UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Matematiche, Fisiche ed Informatiche
Corso di Laurea in Informatica

Analisi e Sviluppo di una Base di Dati per la Gestione dei Contributi Ambientali di Moduli Fotovoltaici

Laureando
Marco Terzulli

Relatore
Prof. Riccardo Martoglia

Anno Accademico 2019/20

RINGRAZIAMENTI

Ringrazio l'Ing. Riccardo Martoglia, mio relatore, per la grande e continua disponibilità e per l'aiuto nell'arco del mio percorso universitario.

Ringrazio Multitraccia per avermi dato la possibilità di sviluppare il presente progetto durante il mio tirocinio.

Ringrazio i miei compagni di corso per il supporto nello studio e nella preparazione degli esami.

Infine, un ringraziamento speciale va alla mia famiglia ed ai miei amici che mi hanno sempre supportato e spronato a dare il meglio.

PAROLE CHIAVE

Database

Fotovoltaico

PostgreSQL

PHP

Python

Sommario

Indice Figure

Introduzione

L'enorme progresso tecnologico dell'ultimo secolo ha creato sfide importanti per la gestione e lo smaltimento dei rifiuti derivati da apparecchiature elettroniche (comunemente noti come *RAEE*).

Il progresso tecnologico ed un mondo sempre più informatizzato e *connesso* hanno portato a richieste energetiche sempre maggiori, che però hanno fatto sorgere problemi di natura ambientale derivati dai gas (anidride carbonica in modo particolare) generati dalla combustione di combustibili fossili. Ciò costituisce tutt'oggi un importante problema ambientale, ed ha creato la necessità della produzione di energia da fonti differenti e rinnovabili – la cosiddetta *energia "pulita*".

La fonte di energia "principe" tra quelle rinnovabili è costituita dall'energia solare, inesauribile (perlomeno, si stima, per i prossimi 5 miliardi di anni!) e disponibile in grande quantità. L'energia solare può essere trasformata in energia elettrica mediante l'utilizzo di particolari dispositivi optoelettronici: i *moduli fotovoltaici*.

Il mercato del fotovoltaico ha avuto un enorme crescita nel corso degli ultimi venti anni, e questo ha fatto sorgere la necessità di un'adeguata regolamentazione per il loro smaltimento; sono, infatti, state emanate svariate Direttive Europee (integrate nell'ordinamento giuridico italiano mediante Decreti Legislativi), che identificano i moduli fotolitici come dispositivi appartenenti alla categoria RAEE.

Le normative attualmente vigenti richiedono la dichiarazione dei pannelli fotovoltaici immessi sul mercato italiano da parte dai relativi produttori. Ai produttori è richiesto il pagamento di una somma di denaro (detta *contributo ambientale*) che garantirà il ritiro ed il corretto smaltimento dei moduli al termine del proprio ciclo di vita, senza che sia compito del privato / azienda che li ha acquistati e montati (ad esempio, sulla propria abitazione) occuparsene.

Il progetto di cui questa tesi è oggetto nasce dalla necessità, da parte di un consorzio di filiera italiano, della memorizzazione delle dichiarazioni dei contributi ambientali dichiarati da parte dei produttori di moduli fotovoltaici, nonché delle informazioni riguardanti le eventuali installazioni dei moduli.

La realizzazione del sistema software per la gestione delle dichiarazioni dei contributi ambientali mi è stato commissionato dall'azienda reggiana Multitraccia s.c., come oggetto del tirocinio universitario che ho svolto in collaborazione con essa. Il progetto ha richiesto la completa riprogettazione della base di dati attualmente presente per un consorzio di filiera (con cui Multitraccia lavora), al fine di aumentarne le performance e di garantire la corretta memorizzazione dei dati, che dovranno essere accessibili per i prossimi 10/20 anni.

Il presente documento è strutturato in cinque capitoli, il cui contenuto illustrerà al lettore tutte le fasi relative allo studio delle tecnologie, l'analisi dei requisiti del sistema, la progettazione e l'implementazione del sistema software nonché dei test svolti al fine di verificarne il corretto funzionamento.

