**DOCUMENTATIE PROIECTAREA BAZELOR DE DATE**

**Candidat: Adrian, Turcus**

**Coordonator științific: Ș.l.dr.ing. Mădălin-Dorin Pop**

**INTRODUCERE**

Acest proiect reprezintă o bază de date care gestionează traficul vamal în contextul diferitelor probleme legate de granițele dintre diferite țări. Granițele în general, de-alungul istoriei au reprezentat o mare problemă atât pe plan intern dar și pe plan extern pentru respectivele țări care o împărtășesc.

Țările în trecut și rareori prezent obișnuiau să dispute diferite granițe în funcție de interesele politice/diplomatice dar și de statut sau diferite resurse importante în respectiva zonă din proximitatea graniței. În prezent, lucrurile stau diferit, iar problemele sunt de altă natură, axate mai mult pe individ decât pe o țară: imigrare ilegală, persoane dispărute în zona graniței sau a țării precum, alte inconveniente de ocazie (rătăcirea diferitelor obiecte de către trecători, mașini/bunuri furate, etc).

Această lucrare are mai multe scopuri, unul dintre ele fiind aprofundarea cunoștiințelor deja existente dar și dobândirea de noi cunoștințe personale în domeniul bazelor de date și proiectării acestora. Domeniul acesta este unul care ar trebui atins de fiecare inginer software și nu numai pentru că datele ne înconjoară peste tot unde am fi: la locul de muncă, la școală/universitate, când mergem să rezolvăm diferite probleme cu instituțiile statului, la magazin . În toate acele exemple și nu numai se află date care trebuie să fie menținute într-o anumită formă, adică în niște baze de date care trebuie proiectate de cineva. Astfel, fie că pe viitor vom lucra în domeniul specific de proiectant al bazelor de date, în creearea de site-uri web/aplicații care să fie legate la baze de date sau big data consider că este un lucru util de știut.

Al doilea principal obiectiv care îl are această lucrare asupra mea este de obișnuire în redactarea unei documentații foarte bune ca a unei lucrări de licență. Prin realizarea acestui proiect dar mai ales al documentației, ne pregătește pentru momentul în care trebuie să redactăm documentația pentru lucrarea de licență, cu toate standardele, restricțiile aferente.

Documentația acestui proiect este structurată în 8 părți: 1 – Introducere, 2 - Analiza stadiului actual în domeniul problemei unde voi discuta pe scurt despre cercetările actuale în domeniul proiectării bazelor de date dar și soluții aproximative cu a mea bine cunoscute; 3 - Bazele teoretice unde povestesc conceptele teoretice applicate; 4 - Soluţia propusă şi metodologia de proiectare/dezvoltare unde explic de ce am ales respective implementare; 5 – Implementare , în care povestesc propriu zis despre soluție cu referire la SQL, Schemă logică, Normalizări, etc; 6 - Utilizare, rezultate experimentale unde voi arăta diferite rezultate în urma implementării; 7 - Concluzii şi direcţii de dezvoltare aici are loc deznodământul legat de proiect și 8 – Bibliografie unde includ resursele bibliografice folosite în citări și nu numai.

**Analiza stadiului actual în domeniul problemei**

În ultimii ani s-au făcut diferite cercetări legate de proiectării bazelor de date relaționale cu diferite subiecte principale: Optimizări în căutarea anumitor date, folosirea bazelor pentru diverse programe și aplicații, conversia către soluții de tip non-relațional și vice versa, etc. În următoarele rânduri voi rezuma despre principalele cercetări.

Într-un articol din 2021 s-a descoperit că se poate crește eficiența bazelor de date relaționale folosind diferite tehnici ale formei a doua normale. Prin folosirea acestei forme se poate reduce redundanța, îmbunătăți integritatea datelor sau optimizarea performanței interogărilor, lucru foarte util pentru sistemele în care viteza este crucială.

Altă tehnică de îmbunătățire a performanței este prin tehnica inversării indexului prin care se pot gestiona mult mai ușor fișiere mari alături de un join optimization asupra aceste structuri. Printr-un multi-way join se pot obține rezultatele listelor de postări (posting list) după cuvinte cheie ce urmează să fie căutate într-un singur proces. De exemplu, dacă un utilizator dorește să caute documente cu patru cuvinte cheie, prin metoda existentă de two-way join , rezultatul ar fi fost dat după trei procese după traversarea listei de postări corespunzătoare celor patru cuvinte cheie. În contrast, prin folosirea unui multi-way join acest lucru se poate îndeplini printr-un singur proces, deoarece se traversează listele de postări simultan aferente celor patru cuvinte.

