UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Balmanya Bocigas, Guillem. *Sistema d'Automatització de Consultes en un Sistema de Data Warehouse*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Grau en Enginyeria de Dades)

This version is available at https://ddd.uab.cat/record/337344

under the terms of the  license.

Sistema d'Automatització de Consultes en un Sistema de Data Warehouse (UAB)

Guillem Balmanya Bocigas 1671206

Resum— Aquest Treball de Final de Grau, realitzat en col · laboració amb l'Oficina de Govern de les Dades de la Universitat Autònoma de Barcelona, té com a objectiu redissenyar el procés d'extracció, transformació i càrrega (ELT) de les dades de recerca del CRIS extern d'Elsevier (EGRETA) cap al Data Warehouse institucional, a causa de la transició tecnològica del format XML al JSON. Després d'avaluar diverses alternatives en termes de rendiment i viabilitat, s'ha implementat una nova solució basada en Oracle Data Integrator i el motor Saxon que garanteix la continuïtat del servei, ja que aprofita la infraestructura existent i minimitza tant els costos d'adopció com la dependència d'eines externes.

Paraules clau— Ingesta de dades, ETL/ELT, Data Warehouse, Oracle Data Integrator (ODI), JSON, XML, API REST, CRIS (Current Research Information System), Apache NiFi, Python Pandas, Avaluació de rendiment i escalabilitat, Integritat i qualitat de les dades.

Abstract— Developed in collaboration with the Data Governance Office of the Universitat Autònoma de Barcelona, this Bachelor's Thesis aims to redesign the extraction, transformation, and loading (ELT) process for research data from Elsevier's external CRIS (EGRETA) into the university's Data Warehouse, driven by the platform's transition from XML to JSON format. After evaluating several alternatives based on performance and technical feasibility, a new solution utilizing Oracle Data Integrator and the Saxon engine was implemented; this approach ensures service continuity while maximizing the use of existing infrastructure, thereby minimizing adoption costs and reliance on external tools.

Index Terms— Data ingestion, ETL/ELT, Data Warehouse, Oracle Data Integrator (ODI), JSON, XML, REST API, CRIS (Current Research Information System), Apache NiFi, Python Pandas, Performance and scalability evaluation, Data integrity and quality.

---

1 Introducció del treball

El present treball aborda la modernització dels processos d'ingesta de dades de recerca de la Universitat Autònoma de Barcelona (UAB), gestionats per l'Oficina de Govern de les Dades. Davant el canvi tecnològic de la plataforma EGRETA (Elsevier), que migrarà el seu format de resposta de dades d'XML a JSON, aquest projecte se centra a dissenyar, avaluar i establir una nova pipeline d'integració que garanteixi la continuïtat i eficiència del sistema.

L'objectiu central és assegurar la transició cap a una arquitectura basada en JSON mitjançant l'estudi, desenvolupament i comparativa de diferents propostes d'implementació. L'anàlisi d'aquestes alternatives permetrà determinar la solució òptima que millor s'ajusti a les restriccions tècniques i d'infraestructura de la Universitat.

Mitjançant una metodologia en cascada o waterfall, s'han dissenyat i avaluat quatre propostes tecnològiques diferents per automatitzar l'extracció de les dades de l'API de recerca, la seva transformació en formats estructurats (CSV) i la seva posterior càrrega al Data Warehouse institucional. Aquestes propostes

abasten des de l'adaptació de l'entorn actual amb Oracle Data Integrator (ODI) i Saxon, passant per desenvolupaments a mida amb Python, fins a l'ús d'eines d'automatització de fluxos de dades com Apache NiFi.

El treball no només resol la necessitat tècnica de la migració de format, sinó que aporta una anàlisi exhaustiva que valora el rendiment dels processos d'extracció i transformació (ELT), el deute tècnic generat i la complexitat de manteniment. A partir d'aquesta comparativa, es documenta i s'implementa la solució definitiva triada.

1.1 Marc contextual

L'any 2021, la Universitat va decidir unificar tots els seus projectes de recerca sota una sola infraestructura, el CRIS (Current Research Information System), pertanyent a la companyia Elsevier, una empresa líder en gestió i anàlisi d'informació científica [15]. Aquest tipus d'infraestructures permeten centralitzar tota l'activitat de recerca, generar informes institucionals i crear portals públics d'investigadors [6,7].

Per poder explotar tota aquesta informació a nivell intern, la Universitat disposa d'un magatzem de dades (Data Warehouse) [8,10]. La filosofia d'aquesta arquitectura analítica es basa en la separació estricta entre els processos transaccionals (producció) i els processos d'anàlisi. Per tal d'evitar sobrecarregar la base de dades d'explotació i no interferir amb operacions crítiques de la Universitat (com ara períodes de matriculació o tancaments d'actes), mai es fan consultes analítiques directament a l'origen.

En lloc de consultar a origen, s'executa un procés ETL (Extracció, Transformació i Càrrega) programat que extreu les dades de producció i genera una capa intermèdia o còpia de seguretat anomenada BCK (Staging). Aquesta extracció cap a la BCK es realitza amb una freqüència setmanal. Posteriorment, amb una freqüència mensual, les dades es bolquen definitivament des de la BCK cap al Data Warehouse analític. Aquest cicle garanteix que sempre es treballi de forma aïllada i segura, encara que sigui amb les dades de la setmana o el mes anterior.

Pel que fa a les dades de recerca (les que provenen del CRIS d'Elsevier anomenat EGRETA), el procés d'extracció és particular perquè no resideixen als servidors de la Universitat, si no en la infraestructura d'Elsevier al núvol. Per tant, l'extracció cap a la BCK es realitza connectant-se als WebServices d'Elsevier mitjançant la Pure API i peticions RESTful [5].

