UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria “Enzo Ferrari”

Corso di laurea in Ingegneria Informatica

SIARD Format: analisi di un nuovo modo di preservare database

Relatore

Prof.ssa Sonia Bergamaschi

Correlatore

Phd Luca Gagliardelli

Candidato

Davide Bucciarelli

Anno Accademico 2021/2022

Indice

1 Introduzione

Lo scopo di librerie, archivi ed altre simili istituzioni è quello di preservare documenti con alto valore scientifico, artistico o sociale, in modo tale che essi siano accessibili dalle future generazioni come sono disponibili a noi oggi. La conservazione di antichi libri o documenti in forma cartacea richiede tecniche estremamente complesse e costose, per limitare il naturale degrado dei materiali di cui sono composti. Da questo punto di vista, le informazioni digitali sembrano avere proprietà ideali: flussi di bit possono facilmente essere conservati senza che vi sia alcuna perdita di informazioni, possono essere copiati o spostati su supporti diversi con minima difficoltà e siamo ormai capaci di mantenere quantità enormi di informazioni su dispositivi dalle piccole dimensioni e basso costo. Quindi, qual è il problema?

La questione è posta dalla seguente domanda: Fra cinquant'anni, esisterà ancora un dispositivo capace di leggere le informazioni conservate digitalmente su un supporto realizzato al giorno d'oggi, e presentarlo in un formato comprensibile ad un umano? Questa domanda sorge dal fatto che i dati non sono fruiti direttamente dall'essere umano, ma abbiamo bisogno di specifiche macchine capaci di decodificare una sequenza binaria costruita seguendo norme ben definite e mostrarla in una maniera che sia facilmente assimilabile dall'utente finale. [1]

L'obiettivo di questa tesi è l'analisi di un sistema di preservazione di dati, in particolare di database relazionali, che tenta di superare il problema presentato precedentemente. Il sistema in questione è il formato SIARD, che si basa sui principi della normalizzazione per convertire database proveniente da qualunque DBMS in un file SIARD, basato su standard open source e di ampia adozione.

2 La preservazione dei dati

In un contesto computazionale, il concetto di conservazione di un dato si riferisce al mantenimento dello stesso per un determinato periodo di tempo, in particolare, la preservazione a lungo termine si riferisce ad un periodo di tempo che varia fra i cinquanta ed i cento anni. L'archiviazione di Database è un caso particolare del "*data archiving*", che si focalizza sull'archiviazione e mantenimento di dati che sono sotto il controllo di un Database Management System (DBMS) e strutturati sotto uno schema di database, come ad esempio un database relazionale. L'obiettivo primario è ovviamente quello di mantenere l'accesso a tali dati in caso siano richiesti. [2]

2.1 Database Archiving

Il termine Database archiving può avere varie declinazioni, ma la definizione più comune è la seguente: *Il processo di rimuovere da database operazionali certi record che non ci si aspetta che verranno utilizzati in futuro e conservarli in un archivio dove possono essere recuperati in caso di bisogno.* Secondo questa definizione, l'archiviazione di un database richiede quindi una selezione dei dati che non sono più richiesti in condizioni normali ed il loro trasferimento su un diverso supporto adatto alla preservazione a lungo termine. I dati sono allora mantenuti in archivio per il periodo di tempo richiesto, alla fine del quale verranno poi trasferiti o distrutti.

Un'altra forma di *Database Archiving,* usata in particolare in ambito scientifico, utilizza un approccio diverso: viene mantenuta una serie di "fotografie" del database collezionate nel tempo, creando quindi una serie di copie che, gestite in modo efficiente, non solo possono fornire un'immagine completa del database in un determinato momento del passato, ma danno anche la possibilità di poter tracciare l'evoluzione dei dati nel tempo. [2]

2.2 Problemi dell'archiviazione di database

Dati da fonti diverse: può diventare difficile la gestione di dati di tipologia diversa, provenienti da sorgenti eterogenee come, ad esempio, DBMS forniti da vendor differenti.

- **Trasferimento dei record:** il trasferimento di record da un database "live" all'archivio deve essere eseguito assicurandosi il mantenimento dei vincoli di integrità sia nel database sorgente, che nell'archivio.
- **Velocità di accesso all'archivio:** I dati negli archivi devono essere accessibili in un periodo di tempo ragionevole, senza richiedere particolari manipolazioni: ad esempio se l'archivio è conservato su nastro magnetico, può essere richiesto molto tempo per l'accesso ad uno specifico record.

- **Dimensione dei dati archiviati:** Soprattutto in ambito scientifico, la dimensione dei dati da archiviare può diventare facilmente un fattore limitante. La manipolazione di grande quantità di dati, anche con le tecnologie attuali può richiedere molto tempo.

2.3 Problemi specifici dell'archiviazione a lungo termine

A ciò che è stato elencato si aggiungono poi questioni specifiche che riguardano l'archiviazione dei dati a lungo termine. Il principale problema, in questo caso, consiste nell'usabilità dei dati archiviati. Con le tecnologie disponibili al giorno d'oggi, infatti, diventa secondario l'aspetto della preservazione fisica dei dati, che grazie a sistemi di storage ridondanti ed estremamente affidabili è pressoché garantita, anche per periodi di tempo molto lunghi. Il dato, inteso come sequenza di bit, diventa però del tutto inutile ed irrecuperabile se il software un tempo usato per la sua fruizione non è più disponibile, e questa evenienza è estremamente probabile nel caso dell'archiviazione a lungo termine. Si possono definire due casi distinti: i dati ed il codice. Data l'esperienza degli ultimi decenni, si può affermare che sia estremamente difficile, se non impossibile, la conservazione a lungo termine di codice, essendo esso troppo mutabile nel tempo ed essendo singoli software molto interdipendenti fra loro. Per quanto riguarda, invece, i semplici dati, la situazione è più semplice: vi sono standard definiti a livello mondiale che sono rimasti pressoché invariati negli ultimi cinquant'anni (ad esempio il codice ASCII), e questo fa sì che i dati abbiano una aspettativa di vita più lunga del codice. Tale problema si riflette in particolar modo nell'ambito dei database relazionali, che dipendono interamente da software proprietari per essere manipolati. Com'è possibile, quindi, la preservazione a lungo termine di un database? [3]

2.4 Due soluzioni classiche

Attualmente i principali approcci per l'archiviazione a lungo termine di database sono due: l'emulazione o la migrazione.

L'emulazione consiste nel conservare una copia, sia essa fisica o simulata, dell'hardware originale, sul quale poi potrà essere eseguito il software da conservare o poter accedere ai dati preservati. Questo approccio, per i database relazionali porta alla necessità di dover preservare non solo i dati conservati nei database, ma anche le specifiche piattaforme software dalle quali tali dati provengono. Questo è quindi come dover conservare uno specifico "ambiente" di funzionamento per ogni specifico oggetto preservato, e ciò è molto oneroso.

La migrazione consiste invece nel trasformare il materiale archiviato in un nuovo formato, ogni volta che il formato di partenza non è più supportato. Questa operazione non è però sempre banale, soprattutto nel caso dei database relazionali, dove vi sono un gran numero di DBMS, ognuno con

diverse caratteristiche che li rendono incompatibili fra loro. Per questo è molto difficile che il trasferimento di un database da un DBMS non più supportato ad uno differente di nuova generazione possa avvenire senza dover modificare manualmente la struttura o i dati conservati nel database, e questo può essere molto oneroso in termini di tempo e risorse. [3]

3 La Normalizzazione

3.1 Una nuova proposta

La normalizzazione è un approccio che mira alla conservazione di dati in modo tale che il materiale archiviato rimanga invariato per sempre. Questo può avvenire se esso è preservato in una forma "normale" per il tipo di dato archiviato: uno standard che sia ampiamente diffuso, sulla base del quale, in qualunque momento chiunque possa programmare un software di conversione per poter de-normalizzare l'oggetto e riportarlo ad una forma fruibile dall'utente finale. Questo metodo quindi, assumendo che esista uno standard ben documentato e di ampia adozione per una determinata tipologia di oggetto, permette di conservare i dati in forma normalizzata senza mai doverli manipolare, e per accedervi bisognerà avere disponibile un software di normalizzazione e de-normalizzazione compatibile con la tecnologia attuale. Questo approccio si presta in particolar modo per l'archiviazione di database, che per la loro varietà avrebbe portato ai problemi elencati sopra. Grazie alla normalizzazione sarebbe quindi possibile "standardizzare" database di ogni tipo conservandoli in un unico formato ben noto, che poi sarà possibile de-normalizzare grazie ad un singolo software che dovrà essere tenuto aggiornato. [3]

3.2 Normalizzazione: cosa preservare?

Non esiste un unico formato, di ampia adozione per la normalizzazione e l'archiviazione di database, possiamo però definire una serie di caratteristiche fondamentali che questo formato dovrebbe avere, definendo quali aspetti del database è effettivamente necessario preservare per ottenere un archivio che sia utile senza dover conservare dati superflui. Ad esempio, la normativa prevede, per la maggior parte degli archivi a livello nazionale, la preservazione delle informazioni, ma non della loro forma. Nel caso dei database questo significa che, almeno a livello normativo, non è necessario conservarne l'intera struttura. Il database relazionale, quindi, che possiamo vedere come un insieme di tabelle che contengono l'informazione primaria, legate fra loro da relazioni, è considerabile completamente preservato se riusciamo a conservare il contenuto informativo di ogni singola cella di ogni tabella di cui esso è composto, insieme con i metadati necessari a ricostruire le relazioni fra queste.

