1. Descrierea Temei

VulnSight este o aplicație software integrată, concepută pentru gestionarea centralizată și remedierea proactivă a vulnerabilităților de securitate (CVE) dintr-o infrastructură IT enterprise. Proiectul simulează mediul unei organizații mari, cu departamente multiple (ex: IT, HR, Finance) și echipe de securitate specializate (ex: AppSec, Infrastructure, Network, Compliance), având ca scop reducerea suprafeței de atac prin prioritizarea inteligentă a riscurilor. Într-un peisaj cibernetic în care volumul de vulnerabilități detectate depășește capacitatea umană de procesare, VulnSight automatizează fluxul de lucru dintre detectare (scanare) și remediere, oferind o vizibilitate clară asupra posturii de securitate.

Scopul principal al sistemului este transformarea datelor brute provenite din scanerele de vulnerabilități în informații acționabile, prin: **prioritizarea riscului** (calcularea unui scor de risc strategic per departament, ținând cont nu doar de severitatea tehnică, ci și de importanța de business a activelor și bugetul departamentului), **monitorizarea performanței (SLA)** (urmărirea eficienței echipelor de securitate prin metrici precum MTTR (Mean Time To Remediate) și respectarea termenelor limită (SLA Breaches)), **și nu în ultimul rand managementul fluxului de lucru,** prin gestionarea cicluilui de viață al tichetelor de remediere, de la creare până la validarea fix-ului.