Històricament, l'API d'Elsevier ha subministrat la informació en format XML (un llenguatge de marcatge jeràrquic basat en etiquetes) [3]. Tot i això, a causa de l'evolució dels estàndards web, el proveïdor ha anunciat l'obsolescència de l'XML en favor del format JSON (un format de dades molt més lleuger basat en col · leccions clau-valor) [2]. Aquesta transició tecnològica forçada invalida els fluxos d'integració que l'Oficina té actualment configurats a la seva plataforma Oracle Data Integrator (ODI).

El principal repte i limitació del procés d'extracció actual és que el motor de transformació configurat a ODI depèn de regles i plantilles dissenyades exclusivament per llegir arbres XML. Davant l'arribada del nou format JSON, es fa imprescindible redissenyar l'arquitectura per garantir la continuïtat del sistema.

2 Estat de l'art

2.1 Infraestructura i tecnologies actuals

A la Figura 1 s'il · lustra el procés complet que se segueix actualment per extreure les dades des de l'origen fins a la seva càrrega.

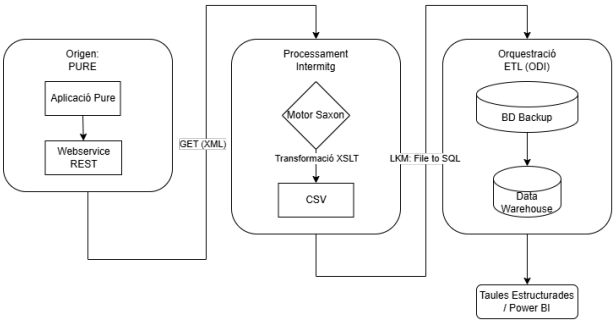


Figura 1. Esquema del procés d'extracció de les dades.

Origen de les dades: Sistema CRIS i Pure API

El punt de partida del flux de dades és el CRIS (Current Research Information System) d'Elsevier, concretament la plataforma EGRETA. L'extracció d'aquesta informació es realitza mitjançant la Pure API, un servei web basat en l'arquitectura REST (Representational State Transfer). Aquesta API utilitza mètodes HTTP estàndard (GET, POST, PUT, DELETE) per permetre una comunicació eficient i estandarditzada [5].

La informació s'estructura en diferents endpoints, que són adreces web específiques que donen accés a conjunts de dades concrets del model de recerca [19]. L'endpoint principal per l'estudi inicial és Awards, el qual proporciona informació sobre els ajuts de recerca. Per a l'exploració prèvia d'aquests endpoints i l'anàlisi de l'estructura de les dades de retorn, l'equip utilitza Postman, una eina fonamental per testejar peticions REST [9].

Integració i transformació: Oracle Data Integrator i Saxon

El nucli de l'automatització recau sobre Oracle Data Integrator (ODI), una plataforma d'integració de dades desenvolupada per la companyia Oracle basada en el paradigma ELT (Extract, Load and Transform) [11].

Ates que Pure API retorna les dades en format XML, l'arquitectura actual utilitza el motor Saxon. Aquest processador especialitzat s'encarrega d'executar transformacions XSLT per convertir els complexos arbres XML en fitxers plans (CSV), aplicant-hi la lògica de neteja i normalització abans de la ingesta.

Emmagatzematge: Staging i Data Warehouse

Abans d'arribar a la seva destinació final, les dades es carreguen a una capa intermèdia carregada setmanalment coneguda com a BCK. Finalment, la informació consolidada es bolca al Data Warehouse de la Universitat, desenvolupat sota tecnologia Oracle [20]. Aquesta base de dades analítica reuneix informació de múltiples orígens per permetre activitats d'intel · ligència de negoci i respondre preguntes estratègiques.

2.2 Tecnologies avaluades

Aquest treball no només adapta la infraestructura existent, sinó que explora i compara diferents alternatives d'implementació. Per a fer-ho, s'ha analitzat les següents tecnologies:

Saxon i XSLT 3.0

Per tal d'avaluar l'adaptació de l'entorn actual amb el mínim impacte possible, s'ha provat d'utilitzar l'estàndard XSLT 3.0. L'elecció d'aquesta versió es justifica per la seva capacitat nativa de processar JSON mitjançant funcions com json-to-xml(). Això permet transformar l'estructura JSON a XML de manera interna i, d'aquesta manera, reaprofitar les plantilles de transformació que la Universitat ja tenia desenvolupades per al motor Saxon [12,13,14].

Python i llibreries externes

Per a la segona i tercera iteració tecnològica s'ha apostat per Python, el llenguatge de referència actual en el tractament de dades massives.

D'una banda, s'ha utilitzat la llibreria Pandas (optimitzada en C), que ha estat escollida per la seva capacitat de manipular grans volums d'informació i processar estructures JSON de manera gairebé instantània, buscant el màxim rendiment [21].

D'altra banda, l'ús de Python pur (només amb llibreries estàndard com json i csv) s'ha introduït per complir amb la voluntat de la Universitat de no dependre d'eines o llibreries externes. Addicionalment, Python ha estat el llenguatge utilitzat per desenvolupar els scripts automatitzats de comprovació d'existència de fitxers, tests de qualitat i altres funcions auxiliars.

Apache NiFi i JOLT

Finalment, com a alternativa paradigmàtica a ODI i al desenvolupament d'scripts a mida, s'ha incorporat Apache NiFi. Aquesta decisió es justifica per la necessitat de provar una eina dissenyada de forma nativa per a l'automatització, ingesta i enrutament de fluxos

de dades (com crides REST i fitxers JSON) en temps real.