Si può affermare che i database relazionali siano definiti dal linguaggio di interrogazione usato per estrapolarne le informazioni: il linguaggio SQL (*Structured Query Language*) è ciò che a tutti gli effetti definisce cosa è un database relazionale. Quindi, dato che siamo solo interessati al mantenimento dell'accesso ai dati, non alla loro modificazione, possiamo concludere che un

database relazionale è completamente preservato se la stessa query SQL *SELECT,* eseguita sul database originale e sul database preservato, genera il medesimo risultato. [3]

3.3 Criterio di completa preservazione

È possibile allora separare la questione della conservazione completa di un database relazionale in due problematiche ben distinte: la preservazione dei metadati e la preservazione dell'informazione primaria. [3]

3.3.1 Criteri per la preservazione dei metadati

Basandosi su quanto detto prima, è indispensabile conservare solo una piccola parte dei metadati relativi ad una tabella, poiché solo alcuni di questi influiscono sull'effettivo risultato delle query di tipo SELECT. Per soddisfare unicamente questo requisito, i dati che è necessario conservare per ogni tabella sono solamente il nome e, per ogni sua colonna, il tipo ed il nome. In questo modo però verrebbero però perse informazioni potenzialmente molto utili, riguardanti la struttura e le relazioni fra i dati e le tabelle. Possiamo quindi considerare come un'aggiunta non indispensabile, ma molto gradita, la conservazione dei metadati riguardanti chiavi primarie, chiavi secondarie e chiavi esterne. Ci sono però molti altri oggetti SQL che non influiscono sui risultati di una query SELECT, come viste, vincoli CHECK, TRIGGERS e Stored Procedures. Tutti questi elementi sono di secondaria importanza per la preservazione del database: in particolare, servono per prevenire cambiamenti che porterebbero a problemi di consistenza, cambiamenti che non avvengono in un database archiviato per la sola lettura. Oltretutto tutti questi oggetti possono contenere porzioni di codice, che come già detto può portare a problemi di compatibilità. Infine, ci sono oggetti SQL che regolano i diritti di accesso e modificazione ai dati (USERS, ROLES, GRANTS). In situazione di archivio non è necessario mantenere queste restrizioni. [3]

3.3.2 Criteri per la conservazione di dati primari

La conservazione dei dati primari è la questione di importanza fondamentale: bisogna dimostrare che, dopo il processo di normalizzazione e de-normalizzazione, il risultato di qualsiasi query SELECT, eseguita sul database originale e su quello rigenerato, non presenti alcuna differenza. Innanzitutto bisogna assicurarsi che, per ogni tabella TO del database originale, composta da N colonne CO1, CO2, …, CON, esista una corrispondente tabella TR nel database rigenerato, composta dallo stesso numero di colonne CR1, CR2, … , CRN. Quindi si passa a verificare il contenuto di ogni cella di ogni tabella. Questa verifica può diventare complessa nel caso vi siano record nella stessa tabella con valori identici, evenienza possibile in alcune versioni di SQL

(SQL1999), oppure nel caso di tipi definiti dall'utente. L'idea di base è ben illustrata dalle seguenti tre query:

```
SELECT COUNT(*) FROM TO - - number of records of TO,
SELECT COUNT(*) FROM TR - - number of records of TR,
SELECT COUNT(*) FROM TO,TR
WHERE
TO.CO1 = TR.CR1 AND
TO.CO2 = TR.CR2 AND  ... TO.CON = TR.CRN
```

Il risultato di queste query dovrà essere identico se i dati sono stati preservati in modo corretto sulla base del significato del segno "=" in SQL. Ciò significa che non è indispensabile il mantenimento esatto del tipo del dato, basta essi siano confrontabile: ad esempio VARCHAR(10) e CHAR(5), oppure SMALLINT e INT. Questa condizione è problematica anche per quegli oggetti per cui, in SQL, non sia prevista la possibilità di confronto tramite l'operatore "=", come i BLOBS (Binary Large Objects), per cui bisognerà implementare una funzione di comparazione specifica. [3]

4 Il formato SIARD

A partire da questi principi è stato concepito il formato di normalizzazione per database relazionali SIARD (Software Independent Archiving of Relational Databases), sviluppato dagli Swiss Federal Archives. Esso è stato progettato per soddisfare una serie di requisiti basati sui criteri di preservazione illustrati prima:

- **Produrre un singolo file:** il formato SIARD deve immagazzinare un intero database in un singolo file. Questo perché, nel caso di formati che prevedono la conservazione di un file per ogni singola tabella, spesso si presentano gravi problemi di integrità referenziale.
- **Non conservare ciò che è inaccessibile:** un file SIARD deve contenere solamente informazioni che sarebbero accessibili, sul database originale, dall'utente usando query SQL, nulla di più e nulla di meno, in modo tale da poter accuratamente definire ciò che è accessibile al software normalizzatore tramite la creazione di appositi utenti con specifici privilegi.
- **Uno standard aperto:** SIARD deve essere uno standard completamente Open Source, per il mercato di nicchia che va a ricoprire, è indispensabile che esso sia accessibile liberamente da chiunque. Questo aspetto è fondamentale per qualunque formato per la preservazione a lungo termine.
- **Conservazione di dati primari e metadati:** ovviamente è imprescindibile la conservazione della totalità dei dati primari e la preservazione di quei metadati necessari al rispetto del criterio di completa preservazione.
- **Non avere dati di una istituzione specifica:** il formato SIARD deve essere del tutto indipendente dalle istituzioni che ne fanno uso, e non contenere metadati specifici dell'archivio che l'ha utilizzato. un archivio SIARD deve poter essere utilizzato allo stesso modo in qualunque istituto di archiviazione.
- **Conservazione delle chiavi:** esse non sono indispensabili per l'archivio dei database in quanto questi non vengono più modificati, ma sono di fondamentale importanza per capire la struttura del database. Per questo, esse vengono preservate solo dove possibile. [3]

4.1 Struttura di un file SIARD

Un file SIARD è basato interamente su standard aperti ed ampiamente utilizzati:

- ZIP per il container,
- SQL: 1999 per la struttura,
- XML per i dati,

- UTF-8 per i caratteri.

Il file consiste di due cartelle di livello più alto:

- **La cartella *header***, che contiene il file *metadata.xml*, che descrive in modo completo il database elencando tutti gli oggetti in un formato compatibile con SQL: 1999.
- **La cartella *content***, che contiene a sua volta una cartella per ogni schema che è stato archiviato. I nomi di tali cartelle sono codificati come *schema0, schema1, … , schemaN*, quindi i veri nomi degli schema sono salvati nel file *metadata.xml.*

All'interno di ogni cartella relativa ad uno schema è presente a sua volta una cartella per ogni tabella in esso contenuta.

Le cartelle *table:* anche le cartelle relative alle tabelle sono nominate con la stessa modalità delle cartelle schema (*table0, table1, … , tableN*). Ogni *table folder* contiene una definizione dello schema della tabella in XML, un file XML che contiene i dati primari contenuti nella tabella, e, se necessario, cartelle (*lob0, lob1, …, lobN*) per ogni colonna Large Object. I tipi dei dati sono mappati da SQL a tipi XML. Ogni dato, nel file XML è identificato da un'etichetta di riga ed una di colonna, che devono essere di dimensione contenuta.

Il problema dei BLOBs: gli oggetti binari, detti anche BLOBs (Binary Large Objects), presentano un problema particolare: potrebbero contenere qualunque cosa, potendo quindi non essere adatti alla conservazione a lungo termine. Questo fatto solitamente non crea problemi: si avrebbero semplicemente piccole porzioni del database che, passato un certo periodo di tempo, non sarebbero più comprensibili per la natura stessa dei dati. La conservazione dei BLOBs è quindi facoltativa: dipende dalla volontà di chi sta creando l'archivio, in relazione al loro tipo ed alla loro conformità alla preservazione.

Altri oggetti SQL:

- **Chiavi:** Anche se non sono strettamente necessarie, queste vengono, se possibile, conservate insieme ai metadati per ogni tabella.
- **Viste:** Le query sono un altro oggetto problematico in quanto possono contenere porzioni di codice. Il testo della vista può però essere utile per la comprensione dell'utilizzo del database. Quindi queste sono salvate sotto forma di testo della query, se possibile. Altrimenti viene conservata la lista di colonne di cui essa è composta

- **Vincoli CHECK e Triggers:** I vincoli check non sono necessari per i principi di conservazione e, dato che possono contenere codice, questi non vengono mantenuti durante il processo di normalizzazione. Stessa cosa si verifica per i trigger, anch'essi non preservati. [3]

4.2 SIARD Format: Hands On

Per testare sul campo l'effettivo funzionamento del formato SIARD, è stato usato il software SIARDGui, software realizzato dagli stessi sviluppatori del formato. Esso fornisce un'interfaccia grafica che permette di eseguire normalizzazioni e de-normalizzazioni di database ospitati sui DBMS più diffusi. In particolare, ne è stata usata la versione 2.1.134, la più recente disponibile al momento del testing. I DBMS supportati sono Microsoft Access, DB2, H2 Database, MySQL, Oracle, PostgreSQL ed SQL Server. SIARDGui è inoltre utile per accedere ai database archiviati leggendo il file SIARD corrispondente. I file generati dalle normalizzazioni sono quindi file *.siard*, che però sono semplici archivi ZIP, che si possono scompattare e a cui si può accedere senza dover necessariamente usare SIARDGui.

5 Testing di SIARD

La sperimentazione si è basata in particolare su quattro DBMS: SQL Server, MySQL, PostgreSQL e Oracle. Sono state analizzate le tre fasi in cui dovrebbe avvenire la preservazione di un database: normalizzazione, accesso al database normalizzato e de-normalizzazione, verificando le differenze fra:

- Database originale
- Database in forma normalizzata
- Database recuperato e de-normalizzato

È stata inoltre posta particolare attenzione sulle differenze di comportamento dovute all'uso di DBMS sorgente e destinazione diversi.

