplicacions desenvolupades s'han programat amb llenguatges que mai havia utilitzat i, per tant, desconeixia.

Per acabar aquestes conclusions, val a dir que l'aplicació mòbil pot presentar algunes millores en el futur, ja que degut al temps limitat per a poder realitzar el projecte, m'he hagut de centrar en les funcionalitat principals de recollida i consulta de dades. Algunes millores futures podrien ser el mètodes d'exportació, ja que es podrien realitzar d'una manera més directa o remotament. A més és podrien incorporar algunes eines a la part cartogràfica. I per últim, generar avisos, com s'ha esmentat anteriorment.

6. Referències

Referències bibliogràfiques:

- Burnette, Ed. *Android*. Anaya Multimedia. Madrid. 2010.

- Burnette, Ed. *Android 2*. Anaya Multimedia. Madrid. 2011.

- Ribas Lequerica, Joan. *Desarrollo de aplicaciones para Android*. Anaya Multimedia. Madrid. 2011.

- Montero Miguel, Roberto. *Desarrollo de aplicaciones para Android*. Ra-Ma. Paracuellos de Jarama. 2012.

Referències electròniques:

- http://www.postgresql.org/

- http://developer.android.com/guide/index.html

7. Annexes

7.1. Programació de l'aplicació de transmissió de dades: fitxer de configuració

```
#dades per a la connexio de la base de dades de postgresql
localhost = 'xx.xx.xx.x'
db = 'nom_de_la_base_de_dades'
usuari = 'nom_usuari'
pword = 'contrasenya'

#Ruta per a la exportacio de dades del SIG cap a aplicacio mobil
ruta_export_sig = r'D:\xxxxxxx\consulta_CanCata.xml'
#Ruta per a la importacio al SIG de dades recollides al camp
ruta_import_sig = r'D:\xxxxxxx\recollida_CanCata.xml'
```