3 Objectius del treball

L'objectiu principal d'aquest treball és dissenyar, avaluar i validar una nova pipeline d'ingesta de dades que permeti la transició tecnològica del format XML al format JSON per a la informació de recerca de la UAB.

Per tal d'assolir aquest propòsit central, s'estableixen els següents objectius específics:

1. Adaptar i desenvolupar els processos d'ingesta.
2. Explorar i avaluar la capacitat de l'arquitectura actual.
3. Garantir la integritat i equivalència de les dades.
4. Dissenyar i comparar alternatives tecnològiques.
5. Validar i testejar en entorns reals.
6. Argumentar millores i documentar el procés.

4 Metodologia del treball

La metodologia emprada en aquest projecte segueix un enfocament en cascada, la planificació temporal del qual es detalla en el diagrama de Gantt de la Figura 12 [Apèndix 1]. L'execució s'ha estructurat al llarg de sis fases que abasten des de l'anàlisi inicial i definició de requeriments [Apèndix 2] fins al disseny i desenvolupament de les solucions. A continuació, s'ha aplicat un control de qualitat i validació de dades posterior a l'avaluació comparativa i la consegüent tria tecnològica. Les darreres etapes han correspost a l'escalat i mapatge, tancant el cicle amb l'elaboració de la documentació i les propostes de millora pertinents.

5 Desenvolupament

En aquesta secció i en les següents es descriuen detalladament els desenvolupaments realitzats durant el treball per assolir els objectius proposats.

5.1 Restriccions

De manera consensuada amb l'Oficina de Govern de les Dades, s'han establert una sèrie de restriccions i requisits que delimiten l'abast d'aquest projecte i actuen com a marc de referència per dissenyar i avaluar les alternatives tecnològiques. Aquestes directrius exigeixen assolir una integració completa (End-to-End), juntament amb la minimització de dependències externes i el rigorós compliment normatiu. Així mateix, cal consolidar l'adopció del format JSON cercant el millor equilibri possible entre la simplicitat, la contenció del deute tècnic i l'optimització del rendiment.

5.2 Esquema actual

El magatzem de dades (Data Warehouse), que actua com a destinació final d'aquesta solució, consolida la informació de recerca mitjançant un model relacional [22]. Aquest esquema, representat a la Figura 2, està format per múltiples taules, entre les quals destaquen algunes de les més rellevants, com ara Organizations, Prizes, Persons, Awards, Applications, Events, ExternalOrganizations i Activities.

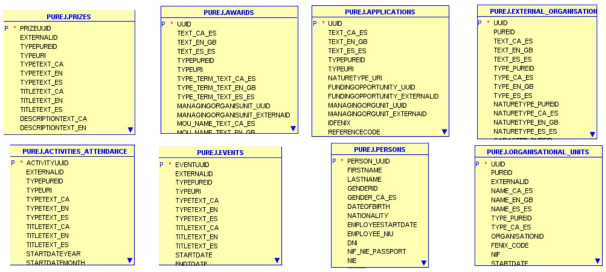


Figura 2. Esquema de taules parcial.

L'endpoint d'Awards, seleccionat com a base per al desenvolupament inicial de les solucions, centralitza tota la informació relativa a les subvencions i ajuts de recerca vinculats als programes d'investigació de la Universitat. Aquest registre emmagatzema de manera íntegra tant els projectes històrics com els vigents, amb un volum actual de 68.193 registres únics. Es considera el candidat ideal per iniciar el projecte a causa de la gran densitat, varietat i complexitat de la seva estructura de dades.

5.3 Comparativa entre els formats XML i JSON

Tot i que ambdós formats, tant XML com JSON, retornen dades funcionalment equivalents per al magatzem de dades, presenten diferències estructurals que afecten el disseny de la nova pipeline. El format JSON prioritza la concisió i la velocitat, reduint la sobrecàrrega d'informació, mentre que l'XML és més dens. L'anàlisi de les respostes ha revelat diferències en la informació proporcionada: el JSON detalla només els identificadors de molts blocs d'informació, mentre que l'XML opta per expandir per defecte algunes entitats relacionades sense obligar a fer crides addicionals.

5.4 Implementació

Totes les pipelines implementades comparteixen un disseny arquitectònic comú basat en tres etapes consecutives: en primer lloc, s'efectua una crida a l'API d'EGRETA per extreure les dades en el nou format JSON. A continuació, s'analitza l'estructura jeràrquica obtinguda, s'apliquen les transformacions i correccions pertinents i s'exporten les dades tractades a format CSV.

Finalment es realitza el procés de mapeig i la càrrega definitiva al Data Warehouse. En aquesta etapa s'estableixen les regles exactes de correspondència entre els camps de les dades prèviament transformades i el model final (vegeu l'esquema a la Figura 3).



Figura 3: Esquema del procés extracció, transformació i mapeig.

Per tal d'avaluar i comparar el rendiment de cadascuna d'aquestes implementacions, s'ha definit una metodologia de proves. D'una banda, per obtenir resultats fiables, totes les mètriques de temps d'execució s'han calculat a partir de la mitjana de cinc execucions independents per a cada solució, minimitzant així l'impacte de possibles fluctuacions. D'altra banda, s'han realitzat execucions simulant diferents volums de registres. Aquesta avaluació escalonada és fonamental atès que no tots els endpoints del sistema contenen el mateix volum d'informació, i permet observar de manera precisa com es comporta i escala cada alternativa.

5.4.1 Primera iteració: Adaptació ODI i Saxon (XSLT 3.0)

La primera implementació desenvolupada en aquest projecte ha sigut la creació des de zero del pipeline d'ingesta de dades utilitzant la plataforma ODI i el motor de dades Saxon per realitzar les transformacions. Aquesta implementació ha comportat la creació del workflow d'ODI (vegueu el workflow d'ODI a la Figura 4, Apèndix 1) encarregat de gestionar el pipeline i l'adaptació inicial de la infraestructura perquè funcioni amb el nou format JSON.