5.1 Testing di normalizzazione da SQL Server

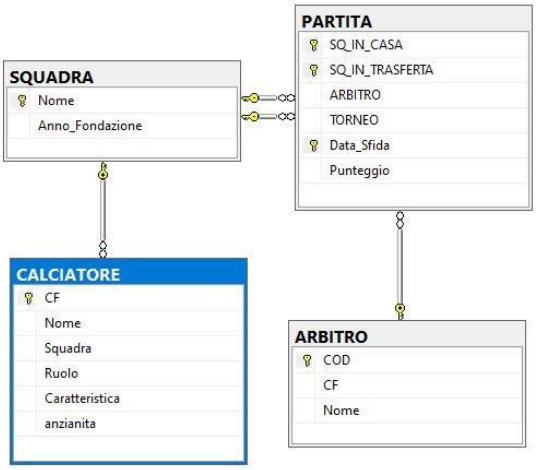
Sono stati fatti test con vari database di prova per verificare il funzionamento del formato SIARD riguardo il mantenimento della struttura e caratteristiche base dei database. In particolare, dato che il principio della conservazione tramite normalizzazione non garantisce la preservazione di tutte le sue componenti, la sperimentazione si è focalizzata su:

- Mantenimento di Primary Keys e Foreign Keys
- Mantenimento di Vincoli, in particolare Check e Not Null
- Tipi di dato "numerici" e di stringhe (in particolare mantenimento delle differenze, ad esempio, fra INT e SMALLINT, o fra VARCHAR e NVARCHAR)
- Conservazione delle Viste
- Preservazione dei Binary Large Objects

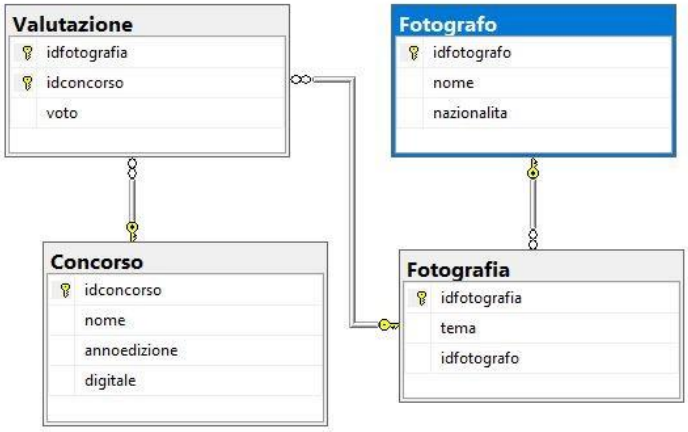
5.2 Database di prova:

Ho usato i seguenti Database per effettuare i test di funzionamento del sistema SIARD. Tali Database sono stati creati originariamente su SQL Server e sono stati scelti in modo tale da poter verificare tutte le componenti elencate precedentemente. Oltre a quelli elencati sotto, è stato usato anche AdventureWorks2019, database di prova fornito da Microsoft, per poter verificare il comportamento di SIARDGui per database più voluminosi e dalla struttura più complessa.

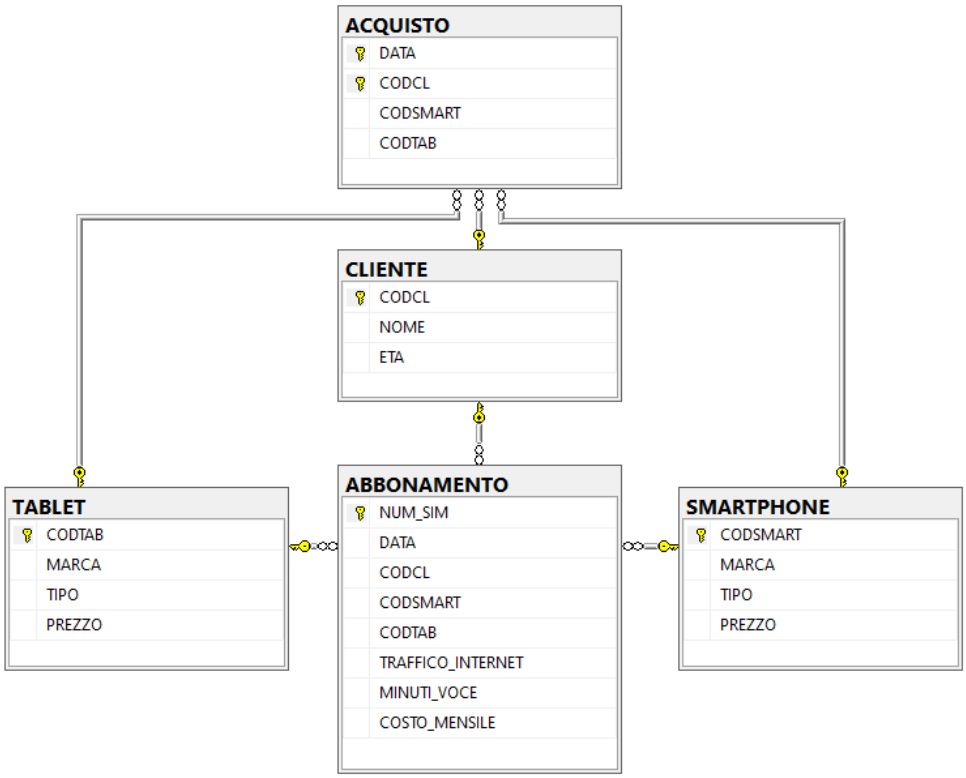
DB CALCIO:



DB CONCORSO FOTOGRAFICO:



DB SMARTPHONE:

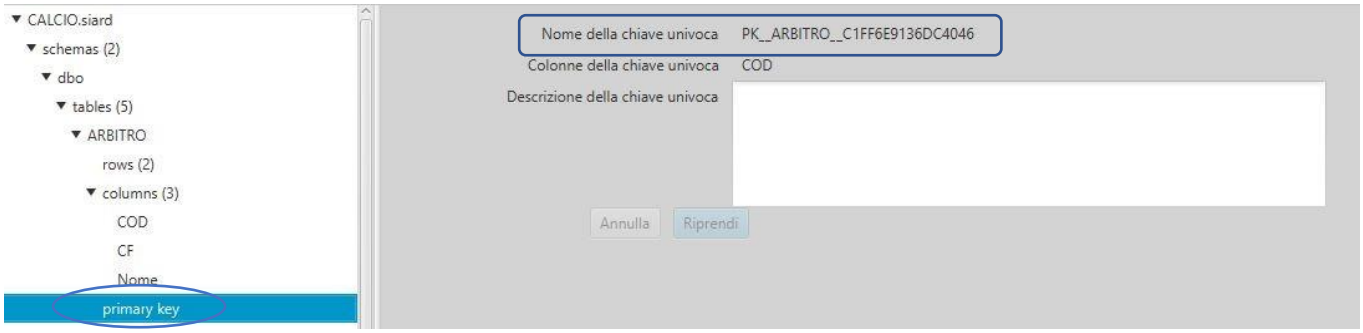


5.3 Test di normalizzazione

Per procedere alla normalizzazione ci si collega al DB con la stringa di connessione JDBC corrispondente al DBMS utilizzato, questa viene generata automaticamente da SIARDGui in base ai vari campi da specificare tramite interfaccia grafica per stabilire la connessione.

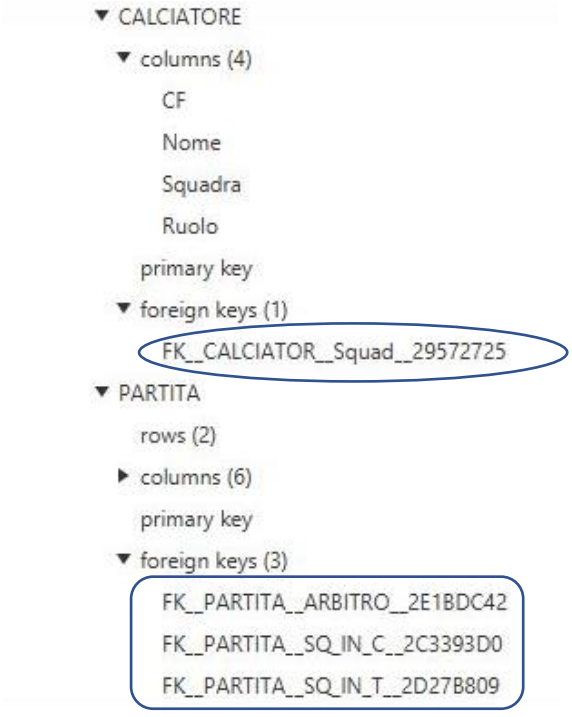
Come prima cosa ho provato ad importare tramite la Utility i Database sopra elencati per generare i file SIARD e vedere quali informazioni vengono mantenute tramite questo processo e quali vengono perse o modificate. Segue un elenco con i risultati ottenuti con relativi screenshots ricavati dal visualizzatore di file SIARD presente in SIARDGui.

- **Mantenute tutte le chiavi primarie:**



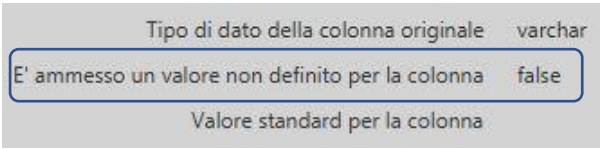
Qui si nota la presenza della Primary Key nella tabella Arbitro. Viene salvato anche il nome del vincolo associato ad essa.

Mantenute tutte le chiavi esterne:



Da questa schermata si può notare come siano mantenute, nel file SIARD, tutte le Foreign Key ed anche i nomi specifici dei vincoli corrispondenti.