O altă lucrare științifică analizează cât de utile și eficiente sunt mesajele de eroare din sisteme cu baze de date relaționale. S-au luat în considerare 4 sisteme de gestiune ale bazelor de date (DBMS): MySQL, Oracle Database, PostgreSQL, SQL Server folosind testul Kruskal – Wallis H și 152 de participanți. Concluziile au fost: eficacitatea mesajelor de eroare în cadrul PostgreSQL și SQL Server a fost mai bună decât cea a MySQL dar nu s-au observat diferențe în cadrul Oracle Databases. Din punct de vedere al utilității percepute a mesajelor de eroare cu privire la rezolvarea problemelor de query, mesajele de eroare din Oracle Databases au fost cele mai puțin utile. Mesajele din PostgreSQL si SQL Server au fost considerate cele mai de ajutor în rezolvarea erorilor. Legat de error recovery confidence nu s-au văzut diferențe semnificative.

Într-o lucrare publicată anul acesta se face o paralelă între timpul de acces la date de către algoritmi complecși, o dată prin baze de date relaționale și alta prin baze non relaționale. Ceea ce a reieșit din articol este că bazele non relaționale, în acest caz exemplul dat fiind Neo4j, se descurcă mai bine la simplificarea modelării datelor și evitarea scrierii de cod. În schimb, se aruncă multe erori OutOfMemory și necesită resurse multe. La bazele de date relaționale precum Microsoft SQL Server se dovedește a fi rezilientă și de încredere, gestionând mai multe cereri simultan foarte bine.

Alte articole abordează folosirea bazelor de date în diverse aplicații precum GIS sau WebGIS, în care pe de o parte se concluzeonează că ar trebui de la început regândit punctul de vedere al designer-ului de la un user view la un conceptual view pentru pe de o parte a putea fi folosită baza de date la mai multe aplicații dar a și extinde aria de acoperire categorizări și teritoriul ales.

Făcând referire la baze de date existente similare cu cea propusă în cadrul acestui proiect, există mai multe exemple relevante. Un prim exemplu este cel de la “Bureau of Transportation Statistics (BTS)” de la Departamentul de Transporturi al Statelor Unite care au baze de date legate de trecerile între SUA – Mexic și SUA – Canada la nivel portuar. Baza lor de date înregistrează coordonatele geografice, unitatea de măsură, numele portului, data sosirii în port. Al doilea exemplu e dat de baza de date FIELDS care este un parteneriat între Frontex, Interpol și Agenția Europeană a Gărzii de Coastă și Granițelor. Când un om dorește să treacă granița într-un punct de trecere, ofițerul vamal scanează documentul de identitate, fie el pașaport sau buletin iar programul acesta identifică datele de pe document și le verifică cu cele existente în baza de date. În funcție de concordanța dată, se poate spune dacă documentul este unul contrafăcut sau nu.

**Bazele teoretice**

Pentru implementarea acestei baze de date am decis să folosesc modelul relațional prin care mai multe tabele sunt legate între ele prin referințe realizate prin chei simbolice.

Primul pas a fost modelarea conceptuală prin care stabilim care sunt entitățile noastre și relațiile dintre acestea. O entitate este dată de totalitatea datelor ce pot fi modelate sub forma unor clase de obiecte concrete sau abstracte, independente și identificabile în mod unic. Acestea pot fi independente fiind de sine stătătoare sau dependente care depind de alte entități pentru a putea fi reprezentate (nu pot fi identificate în mod unic prin atribute proprii). În cadrul acestei baze de date, entitățile noastre dispun și de niște atribute. Atributele sunt modalitatea de reprezentare ale proprietăților atomice distincte ale entităților, proprietăți identificate conform necesităților rezultate din etapa de analiză a cerințelor specifice dezvoltării bazei de date. Ele la rândul lor pot fi de mai multe feluri: de identificare – prin care se facilitează distincția între instanțele unei entitățiș de descriere, care conțin caracteristici suplimentare ale instanțelor conform cu cerințele specific dezvoltării bazei de date, attribute compuse, derivate sau cu valoare multiplă.