El canvi més rellevant ha sigut l'anàlisi, renovació i millora de les plantilles XSLT existents. Aquestes plantilles s'han actualitzat a la versió 3.0 de XSLT per permetre l'ús de funcions natives com json-to-xml() [12], que transformen l'estructura de la resposta JSON a XML internament, permetent així reaprofitar la lògica de transformació ja establerta. També s'han actualitzat les funcions pròpies emprades per la oficina (per processar números i camps multilingües) i s'han afegit comprovacions de dates per a camps prèviament absents.

Com a millora addicional a la infraestructura, s'ha dissenyat un script en Python que centralitza i automatitza la verificació prèvia de l'existència dels fitxers necessaris, substituint l'antiga gestió manual de variables d'ODI i facilitant-ne el manteniment.

Respecte al rendiment, aquesta implementació ha requerit un total de 693 segons per processar la totalitat dels registres d'Awards. Si analitzem el desglossament del temps per fase del procés de transformació podem veure com el 53,7% del temps d'execució es dedica a crides a l'API, el 39,8% es dedica al processament de JSON i CSV, el 6,1% als test de qualitat i finalment 0,4% restant es dedica en les validacions prèvies.

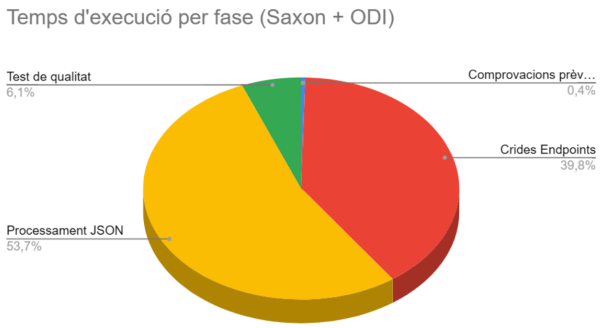


Figura 6: Temps d'execució per fase de la implementació ODI i Saxon.

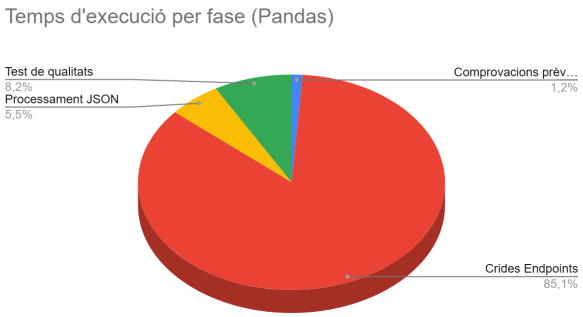


Figura 7. Temps d'execució per les diferents fases de la implementació amb Pandas.

5.4.2 Segona iteració: Processament extern amb Python (Pandas)

Amb l'objectiu de millorar el rendiment i explorar solucions fora del motor Saxon, el següent pas ha consistit a construir un script en Python. S'ha triat aquest llenguatge per la seva alta eficiència en el tractament de dades massives i, específicament, perquè disposa de la llibreria Pandas, una eina optimitzada en C que permet processar estructures JSON i manipular grans volums d'informació de manera gairebé instantània [21].

Amb aquesta nova implementació, el temps d'execució total s'ha reduït dràsticament a 258 segons per al mateix volum de dades (una millora de 2,69 vegades respecte a l'implementació amb Saxon). Aquesta reducció s'explica per un processament més eficient de la resposta JSON a format CSV.

Si desglossem ara el rendiment per fase d'execució, podem comprovar com s'ha aconseguit una reducció de 26,22 vegades (o de 26,22x) en el processament del fitxer JSON, que ha passat de 372 a 14 segons (tal i com es mostra a la Figura 7). Aquesta implementació, per tant, suposa una reducció molt significativa del temps d'execució, el qual ara només està limitat per les crides als endpoints.

Aquesta versió, però, planteja problemes d'escalabilitat del codi. Com que s'utilitzen biblioteques específiques per a la manipulació de les dades, és necessari desenvolupar codi específic per replicar la lògica de transformació, la qual cosa redueix la generalització i augmenta el cost de manteniment del procés.

5.4.3 Tercera iteració: Processament nadiu amb Python pur (Sense llibreries externes)

Un dels requeriments crítics de l'Oficina implica minimitzar la dependència d'eines externes. Per respondre a aquesta necessitat, i després de comprovar les millores obtingudes amb la implementació anterior basada en la llibreria Pandas, s'ha desenvolupat una tercera iteració que empra exclusivament les llibreries estàndard de Python (json, csv).

Aquesta solució aconsegueix temps d'execució molt similars als esmentats en la implementació anterior (pràcticament idèntics, 2,65x més ràpids respecte a l'ODI tal com es mostra a la Figura 8), però amb un cost de personalització altíssim. Ha requerit 1,74 vegades més línies de codi (o un 74% més), ja que cal gestionar de manera manual l'exploració de l'arbre JSON, així com els valors nuls o els tipus de dades per a cadascuna de les antigues plantilles XSLT, a més de tota la lògica que ja integren les mateixes plantilles. Escalar aquesta solució a producció per a tots els

endpoints suposaria una càrrega de desenvolupament poc assumible per l'Oficina i comportaria uns costos de manteniment i un deute tècnic molt grans.