Il primo capitolo introdurrà la collaborazione con Multitraccia, la nascita del progetto ed un preliminare studio dei suoi obiettivi.

Nel secondo capitolo verranno descritte le tecnologie utilizzate per la realizzazione di questo progetto, e sarà illustrata l'analisi preliminare del sistema software attualmente usato dal consorzio al fine di individuare i software più appropriati.

Seguiranno il terzo ed il quarto capitolo, che hanno riguardato rispettivamente le fasi di progettazione e di implementazione del sistema software. Verranno dettagliatamente analizzati i requisiti ed i casi d'uso del sistema, e descritte le scelte che hanno costituito la struttura del database e del sito web implementati.

Infine, verranno descritti alcuni dei test più importanti, che hanno permesso di analizzare le performance del sistema evidenziandone criticità che è stato necessario correggere.

Seguirà un breve capitolo conclusivo, con una breve riflessione sul lavoro svolto e sull'importanza del sistema software realizzato. Verranno, infine, descritti alcuni dei possibili sviluppi del sistema nel corso di una futura collaborazione con Multitraccia.

Prima di procedere con l'analisi del caso di studio e degli obiettivi (oggetto del *Capitolo 1*), è necessario fare alcune importanti premesse:

- Tutti i loghi inseriti all'interno del documento sono di pubblico utilizzo; Multitraccia mi ha dato l'esplicito consenso all'utilizzo del proprio logo
- Tutte le immagini utilizzate (ad eccezione dei loghi di aziende / prodotti software) sono di mia realizzazione, al fine di non violare copyright
- In concordanza con l'accordo per la tutela della privacy e dei segreti aziendali stipulato con Multitraccia, alcune parti del progetto ed immagini sono state alterate (in toto od in parte) al fine di eliminare ogni possibile riferimento al consorzio di filiera ed ai suoi clienti

Parte I

Il Caso di Studio

Capitolo 1

Analisi degli Obiettivi

1.1 – Introduzione

Questo capitolo introdurrà la collaborazione con Multitraccia s.c. e descriverà i tratti principali della richiesta del progetto proposto dall'azienda come argomento di tirocinio nonché di laurea. Sarà analizzato lo scopo del progetto di laurea, dando un breve sguardo alla realtà di interesse, e saranno elencati alcuni riferimenti normativi relativi alla gestione dei RAEE (Rifiuti di Apparecchiature Elettriche ed Elettroniche) sui quali il progetto è fondato.

1.2 – Collaborazione con Multitraccia

La realizzazione di questo progetto nasce dalla collaborazione con **Multitraccia s.c.** [1], come argomento del tirocinio universitario svolto nel corso dell'anno accademico corrente.

Figura 1.1 – Logo di Multitraccia

Multitraccia s.c. (logo in *Figura 1.1*). è una società che ha sede a Reggio Emilia e che offre servizi nell'ambito informatico, come la realizzazione e l'hosting di siti web, servizi di networking, assistenza software e la stampa di digitale.

Il primo incontro con l'azienda è avvenuto a Novembre 2019 nel corso di una breve conferenza, presso la sede del dipartimento FIM dell'Università degli Studi di Modena e Reggio Emilia, durante la quale alcune aziende del modenese e del reggiano hanno esposto i servizi di cui si occupano ed hanno descritto alcune proposte di tirocinio. Durante la suddetta giornata ho avuto il piacere di conoscere Francesco Gregori, il quale ha successivamente ricoperto il ruolo di tutor aziendale nel corso del mio tirocinio.

A seguito di un colloquio conoscitivo con Multitraccia, avvenuto sempre nel corso del mese di novembre, è stata scelta la realizzazione del progetto di cui il presente documento è oggetto. Il progetto è stato motivato

dal mio grande interesse per il mondo dei **Big Data**, una realtà che sta diventando sempre più importante e fondamentale al giorno d'oggi e che è alla ricerca di personale sempre più specializzato.

1.3 – Richiesta Iniziale

Multitraccia mi ha proposto di occuparmi della **progettazione** di una **base di dati** per la **gestione dei contributi ambientali di moduli fotovoltaici** per un consorzio di filiera.