După ce am stabilit entitățile din baza de date și atributele acestora, avem de stabilit relațiile dintre acestea. O relație este o conexiune între două sau mai multe entități de aceelași tip sau de tip diferit. Există mai multe tipuri de relații, tipuri care le-am folosit și în baza noastră de date, acestea fiind: relație de tip 1 la 1 în care o singură instanță a unei entități se asociază cu o singură instanță a altei entități, relație de tip 1 la N în care o singură instanță se asociază cu mai multe, relație de tip N la 1 la care mai multe instanțe se asociază la o singură instanță și relații de tip N la N la care mai multe instațe a unei entități sunt legate la mai multe instanțe a altei entități.

De asemenea în cadrul proiectării avem grijă să evităm diferitele probleme: relații recursive ce pot apărea, o relație recursivă înseamnă că o entitate poate apărea de mai multe ori cu roluri diferiteș capcane de conectare – cele evantai în care există prea multe relații 1 la N în care entitățile devin ambigue sau cele de întrerupere în care uităm să reprezentăm relațiile dintre diferite entități.

Realizând astfel modelul ER (Entitate – Relație) și stabilind astfel care sunt entitățile noastre și relațiile dintre ele, este timpul să trecem la modelul relațional în care aceste entități se transformă în tabele, atributele entităților devenind coloane în cadrul tabelei. Relațiile se pot reprezenta în mai multe feluri: tabele associative (M:N) sau prin chei străine într-unul din cele două tabele-entitate cu legătură către cheia primară din celălalt astfel de tabel.

Transformăm relațiile în felul următor: în relația 1 la 1 cheia străină va fi plasată în tabelul corespunzător cu mai puține înregistrări; în relația N la 1 cheia străină va fi plasată în tabelul corespunzător părții „mai mulți” (N) din cadrul relației ș în relația N la N se folosește un tabel asociativ conținând două chei străine pentru tabelele din cadrul relației.

Pasul următor este proiectarea logică în care se ia modelul conceptual se transformă în cel logic prin pașii spuși mai sus dar la care se aplică și diferite normalizări. Normalizarea este procesul de rafinare a modelului conceptual prin care se asigură eliminarea redundanțelor. Există mai multe feluri de probleme ce pot apărea, cum ar fi anomalii de actualizare, de inserare sau ștergere. Respectând unele reguli de proiectare a bazelor de date relaționale, se pot evita aceste anomalii. O parte din reguli sunt: atributele unor entități diferite nu ar trebui combinate în aceeași relație, cheile străine trebuie utilizate pentru a crea referințe între entități, relațiile trebuie proiectate în așa fel în cât n-uplurile sale să aibă un număr minim de valori NULL.

Pentru a normaliza trebuie să ne uităm la dependențele funcționale. Ele specifică în mod formal cât de bună este o proiectare relațională și sunt reprezentate de constrângeri care descriu relația dintre atribute. Un set de atribute X determină funcțional un set de atribute Y dacă valoarea lui X determină o singură valoare pentru Y (X -> Y). De asemenea, X determină pe Y dacă oricare două n-upluri au aceeași valoare pentru X, acestea trebuie să aibă aceeași valoare pentru Y.

Dependențele funcționale complete sunt dependențele care nu conțin atribute care nu sunt necesare în determinantul său. Având aceste definiții, pentru a fi siguri că baza noastră de date nu suferă de anomalii, trebuie să ținem cont și de regulile de inferență și formele normale. Formele normale sunt de mai multe tipuri din care cele mai importante: 1NF care spune că domeniul atributelor trebuie să cuprindă valori atomice, câmpurile compuse și relațiile imbricate sunt interzise, iar fiecare atribut din n-uplet trebuie să aibă o singură valoare în domeniu; 2NF cere ca relația să fie în 1NF și că trebuie să existe o dependență funcțională completă față de cheia primară; 3NF spune ca relația să fie în 2NF iar toate atributele non-chei ale unei relații depind numai de chei candidate ale acelei relații și toate atributele non-chei trebuie să fie mutual independente. Pe lângă primele trei forme normale menționate mai sunt și BCNF la care pentru orice dependență X -> Y din R, X este o cheie candidat al lui R și orice câmp dintr-o relație depinde de cheia primară, de întreaga cheie primară și numai de ea (este varianta mai restrictivă a formei 3NF) și 4NF elimină anomaliile datorate dependențelor funcționale multivaloare. Deci o relație este în 4NF dacă este în BCNF și nu are dependențe multivaloare.

Realizând astfel modelarea conceptuală, apoi modelul relațional și normalizările respective, putem să descriem diferitele atribute și modul de acces la date, astfel îndeplinind proiectarea fizică.