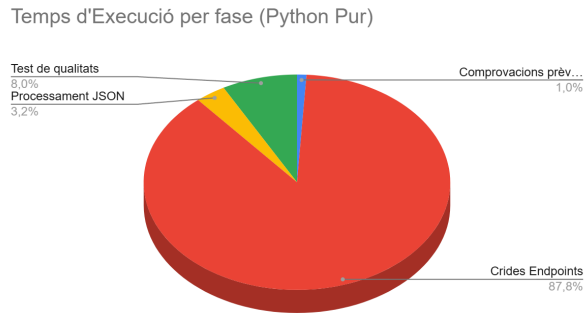


Figura 8. Temps d'execució per les diferents fases de la implementació amb Python nadiu.

5.4.4 Quarta iteració: Processament amb Apache NiFi

Atès que les solucions basades en codi a mida requereixen un nivell de personalització i manteniment massa elevat, s'ha cercat una alternativa més escalable. L'objectiu és reduir dràsticament la necessitat de codi específic i provar un motor d'extracció diferent per comprovar si el coll d'ampolla en la descàrrega és propi d'ODI o d'Elsevier. Aquesta iteració s'ha dut a terme amb Apache NiFi, una eina dissenyada de forma nativa per ingerir, enrutar i transformar dades en temps real (com crides REST i fitxers JSON).

Pel que fa a la implementació automatitzada, el flux extreu, fragmenta i transforma les dades de l'API. El punt clau d'aquesta alternativa és l'ús de la tecnologia JOLT [4], que elimina totalment la necessitat de codi a mida. Mitjançant un simple fitxer de configuració estructurat s'indica com aplanar el JSON, reanomenar camps i extreure valors niats de manera automàtica. Un cop transformades, el flux agrupa les dades i les converteix sobre la marxa a format CSV.

Aquesta solució presenta l'avantatge clar d'eliminar el deute tècnic gràcies a JOLT, ja que afegir una nova plantilla només requereix ajustar un mapeig sense mantenir codi complex. Com a contrapartida, implica introduir i administrar una nova peça d'infraestructura al sistema.

Respecte al rendiment d'aquesta implementació, tal com es pot veure a la Figura 9, Apèndix 1, es posiciona en un punt intermedi entre la velocitat de les implementacions amb Python i la de Saxon, amb un temps d'execució total de 421 segons (1,64x més ràpid que Saxon i 0,62x més ràpid que Python).

5.5 Tria Final, Mapeig i Càrrega

Abans de procedir a la selecció de la implementació final, i aplicant la metodologia d'avaluació definida prèviament, s'ha analitzat l'escalabilitat de les quatre alternatives. L'objectiu és verificar els patrons de creixement de les implementacions.

Els resultats d'aquesta comparativa (vegeu la Figura 10) confirmen que ODI és la solució que requereix un temps d'execució més elevat en tots els escenaris, independentment de la mida de la mostra. Respecte a les altres tres alternatives, s'observa un rendiment molt similar en volums petits. Tot i això, a partir dels 20.000 registres, tal com es pot veure a la Figura 10, el temps d'execució d'Apache NiFi comença a augmentar. Això passa perquè NiFi no està pensat per a la velocitat pura, sinó per a la seguretat. Per evitar pèrdues, l'eina guarda constantment l'estat de les dades al disc dur i en vigila el camí. Aquesta gestió interna fa que, amb volums grans de registres, vagi més lent que un script de Python [23, 24]. Finalment, les dues implementacions basades en Python mantenen un creixement lineal i mostren uns temps d'execució molt similars en qualsevol volum de dades.

Si ens fixem detingudament en la descomposició dels temps d'execució (revisant comparativament les Figures 6, 7 i 8), es fa evident una limitació inherent a l'arquitectura, el temps dedicat exclusivament a les crides a l'API representa un coll d'ampolla inalterable que la Universitat no pot modificar directament, atès que depèn de l'estat i infraestructura de la xarxa d'Elsevier. De fet, si comparem aquests temps de resposta amb els d'altres APIs REST massives i modernes (com les de GitHub, Google o AWS) [16, 17,

18], el temps servit per EGRETA resulta notablement elevat per volums similars.

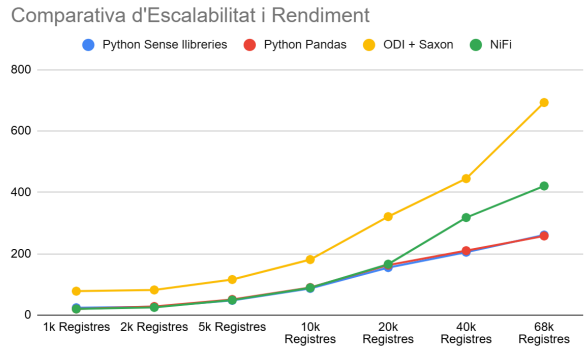


Figura 10: Comparativa d'Escalabilitat i Rendiment per les implementacions.

Abans de prendre una decisió definitiva, s'ha dut a terme un balanç de les diferents implementacions avaluades. Per una banda, les implementacions basades en Python (tant amb la llibreria Pandas com amb llibreries estàndard) oferien la velocitat d'execució més alta i un creixement lineal òptim, però comportaven un deute tècnic elevat i uns costos de manteniment alts en requerir codi a mida per a cada endpoint. Per altra banda, la solució amb Apache NiFi destaca per la seva modularitat i l'eliminació del codi gràcies a l'enfocament amb JOLT, però introdueix una nova peça d'infraestructura que cal administrar i requereix una corba d'aprenentatge específica. En contrast, la combinació d'ODI i Saxon presenta com a principal desavantatge el rendiment més limitat de les quatre opcions, però ofereix el benefici crucial d'aprofitar tota la infraestructura existent i minimitzar el risc de fallada en els processos existents.