Il contributo ambientale rappresenta una quota, da versare allo stato italiano, che garantisce il corretto smaltimento di un pannello fotovoltaico al termine del suo ciclo di vita. Tale quota viene pagata dal produttore del pannello fotovoltaico, nel momento della sua immissione nel mercato italiano.

La banca dati permetterà al consorzio di verificare se è possibile effettuare il ritiro e lo smaltimento del pannello fotovoltaico senza che chi lo ha acquistato, e montato presso la propria abitazione, debba pagare il servizio, dato che il produttore ha già pagato il servizio mediante il versamento del contributo ambientale.

1.4 – Scopo del Progetto

Lo scopo del presente progetto è la **realizzazione** di **una base di dati** per la **gestione dei contributi ambientali di moduli fotovoltaici** per un consorzio di filiera. La banca dati non verrà creata da zero, ma dovrà essere costruita sulla base di un sistema già esistente.

Il sistema utilizzato dal consorzio di filiera è stato realizzato nel 2010 e risiede su server Mac di tale anno; la base di dati è stata realizzata mediante Filemaker. Le modifiche effettuate nel corso degli anni per l'adeguamento a nuove normative e l'elevato numero di record da gestire (al momento superiore a quattro milioni) ha creato la necessità di una riprogettazione della base di dati su sistemi e macchine più recenti.

Multitraccia, a fronte delle limitazioni di FileMaker, ha richiesto la scelta di un nuovo DBMS (proponendo inizialmente MongoDB), al fine di realizzare un database ben strutturato che garantisca **prestazioni migliori** e, in modo particolare, la **consistenza** e la **persistenza** dei dati. Infatti, i dati relativi al pagamento dei contributi ambientali dovranno essere correttamente accessibili per i prossimi 10 o 20 anni.

1.5 – Riferimenti Normativi

Le modalità di smaltimento e recupero dei rifiuti di apparecchiature elettriche ed elettroniche (RAEE) sono descritte e disciplinate dalla Comunità Europea mediante svariate Direttive Europee emanate nel corso degli ultimi 20 anni. Tra le prime direttive si trova la direttiva 2002/95/CE [2] e 2002/96/CE [3], modificate l'anno seguente con la direttiva 2003/108/CE [4]; tra le direttive più recenti troviamo, invece, la 2012/19/UE [5].

Il Parlamento Italiano ha integrato le direttive europee nel proprio ordinamento giuridico. Tra i decreti più importanti troviamo:

- il Decreto RAEE 25 Luglio 2005 n. 151 [6] ed il D.lgs. 152/2006 [7], i primi ad attuare le Direttive Europee sopra citate

- il Decreto 208/2008 [8], che rinvia l'entrata in vigore del regime del "new waste" (ogni produttore di apparecchi elettrici ed elettronici è tenuto a sostenere i costi della raccolta, del recupero e dello smaltimento sicuro di tutti gli apparecchi che ha immesso sul mercato italiano) al 1º gennaio 2010.
- il Decreto Ministeriale 8 Marzo 2010 n. 65 [9], che semplifica le modalità di gestione dei RAEE da parte dei distributori e gli installatori di tali apparecchiature, nonché da parte dei gestori di assistenza tecnica.

Capitolo 2

Tecnologie Utilizzate

2.1 – Introduzione

In questo capitolo verranno descritte le fasi preliminari relative allo studio dell'arte ed alla scelta delle tecnologie, che hanno preceduto la progettazione della base di dati e del sito web.

In particolare, verrà fatto un confronto tra i principali DBMS e tra i database relazionali e non relazionali, al fine di individuare la tecnologia più opportuna per raggiungere gli scopi progettuali. La scelta del DBMS sarà nuovamente discussa, in modo più dettagliato e come conseguenza di alcune scelte progettuali, nei capitoli di Progettazione della Parte II del presente documento.

In questo capitolo, inoltre, saranno elencati gli strumenti di comunicazione utilizzati con Multitraccia, insieme al metodo di sviluppo del software adottato.

2.2 – Studio dello Stato dell'Arte

Il presente progetto si basa su una solida realtà già attiva da ben 10 anni: Multitraccia, infatti, nel 2010 ha realizzato una banca dati per la memorizzazione dei tracciati di moduli fotovoltaici per un consorzio di filiera, il quale si occupa della gestione dei RAEE al termine del loro ciclo di vita.