Pentru a putea insera diferite date în cadrul proiectului avem nevoie de un limbaj care să ne permită asta. Pentru acest lucru vom folosi Oracle SQL, care oferă mai multe categorii de instrucțiuni și anume: DDL – Data Definition Language, DML – Data Manipulation Language, Controlul tranzacțiilor, al sesiunii și sistemului, instrucțiuni SQL încorporate. Instrucțiunile de tip DDL ajută la crearea, modificarea, eliminarea obiectelor din schemă, acordarea revocarea de privilegii, analizarea informațiilor dintr-un tabel, index sau cluster, etc. În general vom folosi comenzile CREATE, ALTER, DROP DATABASE/TABLE/VIEW/INDEX/TRIGGER. Schema este o colecție de obiecte specifice unei baze de date (tabele, vederi, etc). Ea are mai multe caracteristici printre care facilitarea partajarii de obiecte ale bazei de date între mai mulți utilizatori, un obiect trebuie să fie creat într-o singură schemă. Vederile sunt o reprezentare logică sub formă tabelară obținută ca rezultat al interogării a unei sau mai multor tabele/vederi. Au mai multe caracteristici utile printre care facilitarea utilizării de interogări complexe, a transparenței datelor din tabele relaționat, independența datelor, restricționează accesul la date prin ascunderea unor coloane din tabelele de bază precum și faptul că operațiile efectuate asupra unei vederi se vor regăsi și în tabelele de bază. Pot fi vederi simple – furnizează date dintr-o singură tabelă și nu conțin funcții sau grupuri de date sau complexe – furnizează date din mai multe tabele și conțin funcții și grupuri de date.

Pentru a avea baza de date cât mai integră posibil, trebuie să aplicăm niște constrângeri datelor. Aceste constrângeri au rolul de a preveni introducerea datelor invalide, astfel asigurând acuratețea datelor. Ele pot fi realizate fie la crearea tabelelor prin instrucțiunea “CREATE TABLE” sau ulterior prin „ALTER TABLE”. Ele pot fi aplicabile unor anumite atribute (coloane) ale unei tabele sau aplicabile întregii tabele.

Din cadrul instrucțiunilor DML ne vom folosi de INSERT, UPDATE, DELETE, SELECT și sub interogări. Cu ajutorul instrucțiunii INSERT vom adaugă date, cu UPDATE le vom modifica iar prin DELETE le ștergem. Cu ajutorul sub interogărilor putem să vizualizăm datele în diferite forme (ascendent/descendent), grupate după diferite criterii, sau prin diferite metode de selecție.

Putem de asemenea să vedem datele comune din două tabele (intersecția) sau diferențele între ele (diferența) folosind JOIN-uri. INNER se folosește pentru prima situație iar LEFT/RIGHT pentru cea de-a doua.

PL/SQL este o extensie a limbajului SQL care permite pe lângă proprietățile limbajului SQL declararea de variabile și constante, controlul fluxului, declararea de proceduri și funcții și multe altele. Pentru a putea scrie proceduri, funcții și altele de care avem nevoie, trebuie să ținem cont de mai multe lucruri: de structura unui bloc anonim, de tipuri de date în PL/SQL, structuri de decizie și iterare. Blocurile anonime în PL/SQL au un DECLARE care poate fi opțional, BEGIN care e obligatoriu – sub acesta fiind comenzi SQL și instrucțiuni PL/SQL; EXCEPTION, opțional pentru acțiuni executate în caz de ridicare de excepții și END care este obligatoriu.

Există mai multe tipuri de date în PL/SQL și anume: scalare (NUMBER, CHARACTER, DATE, BOOLEAN), compuse (înregistrări), referință (de exemplu REF CURSOR) sau obiecte mari – sunt niște indicatori către obiecte mari stocate separate de alte date (imagini grafice, text, clipuri video)

Structurile de decizie presupun specificarea uneia sau mai multor condiții unor instrucțiuni care urmează să fie executate dacă evaluarea condiției are valoarea adevărat. Opțional se pot defini instrucțiuni care se execută în cazul valorii de fals a condiției. În PL/SQL sunt disponibile: IF-THEN, IF-THEN-ELSE (de asemenea și imbricate), IF-THEN-ELSIF, CASE, CASE (searched).