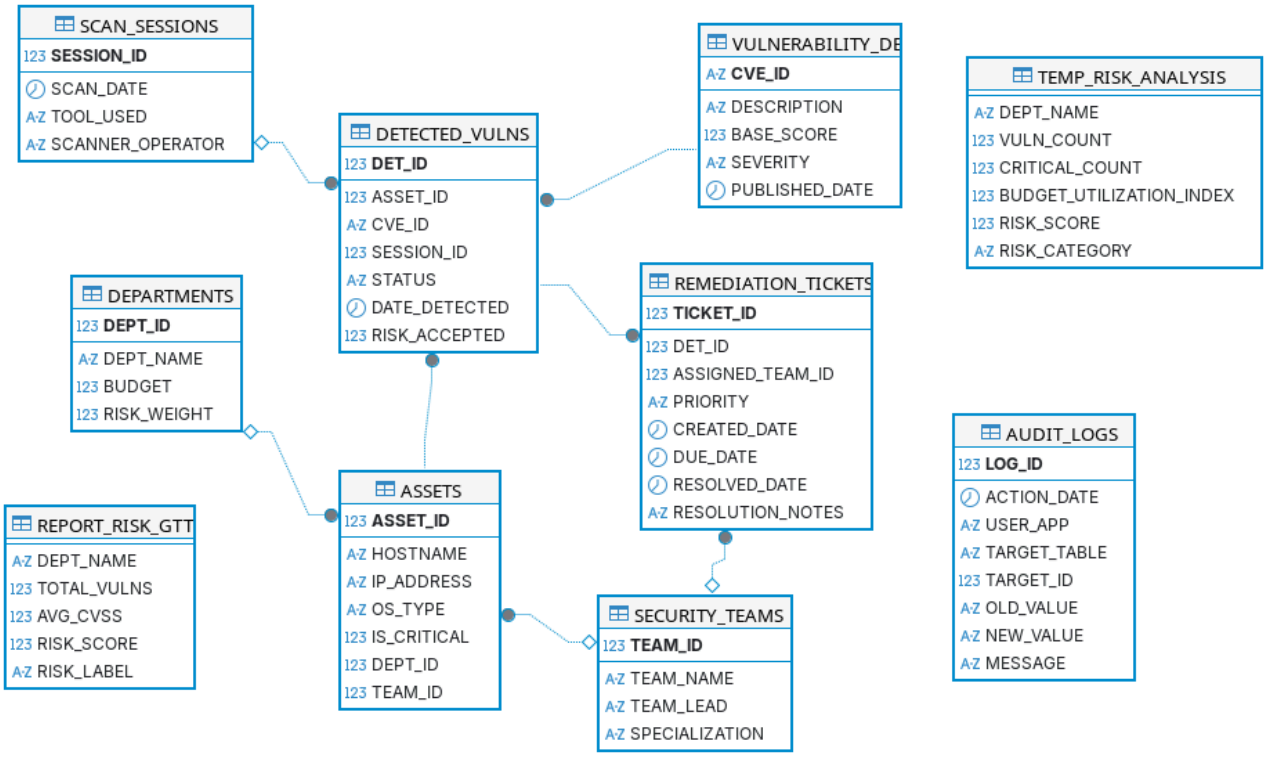
2. Descrierea Bazei de date

Arhitectura bazei de date VulnSight a fost concepută pentru a acomoda fluxul complet de management al vulnerabilităților într-o organizație, pornind de la structura administrativă și ajungând până la remedierea tehnică și auditarea acțiunilor. Fundamentul acestui sistem este reprezentat de stratul organizațional, definit prin tabela **DEPARTMENTS**, care funcționează ca un nomenclator central pentru diviziunile companiei. Aceasta nu stochează doar denumiri, ci și parametri financiari critici, precum bugetul și o pondere de risc specifică, elemente esențiale pentru algoritmii ulteriori de calcul al expunerii financiare. În completarea structurii organizaționale, tabela **SECURITY_TEAMS** gestionează resursa umană specializată, definind echipele de intervenție care vor fi responsabile pentru soluționarea incidentelor. Punctul de convergență între structura administrativă și cea tehnică este realizat prin tabela **ASSETS**. Aceasta reprezintă inventarul activelor IT (servere, stații de lucru) și acționează ca un nod central în diagramă, deoarece fiecare activ este, pe de o parte, proprietatea unui departament (legătură către **DEPARTMENTS**) și, pe de altă parte, intră în aria de responsabilitate a unei echipe tehnice (legătură către **SECURITY_TEAMS**). Astfel, sistemul permite o trasabilitate clară a cui este echipamentul și cine trebuie să îl repare în caz de probleme.

Pentru a evalua securitatea, baza de date integrează un catalog standardizat de amenințări prin tabela **VULNERABILITY_DB**. Aceasta stochează definițiile universale ale vulnerabilităților (CVE-uri), inclusiv descrierile tehnice și scorurile de severitate (CVSS), servind drept referință pentru evaluarea riscului. Dinamica sistemului este capturată prin tabela **SCAN_SESSIONS**, care înregistrează momentele temporale în care infrastructura este scanată și instrumentele utilizate. Rezultatul intersecției dintre activele scanate și catalogul de vulnerabilități este materializat în tabela **DETECTED_VULNS**. Aceasta este cea mai voluminoasă și dinamică tabelă din sistem, reprezentând o relație many-to-many care stochează instanțele concrete ale vulnerabilităților pe fiecare server în parte, reținând starea acestora (Open, Fixed) și data detectării.

Componenta operațională, care transformă o simplă alertă într-o sarcină de lucru, este gestionată de tabela **REMEDIATION_TICKETS**. Aceasta preia vulnerabilitățile detectate și le transformă în procese de remediere asignate echipelor de securitate, monitorizând termenele de rezolvare (Due Date) și prioritățile. În final, pentru a asigura guvernanța și integritatea datelor, sistemul include tabela **AUDIT_LOGS**. Deși funcționează independent de fluxul operațional direct, această tabelă este esențială pentru securitatea aplicației însăși, fiind populată automat prin mecanisme de tip trigger care jurnalizează orice modificare critică asupra tichetelor sau stărilor vulnerabilităților, oferind o pistă de audit completă asupra acțiunilor utilizatorilor.

2.1 Diagrama Baza de date



2.2 Structura Tabelelor

Baza de date este organizată pe un model relațional robust, compus din 10 structuri tabelare distincte (9 tabele permanente și un tabel temporar global), clasificate logic în nomenclatoare, tabele operaționale și structuri de suport/audit.

1. Tabela DEPARTMENTS este fundamentul organizatoric al bazei de date și servește drept nomenclator central pentru structura ierarhică a companiei. Rolul acesteia depășește simpla listare a numelor de departamente, fiind proiectată special pentru a susține analize de risc financiar. Identificarea fiecărei entități se face printr-o cheie primară generată automat (**Dept_ID**), însă elementele distinctive sunt coloanele de analiză. Coloana **Budget**, definită cu precizie financiară (**NUMBER(12, 2)**), permite stocarea unor sume considerabile cu două zecimale, esențiale pentru calcularea impactului economic al potențialelor breșe de securitate. Mai mult, pentru a reflecta realitatea că nu toate departamentele au aceeași importanță critică, a fost introdusă coloana **Risk_Weight**. Aceasta stochează un factor de multiplicare (implicit 1.0, dar variabil, de exemplu 1.5 pentru IT sau 0.8 pentru HR), care va fi utilizat ulterior în algoritmii complecși de calcul al riscului, permițând prioritizarea incidentelor în funcție de sensibilitatea departamentului afectat.

2. Tabela SECURITY_TEAMS realizeaza gestionarea resursei umane specializate. Această tabelă impune reguli stricte de integritate pentru a asigura un flux de lucru coerent. Identitatea fiecărei echipe este protejată printr-o constrângere de unicitate (**UNIQUE**) aplicată numelui, eliminând riscul de a avea duplicate care ar putea crea confuzie în asignarea tichetelor. Cel mai important aspect tehnic al acestei tabele este validarea specializării prin constrângerea de tip **CHECK** aplicată coloanei **Specialization**. Aceasta restricționează valorile posibile la o listă predefinită strictă: 'Infrastructure', 'AppSec', 'Compliance' și 'Network'. Această validare la nivel de bază de date garantează că nicio echipă nu poate fi creată cu o specializare necunoscută sistemului, asigurând astfel că mecanismele automate de distribuire a tichetelor vor funcționa întotdeauna corect, direcționând problemele de rețea către experții de rețea și problemele de cod către experții de securitate a aplicațiilor.