La **base di dati** è stata realizzata mediante l'utilizzo del DBMS (DataBase Management System) **FileMaker** (precedentemente noto come Claire), e risiede su **server MacOS X** del 2010. La base di dati è replicata e suddivisa su due server, il primo con sistema operativo MacOS X 10.6 mentre il secondo con MacOS X 10.5. La progettazione e l'implementazione dell'architettura di rete non rientrano tra gli obiettivi di questo progetto; tuttavia la seguente

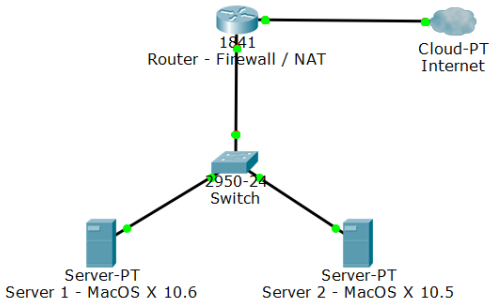


Figura 2.1 - Infrastruttura di Rete

figura (*Figura 2.1*) mostra, ai fini della comprensione del lettore, uno schema dell'infrastruttura esistente.

La base di dati, a seguito delle modifiche effettuate nel corso degli anni per l'adeguamento a nuove normative e per via dell'elevato numero di record da gestire (al momento superiore a quattro milioni – e che Multitraccia prevede si triplicherà nel corso dei prossimi 10 anni), ha creato la necessità di una sua completa riprogettazione su sistemi software e hardware più recenti. L'obiettivo è di garantire la **persistenza**, la **consistenza** e la **coerenza** dei dati per almeno i prossimi 10 o 20 anni, su un **sistema reattivo e prestante**.

Il **sito web** esistente è stato realizzato mediante l'utilizzo dei **linguaggi PHP, Javascript, CSS** ed **HTML5** ed è hostato dai suddetti server. Il sito web non richiede una riprogettazione totale, ma necessita, invece, dell'integrazione con la nuova base di dati al fine di garantire l'accesso a tutte le funzionalità già esistenti. Ci siamo posti l'obiettivo di non modificare particolarmente l'interfaccia web, così da permettere al cliente l'accesso a pagine simili a quelle che già conosceva ed utilizzava; le scelte progettuali relative alla progettazione dell'interfaccia grafica vanno oltre lo scopo di questo capitolo introduttivo, e saranno descritte dettagliatamente nel *Capitolo 3.6*.

2.3 – Database

Un database (o base di dati) è una **collezione di dati**, tipicamente **strutturati**, memorizzati all'interno di un sistema informatico.

I database sono generalmente memorizzati all'interno di **server**, che permettono accessi simultanei ad alte prestazioni e garantiscono la persistenza dei dati mediante meccanismi di duplicazione e correzione degli errori. I database vengono gestiti dai **DBMS** (DataBase Management System), software altamente specializzati che permettono la creazione, la manipolazione e l'interrogazione dei dati.

Al fine di superare i limiti prestazionali del DBMS FileMaker nella gestione di una banca dati contenente svariati milioni di record, Multitraccia ha richiesto la costruzione di un nuovo database mediante l'**utilizzo di un DBMS differente**.

La proposta iniziale di Multitraccia consisteva nell'utilizzo di MongoDB. Tale scelta è stata motivata dall'estrema flessibilità dei database di tipo non relazionale, che negli ultimi anni stanno suscitando un interesse sempre maggiore.

Un'analisi approfondita del database preesistente (di cui si parlerà più nel dettaglio nei *Capitolo 3.4.1*), unita ad una ricerca ed un confronto dei più DBMS più diffusi nel mondo (oggetto di questo capitolo), mi ha portato a scegliere una via alternativa. Multitraccia ha accolto positivamente la mia valutazione, e si è resa disponibile all'utilizzo di un DBMS differente rispetto a quello che aveva inizialmente proposto.