Structurile de iterare presupun execuția de mai multe ori a instrucțiunilor definite într-un bloc de cod. În PL/SQL sunt disponibile LOOP, WHILE, FOR sau imbricări între acestea. În ajutorul acestor structuri avem și cele de control ale iterării: EXIT (se situează în buclă și execuția se termină imediat, controlul programului se reia cu prima instrucțiune de după buclă), EXIT WHEN (se iese dacă condiția se evaluează ca true), CONTINUE (forțează ca următoarea iterație să aibă loc, actuala oprindu-se la întâlnirea cuvântului CONTINUE) și GOTO (oferă un salt necondiționat la o instrucțiune etichetată din același subprogram).

Subprogramele pot fi de două tipuri – proceduri care se utilizează pentru a efectua o acțiune, funcții care se utilizează pentru a calcula și returna o valoare; și pot fi locale – definite în cadrul altui bloc PL/SQL sau subprogram sau stocate - create folosind comanda CREATE. Asemeni blocurilor anonime au trei părți: una declarativă care are declarații de variabile, tipuri, constante; una executabilă – obligatorie și care conține instrucțiunile care efectuează acțiunea dorită; tratarea excepțiilor – care conține codul gestionării erorilor de rulare. Procedurile stocate devin obiecte în schema utilizatorului care a creat-o putând fi apelate în mod explicit de către un client SQL sau dintr-o altă secvență de cod. Funcțiile stocate sunt utilizate pentru a calcula și returna o valoare iar corpul acestora trebuie să conțină cel puțin o comandă RETURN a unei date având tipul specificat).

**Soluția propusă și metodologia de proiectare/dezvoltare**

1. Proiectarea Conceptuală

Pentru acest proiect am implementat o bază de date care gestionează traficul unui punct vamal între două țări. O bază de date a unui punct vamal ar trebui să fie bine implementată deoarece pot apărea diferite probleme de la imigrare ilegală, persoane dispărute în zona graniței sau a țării, precum și alte inconveniente de ocazie (rătăcirea bunurilor, mașina familiei furată, etc) iar această soluție ar putea rezolva câteva din situațiile listate mai sus.