Després de compartir els resultats d'aquests anàlisis amb l'Oficina, s'ha seleccionat formalment la solució amb Saxon. Tot i no ser l'opció amb el rendiment més elevat, és la que comporta un menor impacte en els procediments i la infraestructura actuals, fet que s'adapta perfectament als objectius finals de l'Oficina. Aquesta implementació és una alternativa conservadora que garanteix una transició de format satisfactòria sense comprometre l'estabilitat del sistema i aprofitant també la lògica de negoci i les transformacions ja establertes. A més, aquesta opció no requereix cap corba d' aprenentatge per part de l'equip, ja que utilitza les mateixes eines que ja formen part de la seva infraestructura actual.

Un cop establerta la tecnologia per a l'extracció i la transformació, s'inicia la tercera i última etapa del procés, centrada en el mapeig. En aquesta etapa, s'estableixen les relacions dels camps processats cap als seus equivalents en el model de dades relacional del Data Warehouse. Finalment, aquesta mateixa solució s'ha escalat a la resta d'endpoints rellevants.

5.6 Marc de Validació i Qualitat de les Dades

Més enllà de l'optimització del rendiment, la preservació de la integritat i qualitat de les dades és un requisit fonamental. Per assegurar-ho, s'ha desenvolupat un mòdul de verificació automatitzada mitjançant un script de Python i un fitxer de configuració JSON per a cada plantilla XSLT. Aquest procés valida els fitxers CSV obtinguts executant comprovacions de normalització de dades, validació d'estructura, recompte de files, detecció de duplicats, comparació de contingut i validació de camps numèrics.

5.6.1 Problemes de Qualitat de Dades

Les validacions implementades i l'anàlisi de les respostes JSON han permès detectar deficiències en la qualitat de les dades d'origen. S'ha detectat la manca dels pressupostos a les entitats d'Awards i Applications, i l'absència de la informació dels supervisats a l'endpoint de Persons. Tot i que l'Oficina de Govern de les Dades ha confirmat que actualment aquesta informació no és crítica, s'ha optat per dissenyar i documentar una via de solució.

Aquesta solució proposa un flux iteratiu que extreu l'identificador únic de cada registre principal per realitzar crides subsidiàries a endpoints auxiliars. Un cop obtingut el detall d'aquests camps absents, s'executen les transformacions necessàries per consolidar la informació final, assumint-ne el consegüent cost de rendiment (a causa de l'alt volum de crides addicionals).

6. Proposta de Millores

A partir de la informació obtinguda i de les característiques observades en el disseny i la implementació de les diferents alternatives, s'ha definit un conjunt de millores amb l'objectiu d'explorar optimitzacions i millores.

6.1 Crida API personalitzada

Com a primera via d'optimització, s'ha construït un script específic en Python encarregat de gestionar directament les peticions a l'API. L'objectiu d'aquesta prova empírica és determinar si la latència observada durant les crides als endpoints és una limitació pròpia de l'eina d'integració o si, per contra, és una restricció inherent a la infraestructura de Elsevier.

Un cop realitzades les proves pertinents, els resultats obtinguts han reflectit un temps d'execució de 252 segons. En comparar aquesta xifra amb les mètriques d'extracció registrades prèviament, no s'han observat diferències significatives en el rendiment. Per tant, aquest anàlisi permet concloure de manera fonamentada que el coll d'ampolla en la fase de descàrrega no ve derivat del programari d'integració, sinó que recau exclusivament sobre els temps de resposta i la infraestructura dels servidors d'Elsevier.

6.2 Canvi en l'Estructura de Taules Actual

Com a segona proposta de millora, es planteja un redisseny estructural de les taules finals del magatzem de dades. L'objectiu d'aquesta mesura és reduir el nombre de camps existents per conservar exclusivament aquells dels quals es pugui extreure informació real a partir del nou format JSON, delegant la lògica de creuament de dades (joins) a fases analítiques posteriors. Aquesta reestructuració evita el manteniment de columnes que quedarien íntegrament buides en la nova implementació, optimitzant de manera significativa tant l'espai d'emmagatzematge com el cost computacional del motor de base de dades.

Per l'endpoint Awards, l'aplicació d'aquest redisseny proposa consolidar la informació de les diferents taules auxiliars actuals en una única taula principal adaptada exclusivament a les respostes JSON, tal com es pot veure en la Figura 11, Apèndix 1. Aquesta mesura implica una reducció del 58,14% en el nombre total de camps (passant de 43 a 18).

7. Conclusions

El present treball ha assolit amb èxit l'objectiu de dissenyar i validar una nova pipeline d'ingesta de dades de recerca, garantint la continuïtat del sistema davant la migració de l'API d'EGRETA al format JSON. Després d'avaluar i comparar quatre alternatives tecnològiques, la selecció final s'ha decantat per la solució basada en Oracle Data Integrator (ODI) i Saxon. Tot i presentar el temps d'execució més elevat, ha demostrat ser la decisió més estratègica i pragmàtica, ja que permet reaprofitar la infraestructura i el coneixement tècnic existents, evitant noves dependències i minimitzant el deute tècnic i el risc en la transició a producció.

D'altra banda, el desenvolupament d'un marc de validació i qualitat ha garantit la integritat total de les dades durant la transformació i ha permès detectar i documentar mancances en la informació d'origen. En conclusió, el projecte no només resol la necessitat tècnica inicial de manera robusta i escalable, sinó que dota la UAB d'un valuós anàlisi de rendiment i deixa el terreny preparat (amb un model segur i documentat) per a futures reestructuracions del magatzem de dades institucionals.