2.3.1 – Database Relazionali

I database relazionali sono una tipologia di database che memorizzano e strutturano i dati secondo il **modello relazionale** [10]. Il modello relazionale si basa sulla rappresentazione di dati come *relazioni*, e sulla loro manipolazione mediante operatori dell'algebra relazionale.

I DBMS di tipo relazionale rappresentano i dati in **tabelle**, permettendo la realizzazione di una struttura rigida e consistente (formalmente descritta da un **DDL** – Data Definition Language [11]). Le tabelle sono così costituite:

- uno o più **attributi** (campi dato), che costituiscono le colonne
- uno o più **tipi di dato** (o domini), che definiscono l'insieme dei valori che ogni campo (attributo) di una colonna può assumere
- un **valore** per ciascun attributo, all'interno del proprio dominio di valori (alcuni campi possono anche non essere valorizzati)
- **tuple** (o record), che contengono l'insieme non ordinato dei valori assunti dagli attributi

I record delle tabelle sono identificabili in modo univoco mediante appositi **campi detti chiave**, i quali devono sempre essere valorizzati ed avere valore univoco all'interno di una tabella.

Le tabelle sono collegate tra loro mediante appositi costrutti detti **associazioni**, che permettono di associare il record (identificato da una chiave primaria) di una tabella, detta *master*, ad uno o più record di un'altra tabella, detta *slave*.

L'insieme delle **operazioni** effettuabili sul database, sulle tabelle e sui dati in esse contenute (creazione, inserimento, lettura, aggiornamento e cancellazione) sono definite dal **DML** (Data Manipulation Language [12]).

L'accesso ai dati contenuti delle tabelle avviene mediante l'esecuzione di **query** (o interrogazioni), mediante l'utilizzo di costrutti algebrici e logici. Le modalità di accesso ai dati sono definite dal **DQL** (Data Query Language [13]).

L'insieme di DDL, DML e DQL costituisce l'**SQL** (Structured Query Language [14]), un linguaggio standardizzato che costituisce il pilastro fondante dei DBMS relazionali.

Su questa struttura rigida e precisa si fondano i cosiddetti **principi A.C.I.D.**, ovvero **Atomicità** delle transazioni (processi) sulle tabelle, **Consistenza** dei dati (alla fine ed a seguito di una transazione), **Isolamento** delle transazioni (ogni transazione è indipendente dalle altre), **Durabilità** ovvero la persistenza dei dati.

Vantaggi dei database relazionali

- l'utilizzo delle tabelle, la cui struttura è facilmente associabile a fogli di calcolo, permette una progettazione semplificata
- linguaggio di interrogazione molto espressivo
- linguaggi di accesso, manipolazione ed interrogazione dei dati standardizzato (SQL)
- i principi A.C.I.D. garantiscono una struttura altamente solida e consistente
- l'elevata diffusione ha portato alla creazione di una numerosa comunità di programmatori

Svantaggi dei database relazionali

- la struttura tabellare, ed i rigidi tipi di dato, costituiscono un limite nella manipolazione di aspetti complessi del mondo reale
- performance tipicamente inferiori rispetto a database non relazionali
- limitazioni nella scalabilità "orizzontale" (su più server) del lavoro

2.3.2 – Database Non Relazionali

I database non relazionali si contrappongono a quelli relazionali, storicamente più diffusi, fondandosi su concetti completamente differenti rispetto alle tabelle e con uno **schema non fisso**.

Alla base della loro diffusione, in forte crescita negli ultimi anni, vi è il **movimento NoSQL** [15], che promuove sistemi software caratterizzati dall'assenza dell'utilizzo del modello relazionale nella strutturazione dei dati.

Il termine NoSQL viene talvolta associato, erroneamente, a *"No SQL"*, il che induce a pensare alla totale assenza del modello relazionale; in realtà, tale termine è acronimo di *"Not Only SQL"*: infatti, il movimento non è contrario all'utilizzo dei database relazionali, ma afferma la presenza di casi d'uso nei quali tale modello rappresenta una forzatura.

I database non relazionali, nella maggior parte dei casi, non possiedono uno schema fisso. Il vantaggio principale di questa scelta risiede nella possibilità di organizzare **tipi di dati altamente eterogenei** (definibili anche dal programmatore), in strutture **facilmente modificabili** nel tempo.