3. Tabela VULNERABILITY_DB este un dicționar al definițiilor de securitate. Spre deosebire de celelalte tabele, cheia primară aici nu este un număr generat, ci codul standard internațional al vulnerabilității (**CVE_ID**, de tip VARCHAR2), asigurând interoperabilitatea cu standardele externe de securitate. Integritatea datelor este critică în acest punct, motiv pentru care tabela implementează validări matematice și logice riguroase. Scorul de bază al vulnerabilității (**Base_Score**) este forțat printr-o constrângere **CHECK** să se încadreze în intervalul valid 0-10, conform standardului CVSS. Simultan, clasificarea gravității (**Severity**) este standardizată prin restricționarea valorilor la setul: 'Low', 'Medium', 'High' și 'Critical'.

4. Tabela SCAN_SESSIONS funcționează ca un registru de intrări pentru datele brute. Aceasta nu doar stochează rezultatele, ci oferă contextul operațional în care au fost obținute. Importanța ei rezidă în coloana

Scan_Date (tip **TIMESTAMP**), care capturează cu precizie momentul exact al rulării procesului de detecție. Mai mult, deoarece organizațiile moderne folosesc ecosisteme de securitate, coloana **Tool_Used** permite diferențierea surselor de date (de exemplu, scanări venite din Nessus versus Qualys), în timp ce câmpul **Scanner_Operator** identifică responsabilul care a inițiat procesul, adăugând un nivel suplimentar de responsabilitate asupra datelor introduse în sistem.

5. Tabela APP_USERS implementează mecanismele de autentificare și autorizare. Proiectarea acesteia respectă cele mai bune practici de securitate: parolele nu sunt stocate niciodată în clar, ci sub formă criptată în coloana **Password_Hash**, protejând sistemul în cazul unui leak de date. Din punct de vedere logic, tabela impune o structură ierarhică de acces (RBAC - Role Based Access Control) prin constrângerea de tip **CHECK** aplicată coloanei **Role**. Aceasta limitează strict tipurile de utilizatori la 'Admin', 'CISO', 'Viewer' sau 'Auditor', garantând că un simplu auditor nu poate avea drepturi de modificare asupra configurațiilor critice.

6. Tabela ASSETS reprezintă coloana vertebrală a bazei de date, acționând ca punctul de legătură între lumea fizică (echipamente) și cea administrativă. Aceasta este o tabelă de fapte care integrează referințe către departamente și echipe (**Dept_ID** și **Team_ID**), răspunzând la întrebarea "Cine deține acest server și cine se ocupa de el?". Structura este pregătită pentru infrastructuri moderne, coloana **IP_Address** fiind dimensionată pentru a suporta standardele IPv6. Un element cheie de business este coloana **Is_Critical**, implementată ca un flag numeric boolean și validată prin constrângeri. Acest atribut permite sistemului să prioritizeze automat activele vitale în rapoartele de risc, separând serverele de producție critice de stațiile de lucru obișnuite.

7. Tabela DETECTED_VULNS este cea mai complexă structură din punct de vedere relațional, care materializează legătura de tip Many-to-Many dintre activele infrastructurii și catalogul de vulnerabilități. Aceasta nu este doar o simplă tabelă de asocieri, ci conține logica de stare a întregii aplicații. Coloana **Status** controlează fluxul de viață al unei probleme, permițând tranziții validate între stările 'Open', 'Fixed', 'False Positive' și 'Risk Accepted'. Integritatea datelor este asigurată printr-o constrângere compusă de unicitate (**UQ_Asset_Vuln**), care previne duplicarea înregistrărilor: sistemul garantează că aceeași vulnerabilitate, pe același server, în cadrul aceleiași sesiuni de scanare, nu poate fi înregistrată de două ori.

8. Tabela REMEDIATION_TICKETS reprezintă motorul operațional al aplicației, fiind locul unde o vulnerabilitate pasivă se transformă într-o sarcină de lucru activă. Aceasta leagă logic o detecție tehnică (**Det_ID**) de o echipă umană responsabilă (**Assigned_Team_ID**), facilitând managementul incidentelor. Structura este guvernată de reguli de business stricte implementate direct la nivel de schemă: coloana **Priority** forțează clasificarea tichetelor într-o ierarhie standardizată (P1 - Critic până la P4 - Low), esențială pentru respectarea termenelor. Un aspect tehnic definitoriu este constrângerea logică **CHK_Dates**, care validează cronologia evenimentelor, interzicând imposibilitatea fizică ca data rezolvării (**Resolved_Date**) să fie anterioară datei creării tichetului. De asemenea, clauza **ON DELETE CASCADE** moștenită de la vulnerabilitatea părinte asigură curățenia bazei de date: dacă o vulnerabilitate este ștearsă din sistem, toate tichete asociate dispar automat, prevenind consistența orfană.

9. Tabela AUDIT_LOGS este o structură optimizată pentru scriere rapidă, care capturează amprenta digitală a modificărilor critice. Fiecare înregistrare păstrează un instantaneu complet al evenimentului: momentul exact (**Action_Date** cu precizie de **TIMESTAMP**), autorul modificării (**User_App**) și, cel mai important, starea datelor înainte (**Old_Value**) și după modificare (**New_Value**).