Mantenuti i vincoli di NOT NULL, mentre non c'è traccia dei vincoli CHECK:

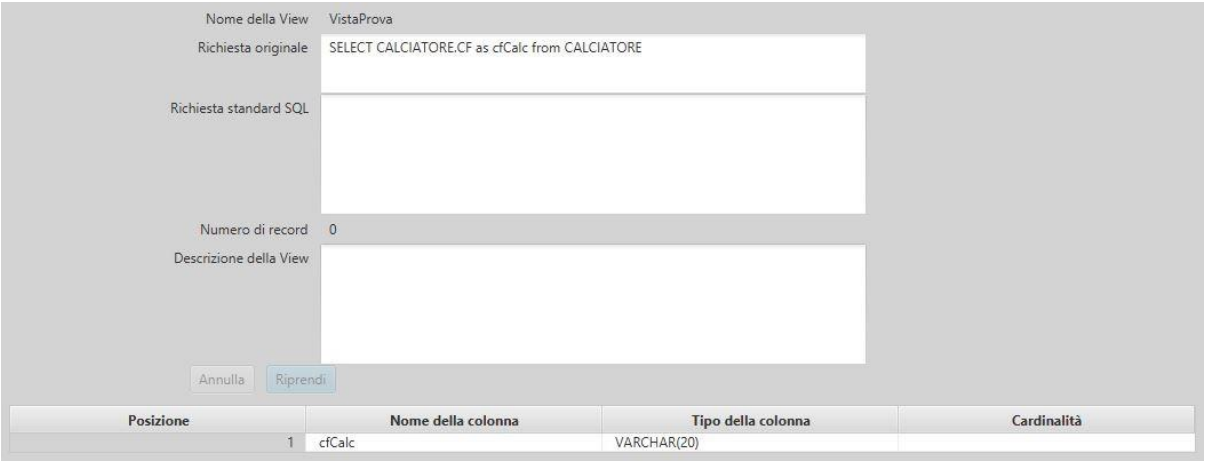


Qui si evidenza la non ammissibilità dei valori NULL per la colonna selezionata, i vincoli CHECK non sono contemplati.

Mantenuti i tipi di dato numerici e di stringa:

Come esempio viene proposta la differenza fra una colonna di tipo VARCHAR(30) e una VARCHAR(20). Questo vale anche per i tipi numerici. Tipi specifici adottati da un certo DBMS vengono però "standardizzati", come si vedrà in un esempio di seguito.

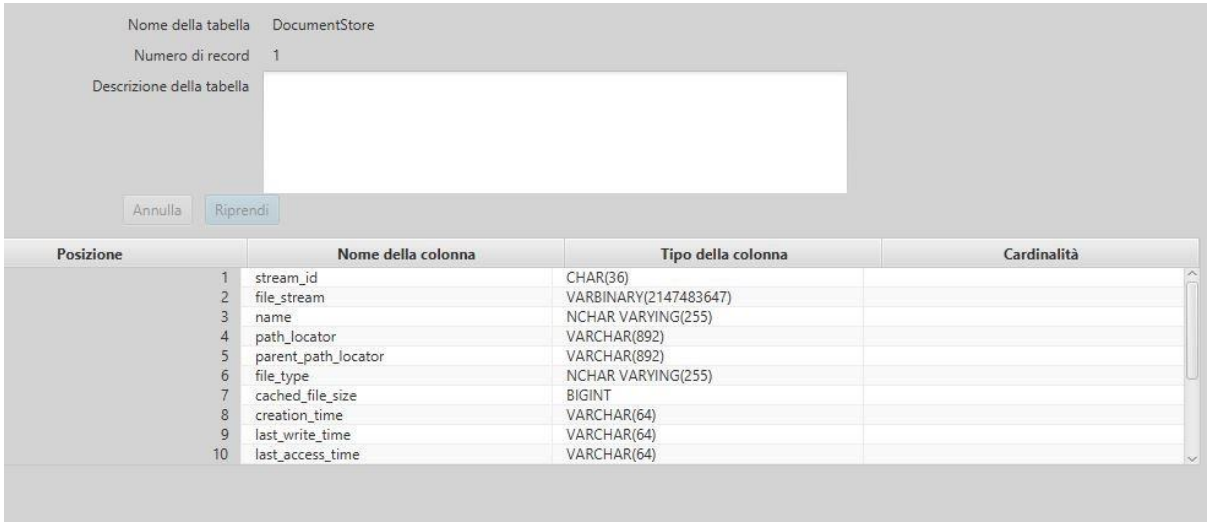
Mantenute le viste in quanto: colonne di cui sono composte, query che le genera e breve descrizione:



Le query sono quindi mantenute in modo statico, in modo tale che, se il fruitore dell'archivio ne avesse la necessità, potrebbe generare nuovamente la query usando le informazioni qui mantenute.

Cosa succede ai BLOBs?

Per verificare invece il comportamento riguardo ai BLOBs, SQL Server è orientato all'uso delle File Tables, quindi ho creato un Database con una File Table che contenesse qualche file. Con questa soluzione i BLOBs sono file mantenuti all'interno del File system, ed il DataBase ne conserva solo i metadati trattandoli come oggetti "condivisi" con il sistema operativo. Come prevedibile, essendo questa una funzione specifica di SQL Server, le File Tables sono salvate come tabelle normali, sono mantenuti quindi i metadati relativi ai file e non i file stessi.



Posizione	Nome della colonna	Tipo della colonna	Cardinalità
1	stream_id	CHAR(36)	
2	file_stream	VARBINARY(2147483647)	
3	name	NCHAR VARYING(255)	
4	path_locator	VARCHAR(892)	
5	parent_path_locator	VARCHAR(892)	
6	file_type	NCHAR VARYING(255)	
7	cached_file_size	BIGINT	
8	creation_time	VARCHAR(64)	
9	last_write_time	VARCHAR(64)	
10	last_access_time	VARCHAR(64)	

Da questo specifico test si può anche notare come certi tipi di dato particolari vengano lasciati invariati ma sono invece mappati al tipo XML che più si addice all'originale: ad esempio HierarchyID viene trasformato in VARCHAR(892).

5.4 Esportazione da file SIARD

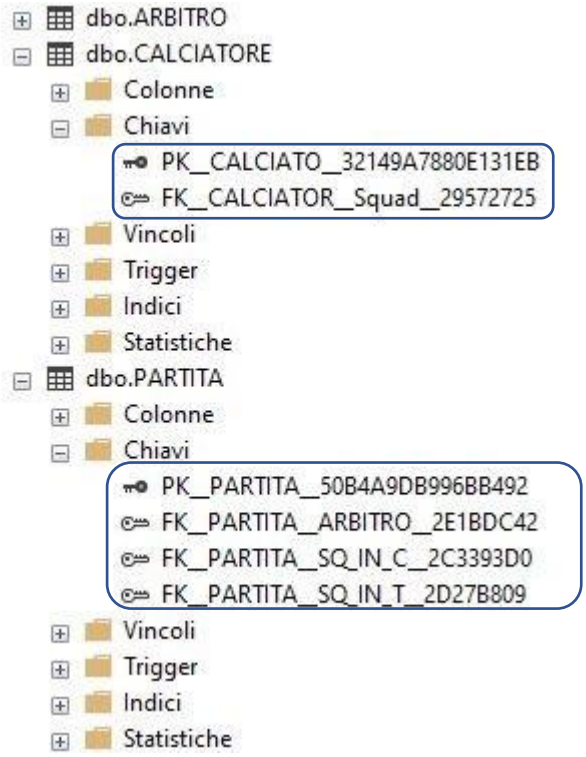
Il Client SiardGui permette anche di esportare i file SIARD come database sui DBMS supportati. I seguenti test sono stati fatti per evidenziare le differenze che il Database generato a partire dal file SIARD ha rispetto al DB originale. Per fare questi test sono stati usati Database provenienti da SQL Server e sono stati esportati su DBMS diversi per tentare di evidenziare gli eventuali problemi di compatibilità che possono sorgere da queste conversioni, prestando attenzione agli elementi elencati prima.

5.5 Test di de-normalizzazione

Per prima cosa sono state verificate le caratteristiche dei database de-normalizzati su DBMS uguale a quello di partenza: SQLServer. Segue un elenco dei risultati ottenuti con relativi screenshot provenienti da SQLServer Management Studio.

Destinazione: SQL Server

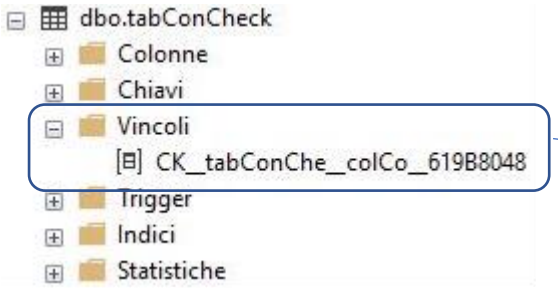
Anche dopo la de-normalizzazione, il database creato mantiene le chiavi primarie ed esterne



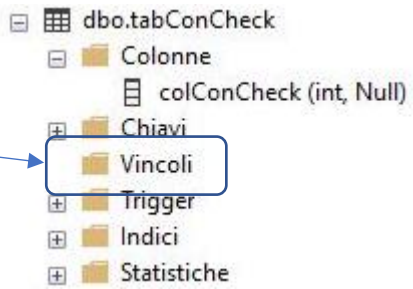
Da notare, anche qui la presenza, nel database de-normalizzato, di tutte le Primary Keys e le Foreign Keys. È mantenuto anche il nome originale dei vincoli.