A diagram of a flowchart

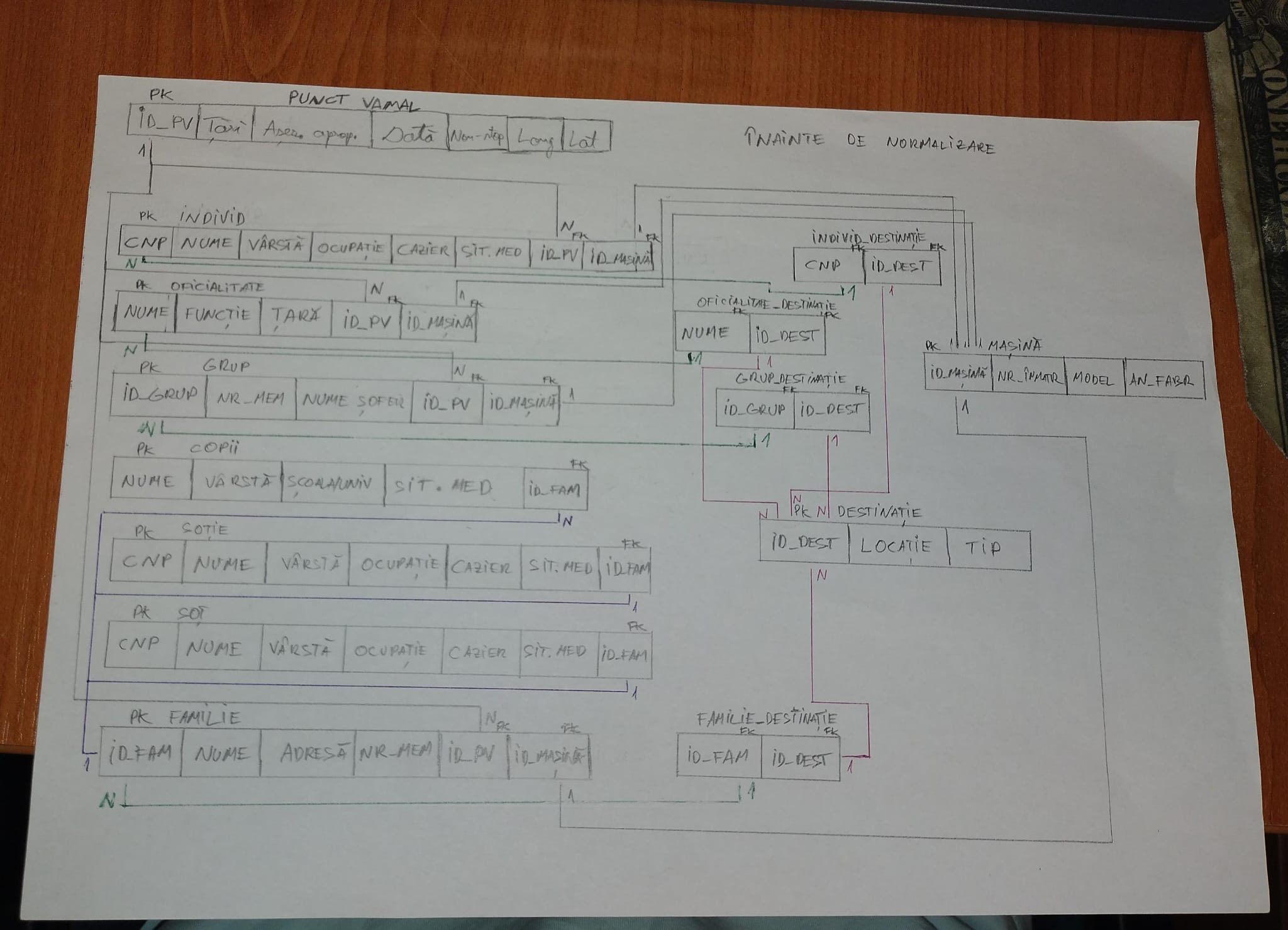
Description automatically generated

În baza mea de date există mai multe entități precum **Punct Vamal** care este principalul punct de plecare prin care pot trece entități **precum Individ, Oficialitate, Grup, Familie**. Între Punct Vamal și aceste entități există o relație de tip 1 la N deoarece prin punctul nostru vamal pot trece mai multe entități în diferite momente ale zilei. De asemenea aceste entități menționate mai devreme sunt conectate la câte o entitate **Mașină** în relație 1 la 1. Pentru a facilita cazurile punctului vamal în care trec familiile am legat la aceasta entitățile **Soț, Soție** în relație 1 la 1 cu entitatea **Familie,** și **Copii** în relație 1 la N cu Familia, aceasta putând avea 1 sau mai mulți copii.

Felul în care este realizată diagrama entitate – relație ne ajută în evitarea capcanelor în evantai (fan traps) deoarece avem o viziune clară asupra a ce entități pot trece punctul vamal acestea fiind distincte și nedorind a generaliza entitățile. De exemplu pentru o oficialitate nu ar trebui ca la punctul vamal să întrebăm despre vârstă, cnp sau cazier pentru a putea scurta timpul de trecere, ci doar despre nume, funcție și țara de origine. Asemenea și la familie, dorim să se vadă clar că au o legătură prin nume, să nu fie doar un grup de oameni. Capcaneele de întrerupere (chasm traps) sunt evitate deoarece reușim să reprezentăm fiecare legătură între entități cu succes.

1. Proiectarea Logică

În continuare dorim să obținem modelul logic din cel conceptual, vom lua fiecare entitate și le vom reprezenta sub o formă din care pot fi văzute mai ușor relațiile între entități.



Și obținem respectiva diagramă, în care sunt reprezentate relațiile de tip 1 la 1, 1 la N și N la N între diferitele entități cât și cheile primare și străine care sunt trecute cu PK (Primary Key) și FK (Foreign Key) deasupra atributului entității.

Dar această reprezentare are ca una din probleme , nefiind normalizată, existența a două entități diferite cu fix aceleași câmpuri alese și fix aceeași cheie primară atât pentru **Soț** cât și pentru **Soție**, lucru care doar complică baza de date. Pentru a putea rezolva această problemă apelăm la normalizare, creând doar o singură entitate **Părinte** la care mai putem adăuga un atribut “SEX”.

Cealaltă problemă mai puțin evidentă este redundanța dată de tabelul cu **Destinație** fiind adăugată de mai multe ori nefiind necesar. Ca exemplu se inserează de mai multe ori aceeași locație dar călătoria are alt scop (dată de atributul “TIP”) : 1 București vacanță ; 2 București afaceri ; 3 București vizită medicală. Acest lucru s-ar putea rezolva folosind încă o tabelă **Detalii\_Destinație** care ar putea ține ca atribut tipul de călătorie pentru a nu mai duplica locația. Separând astfel tabelele, și creând o conexiune între ele prin „ID\_DEST” , ajungem să eliminăm problema, având deci a patra formă normală. Este a patra formă pentru că tipul destinației (“TIP”) reprezintă o dependență multivaloare care rezulta în redundanță și posibile anomalii, eliminarea acesteia fiind esențială.