10. Tabela TEMP_RISK_ANALYSIS este, spre deosebire de restul tabelelor care stochează date persistente, o structură specială de tip **Global Temporary Table (GTT)**, concepută exclusiv pentru optimizarea performanței rapoartelor complexe. Aceasta servește drept zonă tampon pentru procedura stocată **CALCULATE_DEPT_RISK**. Deoarece calculele de risc implică agregări costisitoare (sume condiționale, logaritmi de buget, ponderi de risc), sistemul populează temporar această tabelă cu rezultatele intermediare. Clauza **ON COMMIT PRESERVE ROWS** este critică aici: ea permite datelor să supraviețuiască sfârșitului tranzacției de inserare suficient timp pentru a fi citite de cursorul aplicației, după care sunt curățate automat la închiderea sesiuni. Această abordare reduce încărcarea pe tabelele tranzacționale principale și accelerează generarea tablourilor de bord în aplicația client.

2.3 Descrierea constrângerilor de integritate

Integritatea și consistența datelor în cadrul sistemului sunt garantate printr-un ansamblu robust de reguli implementate direct la nivelul schemei bazei de date. Primul nivel de integritate este asigurat prin definirea cheilor primare **PRIMARY KEY** pentru toate cele opt tabele ale sistemului. Majoritatea tabelelor utilizează chei surogat numerice generate automat de sistem prin mecanismul `IDENTITY` (de exemplu, **Asset_ID,**

Ticket_ID), facilitând indexarea și referențierea rapidă. O excepție notabilă și justificată de logică o constituie tabela **VULNERABILITY_DB**, unde cheia primară este un cod alfanumeric (**CVE_ID**), respectând astfel standardul internațional de numire a vulnerabilităților și prevenind introducerea unor duplicate la nivel de catalog. Integritatea referențială este strict aplicată pentru a menține coerența relațiilor dintre entități. Tabelele operaționale, precum **ASSETS** și **REMEDIATION_TICKETS**, sunt legate de nomenclatoarele de departamente și echipe prin chei străine (**FOREIGN KEY**), asigurând faptul că niciun activ nu poate aparține unui departament inexistent și niciun tichet nu poate fi alocat unei echipe fantomă. Un aspect critic al modelului este gestionarea ștergerilor în cascadă: tabela **DETECTED_VULNS** include clauza **ON DELETE CASCADE** pe relația cu activele IT. Aceasta garantează că, în cazul eliminarii unui server din inventar, toate vulnerabilitățile asociate acestuia sunt eliminate automat, prevenind apariția datelor orfane și menținând baza de date curată fără intervenții manuale suplimentare. Pentru a asigura integritatea domeniului și validitatea datelor de intrare, au fost implementate numeroase constrângeri de tip **CHECK**. Acestea validează parametrii numerici critici, cum ar fi **Base_Score** din tabela de vulnerabilități, care este restricționat matematic la intervalul [0, 10], și simulează tipuri de date booleene prin limitarea coloanelor precum **Is_Critical** sau **Risk_Accepted** la valorile 0 și 1. De asemenea, consistența fluxurilor de lucru este impusă prin restricționarea valorilor textuale la liste predefinite: specializările echipelor, rolurile utilizatorilor și statusurile vulnerabilităților sunt strict validate, eliminând riscul erorilor umane de tastare sau clasificările ambigue.

2.4 Popularea tabelelor

Procesul de populare a bazei de date este complet automatizat prin intermediul unui script Python, conceput să genereze un fișier SQL cu instrucțiuni de inserare masivă. Scriptul utilizează biblioteca externă **Faker** pentru a simula date credibile (nume de persoane, descrieri de text, adrese IP), alături de bibliotecile standard **random** pentru selecții aleatorii, **hashlib** pentru securitatea parolelor și **datetime** pentru manipularea cronologică a evenimentelor.

Logica scriptului respectă cu strictețe constrângerile de integritate referențială ale bazei de date, inserând datele într-o ordine ierarhică specifică. Inițial, sunt populate nomenclatoarele și entitățile statice, precum departamentele, echipele de securitate și utilizatorii aplicației, pentru care parolele sunt stocate sub formă de hash-uri SHA256, simulând standardele reale de securitate. Ulterior, sunt generate cataloagele de vulnerabilități cu scoruri de severitate calculate dinamic și inventarul de active IT, care sunt distribuite aleatoriu către departamente.

Nivelul de complexitate crește odată cu generarea datelor tranzacționale. Scriptul creează asocieri logice între active, sesiuni de scanare și vulnerabilități pentru a popula tabelul de detecții. O particularitate importantă a algoritmului este consistența temporală: tichetele de remediere sunt generate exclusiv pentru vulnerabilitățile marcate ca "Fixed", asigurându-se programatic faptul că data rezolvării este întotdeauna ulterioară datei detectării, iar termenele limită (SLA) sunt calculate corect. Procesul se încheie cu generarea unor loguri de audit simulate, care reflectă acțiunile utilizatorilor asupra diverselor entități din sistem.