Mantenuti i vincoli NOT NULL, ma non i CHECK

Database originale:



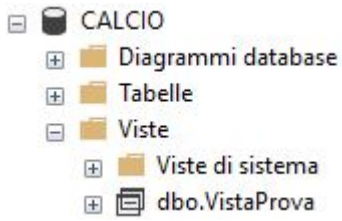
Database dopo la de-normalizzazione



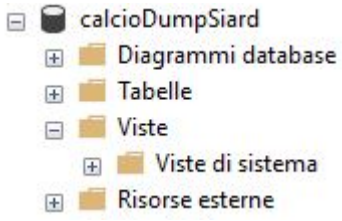
La cartella “Vincoli” del database de-normalizzato è vuota, a differenza di quella nel database originale

Inoltre: Conservati anche i tipi di dato numerici e di stringa. Le viste, invece, non sono presenti nel database rigenerato:

DB originale:



DB dopo la de-normalizzazione:

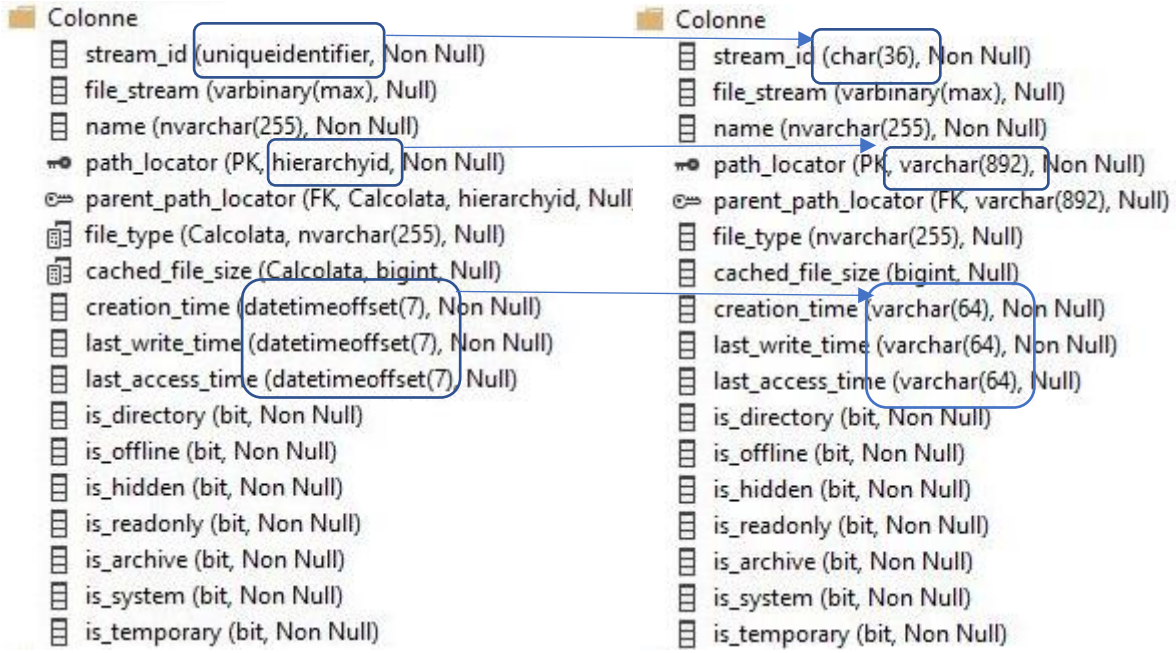


La cartella “Viste” è vuota nel database generato dalla de-normalizzazione, a differenza del database originale.

Per quanto riguarda le File Tables: nel database rigenerato queste sono trasformate in semplici tabelle che mantengono i metadati dei file che erano presenti al momento in cui è stata effettuata la normalizzazione, ma, come già visto, alcune colonne non mantengono il loro tipo originale:

DB originale:

DB dopo la de-normalizzazione:



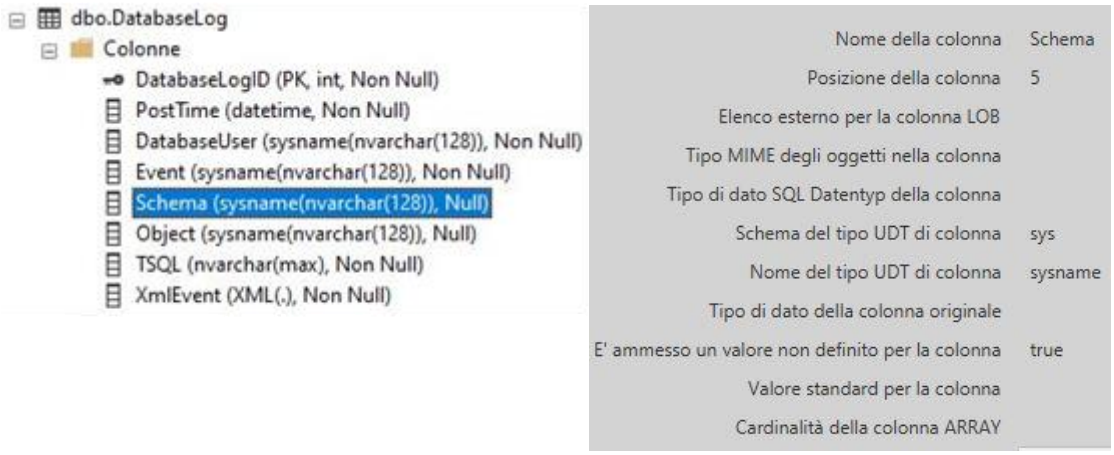
Si evidenzia come i tipi di dato del database originale non vengano sempre mantenuti così come sono nel database de-normalizzato, ma vengano “standardizzati”.

Test di de-normalizzazione del database AdventureWorks2019:

Per fare test più approfonditi è stato utilizzato il database di prova AdventureWorks2019; tuttavia, quando provo a fare l’upload del suo file SIARD nuovo a SQL Server questo non va a buon fine e viene generato il seguente errore:

java.sql.SQLException: MSSQL exception!< com.microsoft.sqlserver.jdbc.SQLServerException: Non è possibile inserire il valore NULL nella colonna 'Schema' della tabella 'awdump.dbo.DatabaseLog'. La colonna non ammette valori NULL. INSERT avrà esito negativo.

L'errore fa riferimento all'impossibilità di inserire valori NULL nella colonna "SCHEMA" della tabella "awdump.dbo.DatabaseLog", che però, sia nel database originale che nel file SIARD, i valori NULL li ammette eccome:



Non mi spiego quindi la comparsa di questo errore. Provando però a rimuovere dal database di partenza i valori NULL da quella tabella, la conversione va a buon fine e si può notare come anche nel database rigenerato sia corretta la presenza o meno di vincoli NOT NULL in ogni colonna, tranne in quella che provocava l'errore, in cui compare erroneamente tale vincolo.

Ricapitolando quanto detto finora:

Operazione	***Dopo la normalizzazione***	***Dopo la de-normalizzazione***
Primary Keys e Foreign Keys	Mantenute	Mantenute
Vincoli Not-Null	Mantenuti	Mantenuti
Tipi di dato "string"	Mantenuti	Mantenuti
Tipi di dato "numerici"	Mantenuti	Mantenuti
Viste	Mantenute le query che le generano, colonne di cui sono composte e breve descrizione	NON Mantenute

BLOBs in FileTables	Mantenuti solo metadati dei file	Mantenuti solo metadati dei file, non più riconosciuta la tabella come File Table
Vincoli Check	NON Mantenuti	NON Mantenuti

5.6 De-normalizzazione: diverse destinazioni.

Per verificare l'effettiva compatibilità del formato con i vari DBMS sono stati effettuati gli stessi test anche con DBMS di destinazione differenti.

5.6.1 Destinazione: MySQL

Segue l'elenco dei risultati ottenuti e screenshot realizzati su Heidi SQL, tool per la connessione e la gestione di database compatibile con vari DBMS, fra cui appunto MySQL e PostgreSQL

Anche qui, dopo l'esportazione, sono mantenute chiavi primarie ed esterne Vincoli NOT NULL mantenuti, vincoli CHECK non presenti e viste non mantenute:

Qui sono elencate le Foreign Keys della tabella, che corrispondono alle originali.

Qui sono evidenziati i tipi di dato, che nel caso di stringhe e numeri sono coerenti con gli originali.

Da questa colonna si può notare quali dati ammettano valori NULL e quali no, rispettando vincoli uguali a quelli presenti nel database originale.

Riscontrato problema di compatibilità riguardo i tipi VARCHAR e NVARCHAR:

In MySQL, il limite per la lunghezza dei dati di tipo VARCHAR è di 256 caratteri, essendo questo limite posto ad 8000 caratteri in SQL Server, insorgono problemi di compatibilità quando nel database sorgente ci sono colonne di tipo VARCHAR con lunghezza massima maggiore di 256 caratteri. Una tale colonna viene convertita in un campo TEXT, che però non può essere usato come Primary Key in quanto di lunghezza variabile. In questo caso viene generato il seguente errore e la conversione non va a buon fine:

java.sql.SQLSyntaxErrorException: BLOB/TEXT column 'DocumentNode' used in key specification without a key length

Questo codice di errore in particolare è stato riscontrato nel tentativo di conversione di una FileTable, in cui la chiave primaria '*DocumentNode*' è, nel file SIARD, di tipo VARCHAR(892). Per questo motivo non è stato possibile l'upload di un DB con FileTables al suo interno. Lo stesso problema è stato riscontrato anche nel tentativo di upload del DB *AdventureWorks2019*, che non va a buon fine per lo stesso motivo.

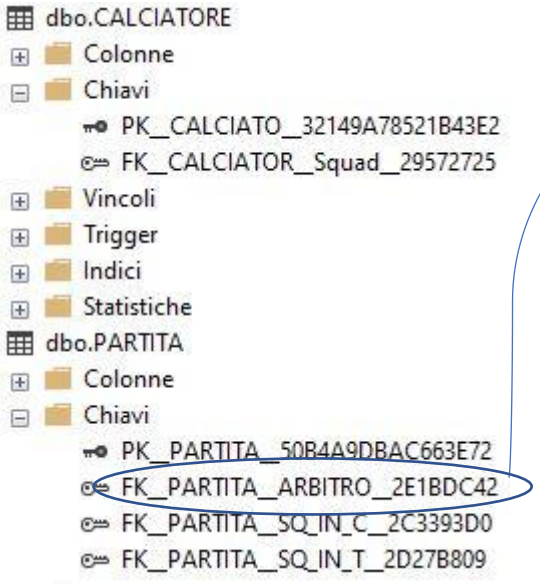
5.6.2 Destinazione Oracle

Problemi nell'esportazione di Foreign Keys: Riscontrato comportamento anomalo riguardante le Foreign Keys: alcune di queste sono mantenute nel DB esportato mentre altre non sono presenti. Elenco di seguito i test che ho fatto a riguardo, evidenziando il confronto fra il DB iniziale e quello risultato dall'esportazione su DBMS Oracle, con screenshot eseguiti su Oracle SQL Developer, programma usato per l'accesso e la gestione dei database Oracle.