---

1. *E-mail de contacte: 1671206@uab.cat*
2. *Treball tutoritzat per: Espinosa Morales, Antonio (Àrea d'Arquitectura i Tecnologia de Computadors)*
3. *Curs 2025/2026*

8 Bibliografia

1. AWS. (s.d.). Difference between JSON and XML. Amazon Web Services. Recuperat de https://aws.amazon.com/es/compare/the-difference-between-json-xml/
2. AWS. (s.d.). What is JSON? Amazon Web Services. Recuperat de https://aws.amazon.com/es/documentdb/what-is-json/
3. AWS. (s.d.). What is XML? Amazon Web Services. Recuperat de https://aws.amazon.com/what-is/xml/
4. Bazaarvoice. (s.d.). JOLT. GitHub. Recuperat de https://github.com/bazaarvoice/jolt
5. Elsevier. (s.d.). Pure API documentation. Elsevier. Recuperat de https://api.elsevierpure.com/ws/api/documentation/index.html
6. euroCRIS. (s.d.). What is euroCRIS? euroCRIS. Recuperat de https://eurocris.org/what-eurocris
7. euroCRIS. (s.d.). Why does one need a CRIS? euroCRIS. Recuperat de https://eurocris.org/why-does-one-need-cris
8. IBM. (s.d.). What is a data warehouse? IBM. Recuperat de https://www.ibm.com/mx-es/think/topics/data-warehouse
9. OpenAPI. (s.d.). Postman: What it is and how it works. OpenAPI. Recuperat de https://openapi.com/blog/postman-what-it-is-and-how-it-works
10. Oracle. (2024). Introduction to data warehouse concepts. Oracle Database Data Warehousing Guide. Recuperat de https://docs.oracle.com/en/database/oracle/oracle-database/26/dwhsg/introduction-data-warehouse-concepts.html
11. Oracle. (s.d.). Oracle Data Integrator Documentation (14.1.2). Oracle Help Center. Recuperat de https://docs.oracle.com/en/middleware/fusion-middleware/data-integrator/14.1.2/index.html
12. Saxonica. (s.d.). Saxon Documentation: fn:json-to-xml. Saxonica. Recuperat de https://www.saxonica.com/html/documentation12/functions/fn/json-to-xml.html
13. Saxonica. (s.d.). SAXON XSLT 3.0 and XQuery 3.1 processor documentation. Saxonica. Recuperat de https://www.saxonica.com/documentation/
14. Saxonica. (s.d.). What is Saxon? Saxonica. Recuperat de https://www.saxonica.com/html/documentation12/about/whatis.html
15. Universitat Autònoma de Barcelona. (2021). Informe sobre el ciberatac – Claustre desembre 2021. UAB. Recuperat de https://www.uab.cat/doc/informe-ciberatac-claustre-des21
16. GitHub. (s.d.). GitHub REST API Documentation. GitHub. Recuperat de https://docs.github.com/en/rest
17. Google Cloud. (s.d.). API Design Guide: Performance. Google Cloud. Recuperat de https://cloud.google.com/apis/design/performance
18. Amazon Web Services. (s.d.). Best practices for Amazon API Gateway. AWS. Recuperat de https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-metrics-and-dimensions.html
19. IBM. (2024). What is an API endpoint? Think Topics. Recuperat de https://www.ibm.com/think/topics/api-endpoint
20. Oracle. (2025). What is a Data Warehouse? Oracle. Recuperat de https://www.oracle.com/database/what-is-a-data-warehouse/
21. NVIDIA. (2025). What is Pandas in Python? NVIDIA Glossary. Recuperat de https://www.nvidia.com/en-us/glossary/pandas-python/
22. Databricks. (2025). What is a Relational Data Model? Databricks Blog. Recuperat de https://www.databricks.com/blog/what-is-a-relational-data-model
23. Apache Software Foundation. (2026). Apache NiFi Overview: Design Goals and Core Concepts. Apache NiFi. Recuperat de https://nifi.apache.org/docs/nifi-docs/html/overview.html
24. Apache Software Foundation. (2026). Apache NiFi System Administrator's Guide: Repository Configuration. Apache NiFi. Recuperat de https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html

9 APÈNDIX

A.1 Figures

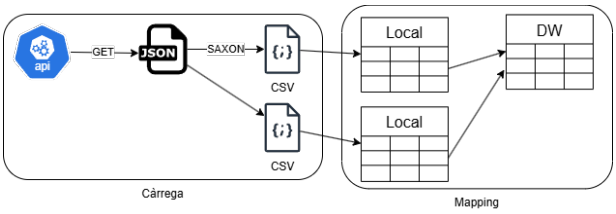


Figura 4: Pipeline d'ingesta de dades amb el motor Saxon a ODI.

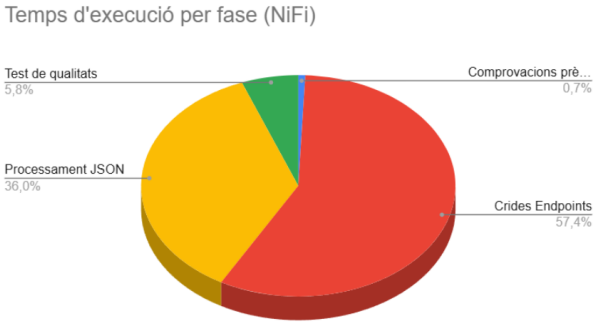


Figura 9: Temps d'execució per les diferents fases de la implementació amb Apache NiFi.