2.5 Descrierea procedurilor, funcțiilor și triggerelor

GET_CRITICAL_ASSETS este cel mai avansat raport din aplicație, fiind conceput pentru a identifica problemele critice de securitate care au fost ignorate o perioadă lungă de timp. Procedura analizează toate activele din infrastructură și selectează doar acele servere sau echipamente care au vulnerabilități nerezolvate („Open” sau „Risk Accepted”) mai vechi de o lună. Rezultatul nu este o simplă listă, ci o **agregare**: pentru fiecare server (identificat prin Hostname și IP), procedura numără câte vulnerabilități există și care este cel mai mare scor de risc prezent pe acel echipament. De asemenea, calculează zilele de expunere (cât timp a trecut de la detectare). Raportul permite filtrarea dinamică prin parametrul **p_min_score**, afișând doar problemele care depășesc un anumit prag de severitate. Acest raport satisface cu succes cerința de complexitate maximă, **cumulând 8 puncte**:

1. Trei clauze de **JOIN** (conectează tabelele **Assets, Departments, Detected_Vulns** și **Vulnerability_DB**).
2. Trei condiții de filtrare **WHERE** (statusul vulnerabilității, scorul minim și vechimea detectării).
3. O clauză de grupare **GROUP BY** (pentru a agrega datele per echipament).
4. O clauză **HAVING** (pentru a exclude echipamentele curate, fără vulnerabilități).

CALCULATE_DEPT_RISK oferă o imagine de ansamblu asupra expunerii la risc a fiecărui departament, corelând aceste date cu bugetul disponibil. Este singura procedură care utilizează procesare intermediară

prin tabele temporare. Funcționarea este împărțită în doi pași. Mai întâi, procedura șterge datele vechi și populează un tabel temporar (**TEMP_RISK_ANALYSIS**) cu date proaspete calculate. Calculul riscului este ponderat: se înmulțește suma scorurilor vulnerabilităților cu factorul de risc specific departamentului (ex. departamentul Financiar poate avea un factor mai mare decât Marketing). În al doilea pas, pe baza scorului rezultat, procedura clasifică departamentul într-o categorie de risc („EXTREME”, „HIGH”, „MODERATE”) folosind o structură decizională. Rezultatul final ajută managementul să vadă, de exemplu, **dacă un departament cu buget mic are un risc extrem de mare.** Raportul se încadrează în categoria de complexitate medie **(6 puncte)**:

1. Trei clauze de **JOIN** (pentru a lega departamentele de vulnerabilitățile active).
2. O condiție **WHERE** (filtrează doar vulnerabilitățile active - „Open”).
3. O clauză **GROUP BY** (calculează totalurile per departament).
4. O clauză **HAVING** (exclude departamentele care nu au nicio vulnerabilitate detectată).

GET_TEAM_PERFORMANCE este un instrument de evaluare a performanței pentru echipele de securitate, transformând datele brute din tichete într-un clasament al eficienței. Procedura utilizează o structură avansată de tip **Common Table Expression (CTE)** pentru a calcula statistici detaliate pentru fiecare echipă. Logica determină un scor de eficiență complex: echipa primește puncte pentru fiecare tichet rezolvat (cu bonus pentru vulnerabilități critice), dar este penalizată drastic dacă a depășit termenul limită (SLA - Service Level Agreement). În final, raportul returnează un clasament și atribuie automat o etichetă de performanță: „Elite” (fără întârzieri), „Strong” (scor mare) sau „Needs Training”. Deși îndeplinește cerința minimă, complexitatea logică reală este **5 puncte:**

1. Trei clauze de **JOIN** (conectează echipele cu tichetele, detecțiile și baza de date CVE).
2. O condiție **WHERE** (doar tichetele rezolvate).
3. O clauză **GROUP BY** (agregare pe echipe și specializări).

SP_GET_ALL_TEAMS are rolul de a popula listele de selecție din interfața de utilizator. Ea returnează lista completă a echipelor de securitate și specializarea lor, ordonate alfabetic, permițând managerului să aloce corect tichetele către grupul responsabil (ex. problemele de rețea către echipa 'NetPatrol').

SP_GET_VULNS_NO_TICKET este esențială pentru prevenirea dublării eforturilor. Ea interoghează baza de date pentru a returna doar vulnerabilitățile active („Open” sau „Risk Accepted”) care **nu au încă un tichet de remediere deschis**. În plus, generează un format de afișare prietenos (**DISPLAY_LABEL**) care concatenează numele departamentului, codul CVE și numele serverului, oferind operatorului contextul necesar direct în lista de selecție.

SP_CREATE_TICKET gestionează logica de creare a tichetelor. Înainte de inserare, efectuează o validare pentru a se asigura că nu există deja un tichet activ pentru aceeași vulnerabilitate. O funcționalitate cheie este calculul automat al termenului limită (SLA): la crearea tichetului, câmpul **Due_Date** este setat automat la 7 zile de la data curentă (**SYSDATE + 7**), standardizând timpul de răspuns pentru incidente.

SP_GET_OPEN_TICKETS alimentează "Dashboard-ul Operațional" al aplicației. Ea returnează toate tichetele nerezolvate, împreună cu detalii critice precum prioritatea, severitatea vulnerabilității și serverul afectat. Rezultatele sunt ordonate prioritar (P1 -> P4) și cronologic, permițând echipelor să se concentreze pe cele mai urgente probleme.

SP_RESOLVE_TICKET marchează închiderea ciclului de viață al unui incident. Ea actualizează tichetul cu data curentă a rezolvării și salvează notele explicative introduse de tehnician. Procedura include validări pentru a se asigura că se încearcă închiderea unui tichet valid și existent.