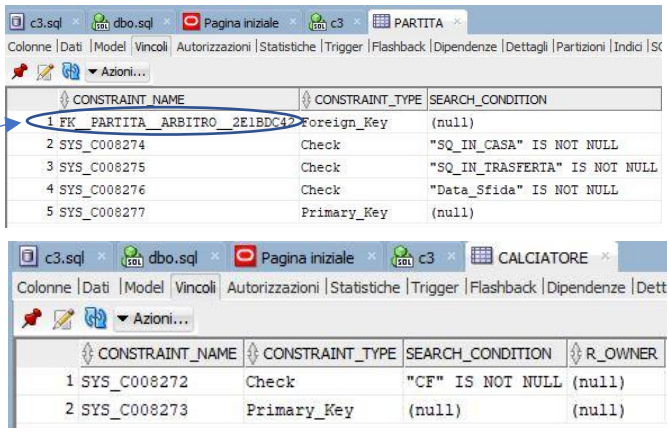
DB CALCIO:

Delle 3 Foreign Keys presenti nel Database originale, soltanto una è mantenuta

DB ORIGINALE:



DB ESPORTATO SU ORACLE:



c3.sql | dbo.sql | Pagina iniziale | c3 | PARTITA

Colonne | Dati | Model | Vincoli | Autorizzazioni | Statistiche | Trigger | Flashback | Dipendenze | Dettagli | Partizioni | Indici

Azioni…

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	FK__PARTITA__ARBITRO__2E1BDC42	Foreign_Key	(null)
2	SYS_C008274	Check	"SQ_IN_CASA" IS NOT NULL
3	SYS_C008275	Check	"SQ_IN_TRASFERTA" IS NOT NULL
4	SYS_C008276	Check	"Data_Sfida" IS NOT NULL
5	SYS_C008277	Primary_Key	(null)

c3.sql | dbo.sql | Pagina iniziale | c3 | CALCIATORE

Colonne | Dati | Model | Vincoli | Autorizzazioni | Statistiche | Trigger | Flashback | Dipendenze | Dett

Azioni…

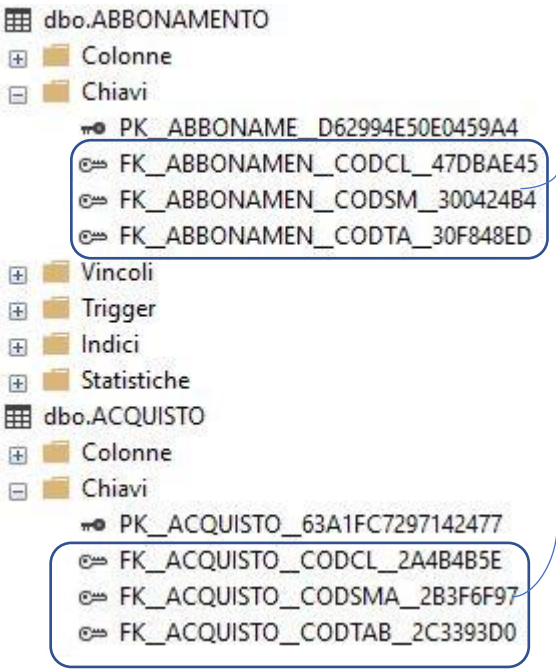
	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1	SYS_C008272	Check	"CF" IS NOT NULL	(null)
2	SYS_C008273	Primary_Key	(null)	(null)

Da qui si nota come sia presente solo una delle tre Foreign Keys nella tabella PARTITA, mentre nessuna è stata mantenuta nella tabella CALCIATORE.

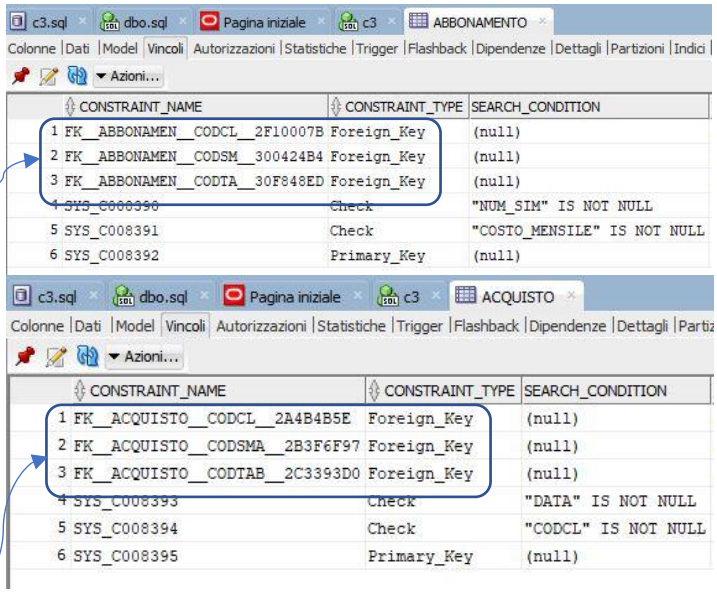
DB SMARTPHONE:

Stavolta tutte le Foreign Keys sono mantenute sul DB esportato su Oracle.

DB ORIGINALE:



DB ESPORTATO SU ORACLE



c3.sql | dbo.sql | Pagina iniziale | c3 | ABBONAMENTO

Colonne | Dati | Model | Vincoli | Autorizzazioni | Statistiche | Trigger | Flashback | Dipendenze | Dettagli | Partizioni | Indici

Azioni…

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	FK__ABBONAMEN__CODCL__2F10007B	Foreign_Key	(null)
2	FK__ABBONAMEN__CODSM__300424B4	Foreign_Key	(null)
3	FK__ABBONAMEN__CODTA__30F848ED	Foreign_Key	(null)
4	SYS_C008390	Check	"NUM_SIM" IS NOT NULL
5	SYS_C008391	Check	"COSTO_MENSILE" IS NOT NULL
6	SYS_C008392	Primary_Key	(null)

c3.sql | dbo.sql | Pagina iniziale | c3 | ACQUISTO

Colonne | Dati | Model | Vincoli | Autorizzazioni | Statistiche | Trigger | Flashback | Dipendenze | Dettagli | Partiz

Azioni…

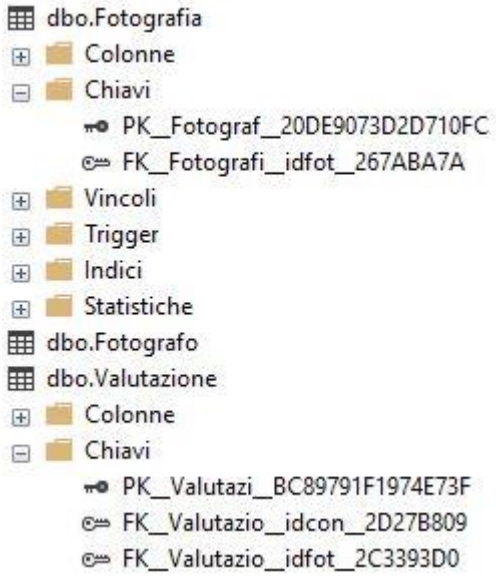
	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	FK__ACQUISTO__CODCL__2A4B4B5E	Foreign_Key	(null)
2	FK__ACQUISTO__CODSMA__2B3F6F97	Foreign_Key	(null)
3	FK__ACQUISTO__CODTAB__2C3393D0	Foreign_Key	(null)
4	SYS_C008393	Check	"DATA" IS NOT NULL
5	SYS_C008394	Check	"CODCL" IS NOT NULL
6	SYS_C008395	Primary_Key	(null)

Sono mantenute tutte le FK del Database originale.

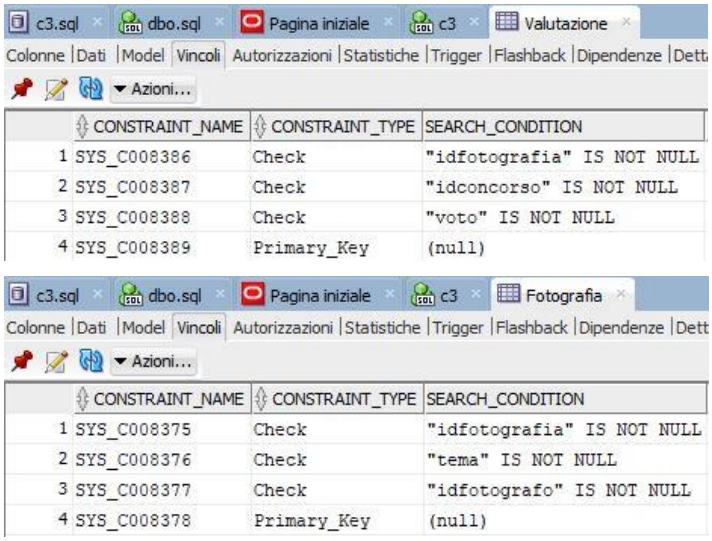
DB CONCORSO FOTOGRAFICO:

In questo caso nessuna delle Foreign Keys viene mantenuta nel DB esportato.

DB ORIGINALE:



DB ESPORTATO SU ORACLE:



Non è presente nessuna Foreign Key fra i vincoli del DB esportato su Oracle.