7.2. Programació de l'aplicació de transmissió de dades: del dispositiu mòbil al SIG

```
from xml.dom import minidom
import psycopg2
import sys
import configcancata as cc

con = None
try:
 con = psycopg2.connect(host=cc.localhost, database=cc.db,
       user=cc.usuari, password=cc.pword)
 cur = con.cursor()

except psycopg2.DatabaseError, e:
 if con:
  con.rollback()
  print 'Error %s' % e
  sys.exit(1)

def ompleEs(cur, sqlSelect):
 cur.execute(sqlSelect)
 rows = cur.fetchall()
 listaTabla = []
 campoid = []
 campodesc = []
 for row in rows:
  campoid.append(row[0])
  campodesc.append(row[1])
 listaTabla = [campoid,campodesc]

 return listaTabla
```

```
def returnId(lista, valor):
 index = 0
 for i in range(len(lista[1])):
  if lista[1][i].strip().lower() == valor.lower():
   index = lista[0][i]
   break
 return int(index)

def autonum(cur, sqlID):
 cur.execute(sqlID)
 row = cur.fetchall()
 campoAutonum = row[0]
 return campoAutonum


estats = ompleEs(cur, "select id_estat, estat from estats_cn")
especies = ompleEs(cur, "select id_esp, nom_esp_cien from especie")
personas = ompleEs(cur, "select id_persona, nom from usuari")
```

```
def returnField(fieldMovil):
 if fieldMovil == "id_cn":
  return "id_cn"
 elif fieldMovil == "data":
  return "data"
 elif fieldMovil == "estat":
  return "estat"
 elif fieldMovil == "subestat":
  return "subestat"
 elif fieldMovil == "t_aranya":
  return "t_aranya"
 elif fieldMovil == "niu_aranya":
  return "niu_aranya"
 elif fieldMovil == "nom":
  return "nom"
 elif fieldMovil == "tipus_trampa":
  return "tipus_trampa"
 elif fieldMovil == "h_activ":
  return "h_activ"
 elif fieldMovil == "h_capt_m":
  return "h_capt_m"
 elif fieldMovil == "anella_m":
  return "anella_m"
 elif fieldMovil == "h_capt_f":
  return "h_capt_f"
 elif fieldMovil == "anella_f":
  return "anella_f"
 elif fieldMovil == "h_desactiv":
  return "h_desactiv"
 elif fieldMovil == "h_allibera":
  return "h_allibera"
 elif fieldMovil == "obs_capt":
  return "obs_capt"
```

```
elif fieldMovil == "any_identif":
 return "any_identif"
elif fieldMovil == "anella_m_id":
 return "anella_m_id"
elif fieldMovil == "anella_f_id":
 return "anella_f_id"
elif fieldMovil == "especie":
 return "especie"
elif fieldMovil == "polls_mesurats":
 return "polls_mesurats"
elif fieldMovil == "cm_niu_fresc":
 return "cm_niu_fresc"
elif fieldMovil == "obs_id":
 return "obs_id"
else:
 return fieldMovil

def ompleInsertsValues(textValues, data):
 if textValues == "":
  textValues = "'" + data +"'"
 else:
  textValues = textValues + ",'" + str(data) + "'"
 return textValues


def dalecanya():
 text = ""
 xmldoc=minidom.parse(ruta_import_sig)
 fileid=xmldoc.getElementsByTagName('id_cita')

for n in  xmldoc.childNodes :
 for id_cita in n.childNodes:
  textInsert = ""
  typeConsulta = ""
  sqlString = ""
  textInsertCita = "insert into cita (id_cn, data, id_estat, subestat,"
  "t_aranya, niu_aranya, id_persona" + ") values ("
  textInsertCaptura = "insert into captura (n_polls, tipus_trampa, h_activ,"
  "h_capt_m, anella_m, h_capt_f, anella_f, h_desactiv, h_allibera, obs_capt" + ") values ("
  textInsertIdentificacio = "insert into identificacio (any, anella_m_id,"
  "anella_f_id, especie, polls_mesurats, cm_niu_fresc, obs_id" + ") values ("
  listaInserts = []
  campos = []
  campoval = []
```

```
for registro in id_cita.childNodes:
 if registro.nodeType == minidom.Node.ELEMENT_NODE:
  if ((returnField(registro.nodeName) == "estat")):
   typeConsulta = "cita"
   textInsert = textInsertCita
   break
  elif ((returnField(registro.nodeName) == "n_polls")):
   typeConsulta = "captura"
   textInsert = textInsertCaptura
   break
  elif ((returnField(registro.nodeName) == "any")):
   typeConsulta = "identificacio"
   textInsert = textInsertIdentificacio
   break

for registro in id_cita.childNodes:
 if registro.nodeType == minidom.Node.ELEMENT_NODE:
  campos.append(returnField(registro.nodeName))
  campoval.append(registro.firstChild.data)

listaInserts = [typeConsulta,campos,campoval]
if listaInserts[0]!='':
 print listaInserts[0]
 createInserts(listaInserts)

def executeInsert(cur, sqlSelect):
 cur.execute(sqlSelect)

def parseTipusCamp(val,tipus):
 valor="";
 if(val.strip()==''):
  valor="null"
 else:
  valor=val
  if(tipus=="text"):
   valor == valor.replace("'","''")
   valor = "'" + valor + "'"

 return valor



def createInserts(listaInserts):
 index = 0
```

```
textInsertCita = "insert into cita (id_captura, id_identif, id_cn,"
"data, id_estat, subestat, t_aranya, niu_aranya, id_persona" + ") values ("
textInsertCaptura = "insert into captura (n_polls, tipus_trampa, h_activ,"
"h_capt_m, anella_m, h_capt_f, anella_f, h_desactiv, h_allibera, obs_capt"
+ ") values ("
textInsertIdentificacio = "insert into identificacio (any_identif, anella_m_id,"
"anella_f_id, id_esp, polls_mesurats, cm_niu_fresc, obs_id" + ") values ("

if listaInserts[0] == "cita":
 textInsertCita = textInsertCita + "null, null, " + parseTipusCamp(listaInserts[2][0],'num') +
 ", " + parseTipusCamp(listaInserts[2][1],'text') + ", " +
 parseTipusCamp(str(returnId(estats, listaInserts[2][2])),'num') + ", " +
 parseTipusCamp(listaInserts[2][3],'text') + ", " + parseTipusCamp(listaInserts[2][4],'num') +
 ", " + parseTipusCamp(listaInserts[2][5],'num') + ", " +
 parseTipusCamp(str(returnId(personas, listaInserts[2][6])),'num') + ");"
 print textInsertCita
 executeInsert(cur,textInsertCita)
 con.commit()

elif listaInserts[0] == "captura":
 textInsertCaptura = textInsertCaptura + "" + parseTipusCamp(listaInserts[2][2],'num') +
 ", " + parseTipusCamp(listaInserts[2][3],'text') + ", " + parseTipusCamp(listaInserts[2][4],'text') +
 ", " + parseTipusCamp(listaInserts[2][5],'text') + ", " + parseTipusCamp(listaInserts[2][6],'text') +
 ", " + parseTipusCamp(listaInserts[2][7],'text') + ", " + parseTipusCamp(listaInserts[2][8],'text') +
 ", " + parseTipusCamp(listaInserts[2][9],'text') + ", " + parseTipusCamp(listaInserts[2][10],'text') +
 ", " + parseTipusCamp(listaInserts[2][11],'text') + ");"
 print textInsertCaptura
 executeInsert(cur,textInsertCaptura)
 con.commit()
 numAuto = str(autonum(cur, "select max(id_captura) from captura"))
 numAuto = numAuto.replace('(','')
 numAuto = numAuto.replace('L,)','')
 textInsertCita = textInsertCita + parseTipusCamp(numAuto,'num') + ", null, " +
 parseTipusCamp(listaInserts[2][0],'num') + ", " + parseTipusCamp(listaInserts[2][1],'text') +
 ", null, null, null, null, " + parseTipusCamp(str(returnId(personas,listaInserts[2][12])),'num') + ");"
 print textInsertCita
 executeInsert(cur,textInsertCita)
 con.commit()

elif listaInserts[0] == "identificacio":
 textInsertIdentificacio = textInsertIdentificacio + parseTipusCamp(listaInserts[2][1],'num') +
 ", " + parseTipusCamp(listaInserts[2][2],'text') + ", " + parseTipusCamp(listaInserts[2][3],'text') +
 ", " + parseTipusCamp(str(returnId(especies, listaInserts[2][4])),'num') + ", " +
 parseTipusCamp(listaInserts[2][5],'text') + ", " + parseTipusCamp(listaInserts[2][6],'num') + ", " +
 parseTipusCamp(listaInserts[2][7],'text') + ");"
 print textInsertIdentificacio
 executeInsert(cur,textInsertIdentificacio)
 con.commit()
 numAuto = str(autonum(cur, "select max(id_identif) from identificacio"))
 numAuto = numAuto.replace('(','')
 numAuto = numAuto.replace('L,)','')
 textInsertCita = textInsertCita + "null, " + parseTipusCamp(numAuto,'num') + ", " +
 parseTipusCamp(listaInserts[2][0],'num') + ", null, null, null, null, null, " +
 parseTipusCamp(str(returnId(personas,listaInserts[2][8])),'num') + ");"
 print textInsertCita
 executeInsert(cur, textInsertCita)
 con.commit()

dalecanya()
```

```
def rename():
 dire= cc.ruta_import_sig.replace('recollida_CanCata.xml','')
 os.chdir(dire)
 archivos=glob.glob('recollida_CanCata.xml')
 fecha=time.strftime("%d-%m-%Y")
 for dirlist in archivos:
  os.rename(dirlist,fecha+"-"+dirlist)

rename()
```