Pentru a menține consistența datelor între tabele și a impune reguli stricte de business, am implementat triggerul **TRG_COMPLEX_TICKET_PROCESS**, care se declanșează automat la actualizarea unui tichet. Acesta îndeplinește trei funcții vitale. Intai impune o regulă strictă pentru tichetele critice (Prioritate P1). Sistemul blochează încercarea de a închide un tichet P1 dacă tehnicianul nu a oferit o explicație detaliată (minim 10 caractere) în câmpul de note, asigurând astfel documentarea corectă a incidentelor majore. De asemenea asigură consistența între tabelul de tichete și cel de vulnerabilități. În momentul în care un tichet este marcat ca rezolvat (**Resolved**), triggerul actualizează automat statusul vulnerabilității asociate în tabelul **DETECTED_VULNS** din 'Open' în 'Fixed', eliminând necesitatea actualizării manuale în două locuri. Orice modificare de status a unui tichet este înregistrată automat în tabelul **AUDIT_LOGS**. Se salvează utilizatorul

care a făcut modificarea, vechea stare, noua stare și un mesaj descriptiv, garantând trasabilitatea completă a acțiunilor de remediere.

3. Descriere Aplicație și prezentarea modului în care se face conexiunea cu baza de date

Aplicația este construită folosind limbajul **Python** și framework-ul **Streamlit**, ales pentru capacitatea sa de a randa rapid dashboard-uri interactive de analiză a datelor. Vizualizarea grafică a metricilor de risc și performanță este realizată cu ajutorul bibliotecii **Plotly**, care transformă seturile de date brute returnate de procedurile stocate în grafice intuitive. Conexiunea între aplicația Python și baza de date Oracle este realizată direct, folosind driverul nativ **python-oracledb**. Aceasta este o componentă critică a aplicației, asigurând un canal securizat și eficient pentru execuția procedurilor stocate. În cadrul codului sursă, conexiunea este gestionată de funcția **get_db_connection()**, care utilizează decoratorul **@st.cache_resource**. Acest mecanism de caching este esențial pentru performanță, deoarece previne reconectarea la baza de date la fiecare interacțiune a utilizatorului cu interfața (refresh de pagină sau apăsare de buton), menținând o sesiune activă și stabilă.

Parametrii de conectare sunt definiți static (pentru mediul de dezvoltare local) către **localhost:1521/XE**, autentificarea realizându-se cu utilizatorul **STUDENT_BD**. Odată stabilită conexiunea, aplicația **NU execută interogări SQL directe (SELECT/INSERT) din codul Python și nu prelucrează datele folosind un ORM**. În schimb, se instanțiază cursori care apelează exclusiv pachete și proceduri stocate (exemplu: **PKG_VULNSIGHT.GET_CRITICAL_ASSETS** sau **SP_CREATE_TICKET**), delegând responsabilitatea calculelor, a filtrării și a validării datelor către motorul bazei de date.

Interfața utilizator este organizată ergonomic în patru module principale (tab-uri), fiecare adresând o nevoie specifică a fluxului de management al vulnerabilităților. **Analiza de Risc Strategic** oferă o perspectivă de ansamblu (High-Level) asupra stării de securitate. Utilizatorul poate genera rapoarte complexe care calculează scorul de risc per departament. Datele sunt procesate în baza de date prin formule ponderate (ținând cont de buget și numărul de vulnerabilități critice), iar aplicația doar afișează rezultatul sub forma unui grafic color-coded (Roșu pentru risc extrem, Galben pentru moderat). **Monitorizarea Performanței Echipelor** permite vizualizarea eficienței echipelor tehnice. Se apelează proceduri care agregează date istorice despre tichetele rezolvate, calculând metrici precum SLA Breaches (încălcări ale timpului limită) și MTTR (Mean Time To Resolve). Rezultatul este un clasament al echipelor, evidențiind cea mai eficientă echipă. **Identificarea Activelor Critice** este un modul dedicat filtrării avansate, unde utilizatorul poate ajusta din interfață pragul scorului CVSS (de exemplu, afișarea doar a vulnerabilităților cu scor > 9.0). Această interacțiune trimite parametrul direct către procedura stocată, care returnează lista activelor ce necesită atenție imediată. **Centrul de Operațiuni** este modulul tranzacțional al aplicației. Utilizatorul selectează o vulnerabilitate nealocată și o asignează unei echipe, stabilind prioritatea. Aplicația apelează **SP_CREATE_TICKET** pentru a insera datele. Utilizatorul marchează apoi un tichet ca fiind rezolvat, adăugând o notă tehnică obligatorie. Procedura **SP_RESOLVE_TICKET** este apelată pentru a actualiza starea în baza de date și a închide ciclul de viață al incidentului.