Anche qui problemi per quanto riguarda tipi VARCHAR ed NVARCHAR

Riscontrato problema analogo a quanto avveniva su MySql, infatti anche qui sono diversi i limiti di lunghezza dei tipi VARCHAR ed NVARCHAR rispetto a SQLServer. In particolare questi limiti sono di 4000 caratteri per i VARCHAR2 (tipo analogo a VARCHAR di SQLServer) e 2000 caratteri per il tipo NVARCHAR2 (analogo a NVARCHAR in SQLServer). Se nel DB sorgente sono presenti colonne che violano questi limiti l'upload del DB non va a buon fine e si genera l'errore:

ORA-00910: specified length too long for its datatype

Inoltre: Non mantenute le viste ed i vincoli CHECK: in alcuni screenshot tratti da Oracle Developer si può notare come siano comunque presenti vincoli CHECK, ma in realtà sono sempre di tipo NOT NULL. Mantenute Primary Keys, vincoli di NOT NULL e differenze fra i tipi di dato numerici

5.6.2 Destinazione PostgreSQL

Anche qui problemi riguardo le Foreign Keys

Riscontrato problema analogo a quanto avviene in Oracle: non tutte le Foreign Keys sono mantenute nei Database esportati. In particolare, però, le chiavi esterne mantenute sono diverse da quelle mantenute in Oracle:

- Non mantenute FK del DB CALCIO
- Mantenute le FK del DB SMARTPHONE e del DB CONCORSO FOTOGRAFICO

Nessun problema riguardo VARCHAR ed NVARCHAR:

Su PostgreSQL, sia i tipi di dato VARCHAR che NVARCHAR vengono mappati nel tipo VARCHAR, che, non avendo un limite di massima lunghezza, non provoca incompatibilità nella fase di esportazione.

Errore nell'upload del DB AdventureWorks2019:

Anche in questo caso non è stato possibile portare a termine l'upload del DB AdventureWorks2019, stavolta a causa di un non meglio precisato errore di sintassi:

org.postgresql.util.PSQLException: ERRORE: errore di sintassi a o presso (" Posizione: 166;

Per il resto, comportamento analogo ai DBMS di destinazione analizzati in precedenza: Mantenute le Primary Keys, i vincoli NOT NULL e le differenze fra i tipi di dato numerici. Non mantenute viste e vincoli CHECK.

Ricapitolando i risultati ottenuti:

Operazione	***MySQL***	***Oracle***	***PostgreSQL***
Primary Keys	Mantenute	Mantenute	Mantenute
Foreign Keys	Mantenute	Mantenute in parte	Mantenute in parte
Vincoli NOT NULL	Mantenuti	Mantenuti	Mantenuti
Tipi di dato "string"	Problemi con lunghezze massime dei dati VARCHAR	Problemi con lunghezze massime dei dati VARCHAR	Mantenuti, ma non le differenze fra VARCHAR e NVARCHAR

Tipi di dato “numerici”	Mantenuti	Mantenuti	Mantenuti
Vincoli CHECK	NON mantenuti	NON mantenuti	NON mantenuti
Viste	NON mantenute	NON mantenute	NON mantenute
File Tables	NON mantenute	NON mantenute	Mantenuti solo metadati dei file

5.7 Conclusioni sul formato SIARD

Visti i risultati dei test effettuati si può concludere che il sistema di normalizzazione SIARD, usato tramite il corrispettivo software di conversione SIARDGui, rispetti in toto le caratteristiche prescritte dai criteri di conservazione di database. Sono però state riscontrate criticità per quanto riguarda il tool SIARDGui, che non risolve le incompatibilità presenti fra i DBMS sorgente e destinazione, che invece dovrebbero essere del tutto irrilevanti una volta eseguita la normalizzazione. Questo rende quindi impossibile la de-normalizzazione di alcuni file SIARD su specifici DBMS.

6 Sviluppo di DBComparer

Eseguire confronti fra database:

Per portare a termine la sperimentazione si è reso necessario l'utilizzo di uno strumento automatizzato per la comparazione di database. Per far fronte a questa esigenza è stato sviluppato un software ad-hoc per confrontare, in particolare, i risultati di query eseguite su diversi database.

6.1 Software di comparazione di database: DBComparer

Scopo

Lo scopo del software è quello di avere un tool per confrontare in modo veloce due database, in particolare con l'obiettivo di verificare eventuali differenze nei risultati di specifiche query select. Esso è stato creato per verificare il rispetto dei principi di normalizzazione del formato SIARD.

6.2 Descrizione generale

Il software, realizzato in java, si avvia da linea di comando ed accetta due parametri: il primo sarà il nome del file di input, che dovrà essere formattato in JSON e conterrà i parametri necessari alla connessione ai database da confrontare, le query da usare per i confronti e un'indicazione di quale sia il database di riferimento, mentre il secondo sarà il nome del file di output, in cui verranno scritti i risultati. Il confronto viene fatto a partire da un database di riferimento, indicato nel file di input, e tutte le differenze trovate negli altri database saranno relative a quest'ultimo.

6.3 Requisiti:

Parsing File di input:

Il software, aperto il file di input dovrà effettuarne il parsing per generare gli oggetti necessari per la creazione di tutte le connessioni e l'esecuzione delle query in esso specificate.

Connessione ai Database:

Il software deve connettersi correttamente a database di diverse tipologie, in particolare supportare MSSQL, MYSQL, PostgreSQL e Oracle. Per farlo vanno specificati nel file JSON di input i seguenti parametri:

- Tipo del database: Stringa che indica il DBMS che ospita il database, per poter caricare il driver JDBC corretto.
- Host
- Port

- Username e password
- Nome del database
- Schema: è necessario specificare lo schema solo nel caso di PostgreSQL ed Oracle.

Confronto fra Database:

Il software dovrà quindi riconoscere le differenze nei risultati delle query elencate nel file di input. Tutte le query elencate sono eseguite su tutti i Database descritti nel file. In particolare, vengono rilevati:

- Diverso numero di colonne:
- Diverso nome delle colonne
- Diverso tipo delle colonne
- Diverso numero di righe
- Diverso contenuto nei record, specificando in particolare numero di riga e nome della colonna in cui avviene l'incongruenza.

Inoltre, in caso di diverso numero delle colonne non sarà possibile analizzarne il contenuto. Vengono analizzati i dati presenti all'interno delle colonne anche nel caso in cui queste siano di tipo diverso: non è raro che su DBMS differenti, tipi fra loro compatibili abbiano nome diverso, come ad esempio i tipi INT (in MSSQL) e NUMERIC (in Oracle). I confronti sulla base del nome delle colonne sono da effettuare in maniera case-insensitive, in modo da ridurre la dimensione del file di output evitando di mostrare informazioni irrilevanti.

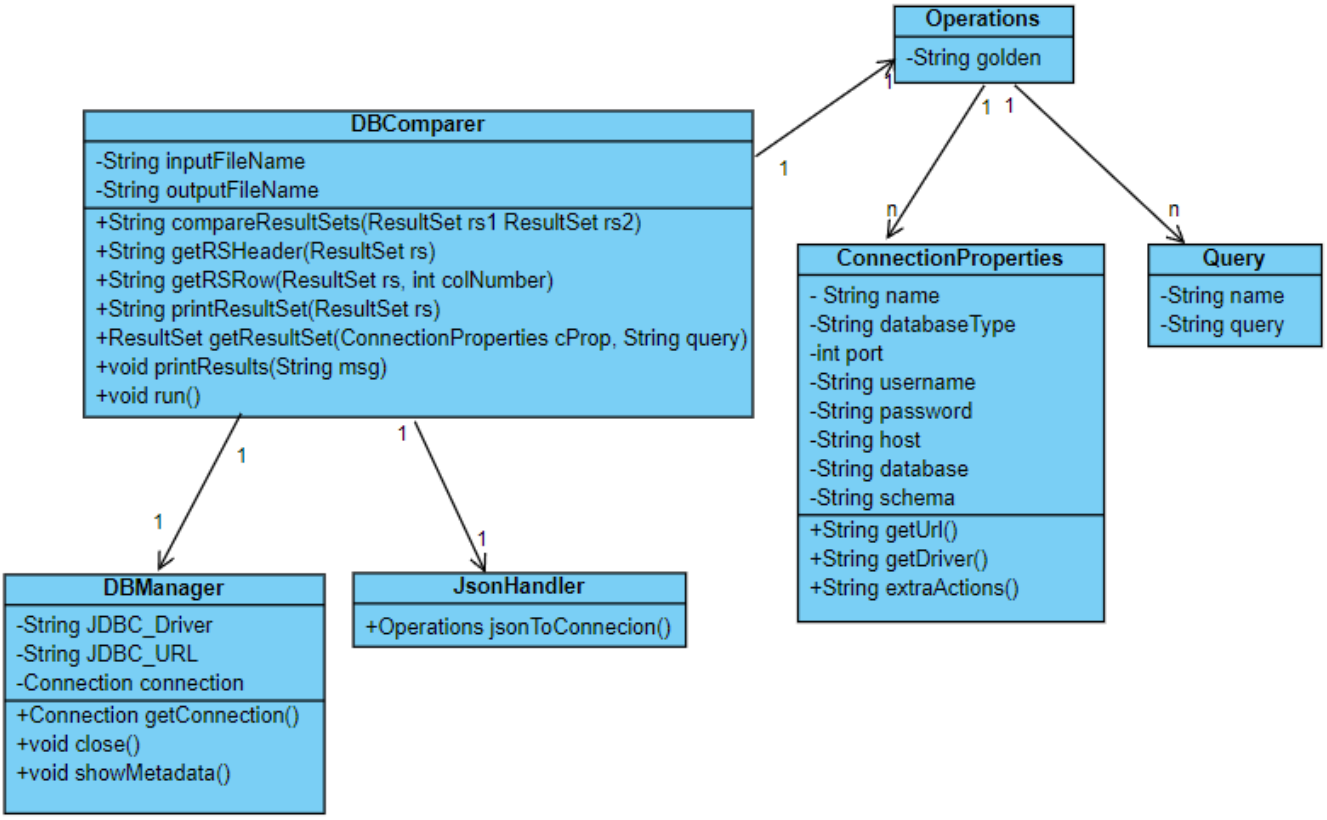
Creazione file di output:

L'output del file sarà quindi presentato in forma testuale su un file specificato dall'utente, usando un formato personalizzato in cui si mostrano prima le eventuali differenze a livello di schema, quindi quelle riscontrate nel contenuto vero e proprio, indicando anche il numero di riga ed il nome di colonna in cui si verificano.