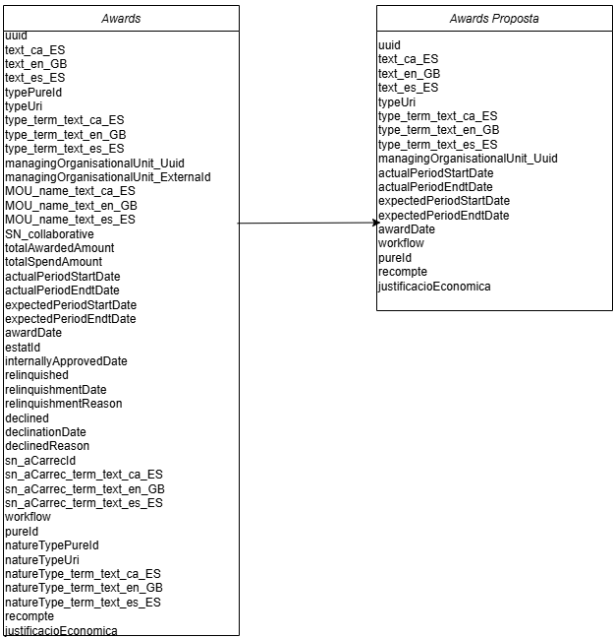


Figura 11. Proposta d'actualització de taules.

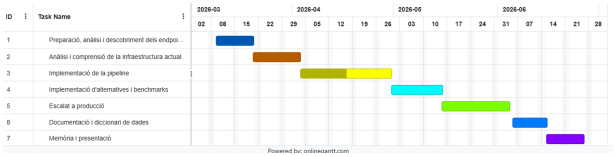


Figura 12. Diagrama de Gantt.

A.2 Anàlisi detallat de camps XML i JSON

El llenguatge de marcat extensible o XML és un llenguatge de marcatge que permet definir i emmagatzemar dades de forma compatible amb altres sistemes. Aquest llenguatge permet l'intercanvi d'informació entre sistemes tan diferents com sistemes de computació, pàgines web, bases de dades i aplicacions de tercers. Els fitxers en format XML compleixen unes normes i una estructura semirígida que facilita aquest intercanvi i assegura la compatibilitat. A diferència d'altres llenguatges de programació, no és capaç de realitzar operacions de computació per si mateix, sinó que és la base sobre la qual, mitjançant altres llenguatges (com en aquest cas Python), es realitzen

operacions per a l'administració estructurada de les dades.

Aquest llenguatge s'estructura principalment basant-se en etiquetes, que són símbols de marcatge que s'utilitzen per definir les dades que es representen en el fitxer. Aquestes estructures estan delimitades pels símbols < i > i acostumen a tenir noms significatius de la informació que representen. Un arxiu XML es pot dividir en diversos components: a l' inici conté metadades sobre la versió o codificació del document, seguit de les etiquetes arrel que indiquen el principi i final del contingut. Així mateix, aquestes etiquetes poden contenir descriptors que amplien la informació disponible sobre un element. Aquest format és molt utilitzat gràcies a la seva flexibilitat, ja que permet guardar informació amb la màxima fidelitat possible, transferir les dades juntament amb la seva pròpia descripció semàntica i garantir-ne l'alta integritat entre múltiples plataformes.

En el context històric de l'API d'EGRETA, la resposta en format XML basa la seva representació en aquest model jeràrquic d'etiquetes. La resposta s'organitza mitjançant un objecte embolcall (wrapper) que facilita la gestió de la paginació i inclou metadades de control, com ara el nombre total de resultats disponibles i els marcadors de posició. A l'interior d'aquest embolcall, la informació de cada registre principal (com ara els ajuts a la recerca) es desglossa en elements complexos que relacionen identificadors secundaris, dates, participants i les unitats organitzatives vinculades. Tot i ser un format altament estandarditzat i robust, la seva sintaxi en forma d'arbre és molt densa i completa, la qual cosa resulta en fitxers més pesants i estructures més complexes de processar computacionalment.

D'altra banda, JavaScript Object Notation o JSON és un format de dades obert i intercanviable que és apte per a la lectura tant per humans com per màquines. El nou format proposat per Elsevier (JSON) ofereix una representació basada en col · leccions de parelles de clau-valor i llistes (arrays). Aquest format comparteix amb l'XML l'ús d'un objecte embolcall per al control de la paginació, on les dades reals es troben llistades al seu interior. No obstant això, estructura les dades internes de cada entitat de manera molt més directa. La informació detallada dels projectes, les dates d'execució, els investigadors implicats i les entitats finançadores s'hi emmagatzemen en un format lleuger. Aquesta optimització de la sintaxi millora dràsticament el rendiment de les crides de xarxa i accelera la lectura i transformació per part dels sistemes d'integració automatitzats.

A.3 Taules de Rendiment i Escalabilitat

A continuació es detallen els resultats empírics obtinguts durant la fase de proves i avaluació de les diferents implementacions proposades. La **Taula 1** mostra el desglossament del temps d'execució mitjà (en segons) de cadascuna de les fases de la pipeline per a l'endpoint principal d'Awards (amb un volum de 68.193 registres). Aquests resultats representen la mitjana aritmètica obtinguda en 5 execucions independents per a cadascuna de les quatre alternatives.

Implementació	Total	Crida API	Proc. JSON	Test Qualitat	Val. Prèvies
ODI i Saxon	693,2	276,0	372,4	42,2	2,6
Python (Pandas)	258,0	219,6	14,2	21,2	3,0
Python (Pur)	261,4	229,6	8,4	20,8	2,6
Apache NiFi	421,0	241,8	151,6	24,6	3,0

Taula 1: Mitjana de temps d'execució per fases (en segons)

La **Taula 2** il · lustra l'escalabilitat i el patró de creixement del temps d'execució total (en segons) per a cada alternativa tecnològica a mesura que incrementa el volum de registres processats.

Registres	Python Pur	Python Pandas	Apache NiFi	ODI i Saxon
1.000	24	20	20	78
2.000	27	28	25	82
5.000	48	51	49	116
10.000	87	90	89	181
20.000	155	163	166	321
40.000	205	210	318	445
68.000	261	258	421	693

Taula 2: Temps d'execució segons volum de registres (en segons)