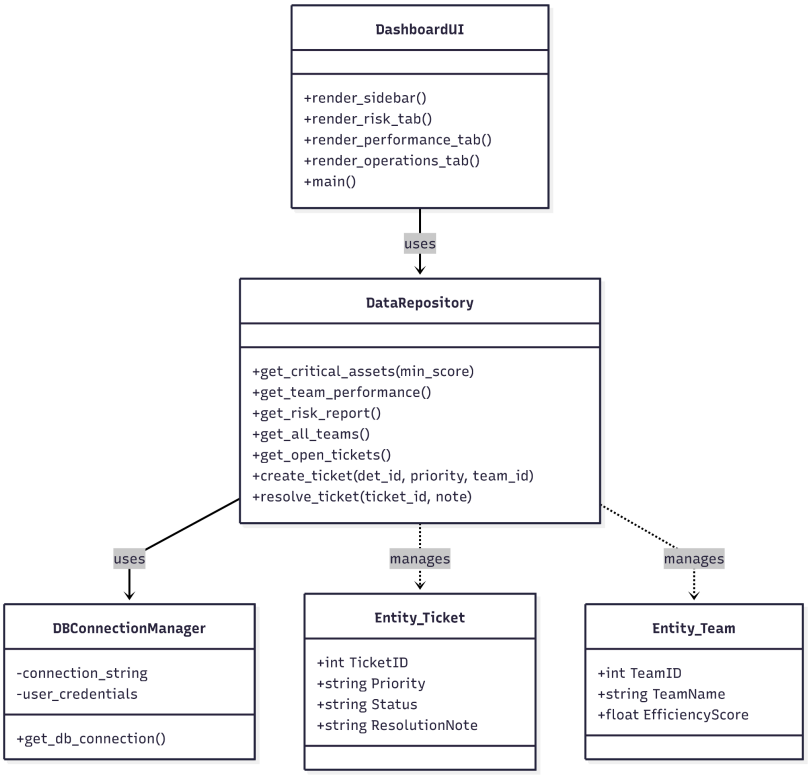
3.1 Diagrama de clase și structura claselor

Pilonul central al comunicării cu infrastructura backend este clasa **DBConnectionManager**, care acționează ca un Singleton la nivelul sesiunii aplicației. Această componentă are responsabilitatea critică de a inițializa și menține o conexiune stabilă cu serverul Oracle.

Interacțiunea efectivă cu datele este abstractizată complet prin clasa **DataRepository**, care funcționează ca un Service Layer (strat de servicii). Acest modul este intermediarul exclusiv între interfața grafică și logica de business stocată în pachetul PL/SQL **PKG_VULNSIGHT**. Codul Python din acest strat nu conține interogări SQL directe; în schimb, apelează metode specifice care execută proceduri stocate. De exemplu, metoda **get_risk_report()** preia cursorul returnat de baza de date și îl transformă într-un DataFrame Pandas pentru analiză, în timp ce metodele tranzacționale, precum **create_ticket_action** sau **resolve_ticket_action**, gestionează parametrii de intrare, validarea datelor (cum ar fi notele tehnice) și commit-ul tranzacțiilor.

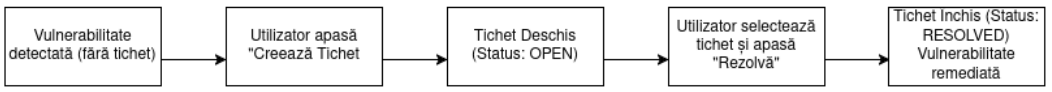
La nivelul interfeței cu utilizatorul, clasa **DashboardUI** reprezintă stratul de prezentare, construit modular folosind framework-ul Streamlit. Rolul acestei clase este strict de a solicita date din **DataRepository** și de a le randa vizual, fără a efectua procesări logice complexe. Structura sa include un modul Sidebar pentru navigare și conectare, precum și secțiuni distincte pentru funcționalități specifice: tab-ul Strategic, care utilizează biblioteca Plotly pentru generarea graficelor de risc, și tab-ul Operațional, care implementează formulare interactive pentru acțiunile de Write-Back (crearea și modificarea tichetelor).

Din punct de vedere al modelului de date, aplicația manipulează o serie de **entități** care, deși definite structural în baza de date, sunt gestionate în cod sub formă de obiecte sau structuri tabelare. Printre acestea se numără entitatea Tichet (obiectul central al fluxului operațional, definit prin ID, prioritate, status și note), entitatea Echipă (ce conține metrici de performanță precum scorul de eficiență și încălcările SLA) și entitatea Vulnerabilitate (o structură read-only ce furnizează detalii tehnice critice, inclusiv scorul CVSS). Această organizare garantează că eventualele modificări ale logicii de calcul din baza de date (de exemplu, ajustarea formulei de risc în PL/SQL) se propagă automat în aplicație fără a necesita rescrierea codului sursă al interfeței.