6.4 Design

Struttura generale:

Di seguito un Class Diagram UML per descrivere la struttura generale del software e le classi di cui è composto:

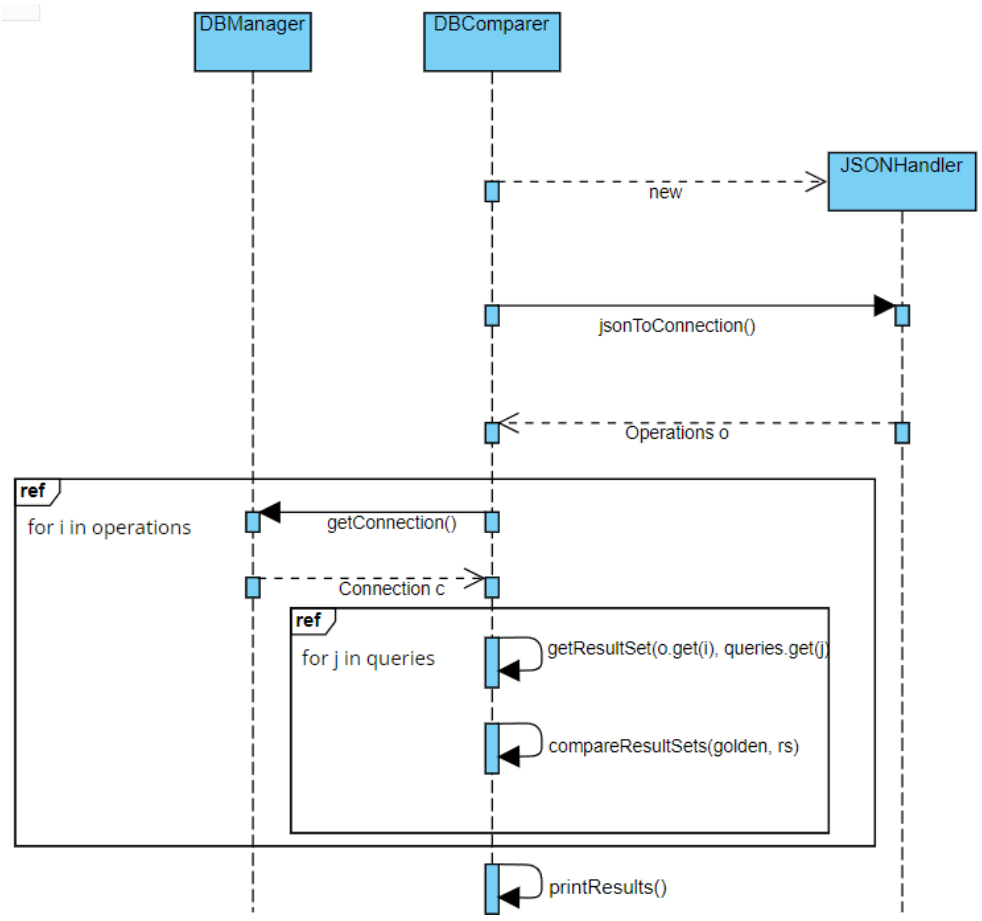


Possiamo in particolare notare le classi:

- **Operations**: contiene la lista di database a cui è richiesto connettersi e le query da eseguire
- **ConnectionProperties**: definisce tutti i parametri necessari ad instaurare una connessione ad un DBMS
- **Query**: per le proprietà di ogni query da eseguire
- **DBComparer**: classe principale che contiene i metodi per il confronto dei ResultSet generati dalle query eseguite sui database
- **DBManager**: classe utility che gestisce l'apertura e la chiusura delle connessioni ai database, nonché l'esecuzione delle query.
- **JsonHandler**: gestisce il parsing del file JSON di input

Funzionamento:

Di seguito un Sequence Diagram UML che descrive la dinamica di funzionamento del software:



Dato un file di input, questo viene aperto e da passato come parametro al metodo jsonToConnection della classe JsonHandler appena istanziata, questa effettua il parsing del file e ritorna un oggetto di tipo Operations, che contiene le proprietà di tutte le connessioni da stabilire e delle query da eseguire. Quindi, per ogni database e per ogni query elencata, DBComparer effettua la connessione (tramite DBManager), esegue le query ottenendo i ResultSet, ed infine li confronta con il database di riferimento. Il risultato di ogni confronto è una stringa che viene quindi scritta sul file di output designato.

Esempio di un file di output:

```
OUTPUT:
Comparing server microsoft and server mysql using query: query1
Columns: ok. Values:
row 1 is different: DATA: 2019-08-15 != 2019-08-14
row 2 is different: DATA: 2019-08-15 != 2019-08-14

Comparing server microsoft and server mysql using query: query2
Columns: ok. Values:
row 3 is different: DATA: 2020-04-15 != 2020-04-14
row 4 is different: DATA: 2020-04-15 != 2020-04-14
row 5 is different: DATA: 2020-05-11 != 2020-05-10
row 6 is different: DATA: 2020-05-11 != 2020-05-10
Comparing server microsoft and server postgres using query: query1
Different column type! column 8: reference money != numeric
Values: ok.

Comparing server microsoft and server postgres using query: query2
Everything Ok
Comparing server microsoft and server oracle using query: query1
Different column type! column 1: reference int != NUMBER
Different column type! column 2: reference date != DATE
Different column type! column 3: reference int != NUMBER
Different column type! column 4: reference int != NUMBER
Different column type! column 5: reference int != NUMBER
Different column type! column 6: reference int != NUMBER
Different column type! column 7: reference int != NUMBER
Different column type! column 8: reference money != NUMBER
row 1 is different: COSTO_MENSILE: 15.0000 != 15
row 2 is different: COSTO_MENSILE: 30.0000 != 30
row 3 is different: COSTO_MENSILE: 20.0000 != 20
row 4 is different: COSTO_MENSILE: 20.0000 != 20
row 5 is different: COSTO_MENSILE: 20.0000 != 20

Comparing server microsoft and server oracle using query: query2
Different column type! column 1: reference date != DATE
Different column type! column 2: reference int != NUMBER
Different column type! column 3: reference int != NUMBER
Different column type! column 4: reference int != NUMBER
Values: ok.
```

Occorre notare come pur essendo rilevate differenze nei tipi e nei valori, queste possano essere comunque del tutto irrilevanti a fini pratici, come ad esempio la differenza fra i tipi INT di SQLServer e NUMBER di Oracle, che viene riportata ma non crea incompatibilità di alcun tipo.

Conclusioni

Il tema della preservazione dei dati a lungo termine è complesso e non presenta una soluzione univoca che sia adatta ad ogni situazione. SIARD si pone come un sistema open source che vuole implementare i principi della conservazione per normalizzazione, che, per quanto riguarda l'ambito dei database relazionali, pare essere la scelta più adatta. È stato verificato come il formato SIARD rispetti tutti i principi della normalizzazione, ma anche che il software SIARDGui presenti numerosi problemi ed imperfezioni, che rendono necessaria la verifica delle caratteristiche dei database creati tramite software esterni. Proprio con questa esigenza in mente è stato sviluppato DBComparer, che effettua tali verifiche in modo autonomo.

Questa tesi è stata realizzata al termine di un'attività progettuale durante la quale penso di aver acquisito nuove competenze. In particolare

- L'utilizzo di sistemi DBMS diversi da SQLServer, con le relative sintassi SQL e le diverse specifiche di connessione, in particolare DBMS Oracle, PostgreSQL e MySQL.
- L'utilizzo di software di gestione di tali DBMS, in particolare HeidiSQL, Oracle SQL Developer e MySQL Workbench.
- L'utilizzo del tool SIARDGui per la normalizzazione e de-normalizzazione di database, con relativi parametri ed opzioni.
- L'utilizzo di librerie Java per il parsing di file JSON.
- L'uso delle librerie JDBC, con driver per SQLServer, MySQL, Oracle e PostgreSQL, con statement e classi ad essi relativi.
- La redazione di report per documentare il lavoro svolto ed i risultati ottenuti
- Più in generale, affrontare problemi al di fuori dei canoni a cui sono abituato, cercando individualmente la strada per risolverli in modo del tutto autonomo, effettuando ricerche e tentando varie soluzioni, con l'obiettivo di portare a termine il compito richiesto nei tempi previsti.

Appendice

Indicazioni sui software utilizzati durante la sperimentazione:

Durante il corso dell'attività progettuale sono stati utilizzati i seguenti software su sistema operativo Windows10.

- SIARD Suite ver. 2.1.34: interfaccia grafica che permette la visualizzazione di file *.siard,* disponibile al link: https://github.com/sfasiard/SiardGui/releases/download/2.1.34/SIARD-Suite-2.1.134.zip
- SQLServer: software per la gestione di database MSSQL: https://www.microsoft.com/it-it/sql-server/sql-server-downloads
- MySQL Workbench: software per la gestione di database MySQL: https://dev.mysql.com/downloads/workbench/
- HeidiSQL: software per la gestione di database di vari DBMS: https://www.heidisql.com/download.php
- Oracle SQL developer: software per la gestione di database Oracle: https://www.oracle.com/tools/downloads/sqldev-downloads.html
- IntelliJ IDEA Community edition: ambiente di sviluppo integrato per linguaggio Java: https://www.jetbrains.com/idea/download/#section=windows
- Repository GitHub del software DBComparer https://github.com/Bucci23/dbComparer

Bibliografia

[1] Uwe M. Borghoff Peter RödigJan Scheffczyk Lothar Schmitz, Long-Term Preservation of Digital Documents Principles and Practices

[2] Heiko Müller, University of Edinburgh, Database Archiving

[3] Dr. sc. math. Hartwig Thomas, SIARD Criterion: http://www.enterag.ch/hartwig/SIARD_Criterion.pdf