7.3. Programació de l'aplicació de transmissió de dades: del SIG al dispositiu mòbil

```
import os
from xml.etree.ElementTree import Element, SubElement, tostring
import psycopg2
import sys
import configcancata as cc


con = None
try:
    con = psycopg2.connect(host=cc.localhost, database=cc.db,
    user=cc.usuari, password=cc.pword)
    cur = con.cursor()


except psycopg2.DatabaseError, e:
    if con:
        con.rollback()

    print 'Error %s' % e
    sys.exit(1)

campos = "CN.id_cn, CN.lat, CN.lon, CITA.data, ESTATS_CN.estat,"
"CITA.subestat, CITA.t_aranya, CITA.niu_aranya, eclosio.ous_gen,"
"eclosio.pr_ou_gen, eclosio.ecl_estim, eclosio.ecl_real,"
"eclosio.sortida_estim, eclosio.sortida_real, eclosio.anellar,"
"eclosio.neteja, eclosio.ous_buits, identificacio.anella_m_id,"
"identificacio.anella_f_id, especie.nom_esp_cien, experiment.nom_exp,"
"experiment.data_exp "
tags = "id_cn, lat, lon, data, estat, subestat, t_aranya, niu_aranya,"
"ous_gen, pr_ou_gen, ecl_estim, ecl_real, sortida_estim, sortida_real,"
"anellar, neteja, ous_buits, anella_m_id, anella_f_id, nom_esp_cien,"
"nom_exp, data_exp "
listCampos = tags.split(", ")

sqlString = "select " + campos + " FROM CITA right join CN ON CITA.ID_CN = CN.ID_CN"
"LEFT join ESTATS_CN ON CITA.ID_ESTAT=ESTATS_CN.ID_ESTAT left join eclosio on"
"cita.id_ecl = eclosio.id_ecl left join identificacio on cita.id_identif = identificacio.id_identif"
"left join especie on identificacio.id_esp = especie.id_esp left join experiment on"
"cita.id_exp = experiment.id_exp WHERE (cita.data IN ( SELECT DISTINCT cita.data"
"FROM cita WHERE cita.data IS NOT NULL ORDER BY cita.data DESC LIMIT 2))ORDER BY cn.id_cn, cita.data"
cur.execute(sqlString)
rows = cur.fetchall()

dataroot = Element('dataroot')
```

```
for i in range(len(rows)):
    cita = SubElement(dataroot, "cita")
    for l in range(len(listCampos)):
        var = SubElement(cita, listCampos[l])
        var.text = str(rows[i][l]).strip()


f = open(ruta_export_sig, 'w')
f.write(tostring(dataroot))
f.close()

print tostring(dataroot)
```