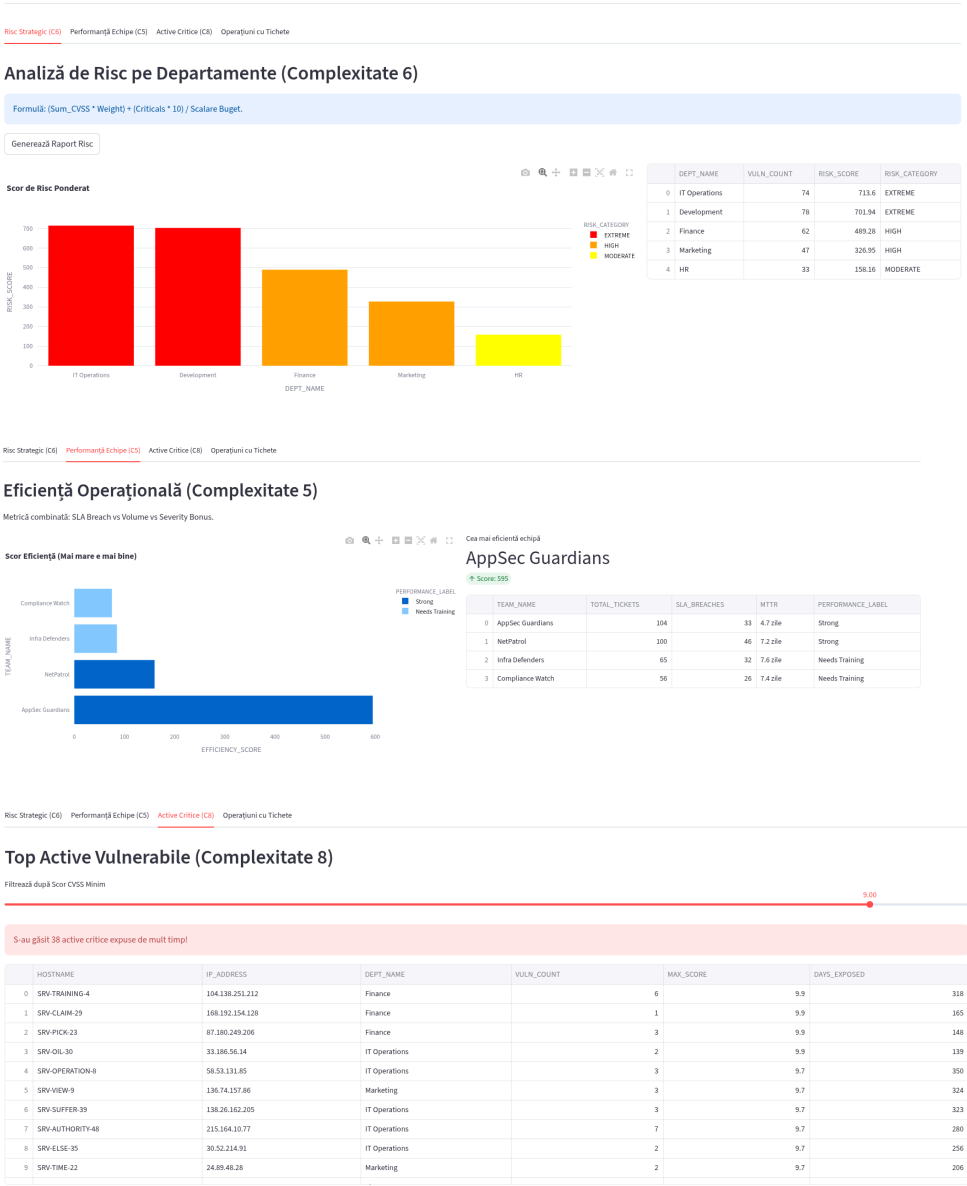


3.2 Diagrama de stări și fluxul de lucru (workflow) pentru aplicație

Totul începe cu etapa de **monitorizare strategică**, în care utilizatorul accesează dashboard-ul „Risc Strategic”, moment în care sistemul apelează în backend procedura `CALCULATE_DEPT_RISK` pentru a evidenția departamentele critice ce necesită intervenție imediată. Odată identificate zonele cu risc ridicat, utilizatorul trece la **analiza activelor critice**, filtrând vulnerabilitățile severe prin intermediul procedurii `PKG_VULNSIGHT.GET_CRITICAL_ASSETS` pentru a prioritiza serverele expuse. Această fază de decizie este urmată imediat de componenta de **dispecerizare**, unde se realizează operațiunile de scriere prin secțiunea „Deschide Tichet Nou”; aici, aplicația execută `SP_CREATE_TICKET`, validând integritatea datelor și alocând responsabilitatea unei echipe tehnice. Ciclul operațional se finalizează cu etapa de **remediere și închidere**, în care operatorul introduce detaliile soluției în secțiunea „Rezolvă Tichet”, declanșând procedura `SP_RESOLVE_TICKET` care actualizează statusul și calculează indicatorii SLA. Această acțiune închide bucla logică, astfel încât, la următoarea reîmprospătare a dashboard-ului, scăderea scorului de risc reflectă în mod direct îmbunătățirea securității în infrastructură.



4. Capturi de ecran



5. Concluzie

Proiectul **VulnSight** a demonstrat cu succes implementarea unui sistem informatic integrat, capabil să gestioneze ciclul de viață al vulnerabilităților dintr-o organizație, de la detecție și analiză strategică, până la remedierea operațională. Obiectivul principal, acela de a proiecta o arhitectură robustă bazată pe un SGBD relațional (Oracle Database), a fost atins prin delegarea completă a logicii de business către proceduri stocate și pachete PL/SQL.

6. Bibliografie

1. **Oracle Database Documentation** – docs.oracle.com.
2. **Streamlit Documentation** – docs.streamlit.io.
3. **Python-oracledb Driver Documentation** – python-oracledb.readthedocs.io.
4. **Plotly Python Graphing Library** – plotly.com/python/.
5. **Faker Library Documentation** – faker.readthedocs.io.