L'assenza di uno schema fisso rende tali database **altamente scalabili** "*orizzontalmente*" (ovvero su più server), mediante meccanismi di replicazione parziale o totale dei dati e di accesso concorrente. Questa è stata la principale ragione della loro crescente diffusione negli ultimi anni, riconducibile ai bisogni delle grandi aziende (come Amazon e Google) nell'ambito del **Cloud Computing** [16].

Vantaggi dei database non relazionali

- elevante prestazioni e scalabilità "orizzontale"
- memorizzazione di tipi di dati altamente eterogenei
- grande flessibilità
- facilmente modificabili ed espandibili
- ottimi per la prototipazione

Svantaggi dei database non relazionali

- la duplicazione dei dati rende difficoltosi (se non impossibili) i controlli di integrità dei dati da parte del DBMS
- assenza di un linguaggio standard di interrogazione e modellazione dei dati
- community di programmatori più ristretta rispetto ai database relazionali, e minore reperibilità della documentazione

2.3.3 – FileMaker

FileMaker, precedentemente noto come Claris, è senza dubbio uno dei DBMS relazionali più famosi per i **sistemi Apple**. Infatti, FileMaker era inizialmente disponibile esclusivamente per piattaforme MacOS, contrapponendosi ai prodotti di Microsoft.

Figura 2.2 – Logo di FileMaker

Un punto chiave di FileMaker è la sua possibilità di manipolare direttamente entità come tabelle e schemi, unendo gli aspetti di visualizzazione dei risultati a quelli di memorizzazione ed elaborazione (aspetti tipicamente distinti nella maggioranza dei DBMS). Le **tabelle** sono spesso **salvate sullo stesso file**, il che rende molto semplice la gestione delle tabelle e la loro modifica, ma, nel caso di progetti grandi e complessi, costituisce un enorme svantaggio dal punto di vista prestazionale.

2.3.4 – MongoDB

MongoDB è il più diffuso DBMS non relazionale. MongoDB si allontana dalla struttura tabellare, tipica ei DBMS relazionali, in favore

Figura 2.3 – Logo di MongoDB

dell'utilizzo di **documenti in stile JSON** (detti BSON). È disponibile sotto licenze gratuite ed offre un consolidato supporto multipiattaforma.

Il punto di forza maggiore di MongoDB risiede nelle sue **prestazioni**. Infatti, mediante meccanismi di **replicazione automatica** del database, permette un bilanciamento del carico di lavoro tra le copie ed un'**elevata scalabilità**.

2.3.5 – Scelta del DBMS

Un'analisi approfondita della realtà di interesse e del database preesistente (di cui si parlerà più nel dettaglio nei *Capitolo 3.4.1*), ha fatto emergere una struttura dei dati piuttosto rigida e precisa, che ha subito poche modifiche sostanziali nel corso del tempo. Ciò mi ha fatto comprendere che **MongoDB**, il DBMS proposto da Multitraccia, **non era la scelta adatta**.

A seguito di un confronto con il mio tutor, Riccardo Martoglia, e con Francesco Gregori, il mio tutor aziendale, ed un ulteriore analisi da parte del team aziendale, è stata accolta la mia proposta relativa **all'utilizzo di un DBMS di tipo relazionale**. Seguono alcune considerazioni che hanno portato alla **scelta di PostgreSQL**.

Confronto tra i DBMS più diffusi

La prima fase che ha portato alla scelta del DBMS usato per la realizzazione del presente progetto è stata l'analisi ed il confronto tra i DBMS attualmente più diffusi [17] (vedere la *Figura 2.4* che segue).

358 systems in ranking, September 2020

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

Figura 2.4 – I 10 DBMS più diffusi

Sono stati analizzati i cinque DBMS attualmente più diffusi: Oracle, MySQL, Microsoft SQL Server, PostgreSQL e MongoDB. Tali database sono tutti di **tipo relazionale**, ad eccezione di MongoDB che è nella classifica con una popolarità sempre crescente negli ultimi anni (vedere la *Figura 2.5* a lato).

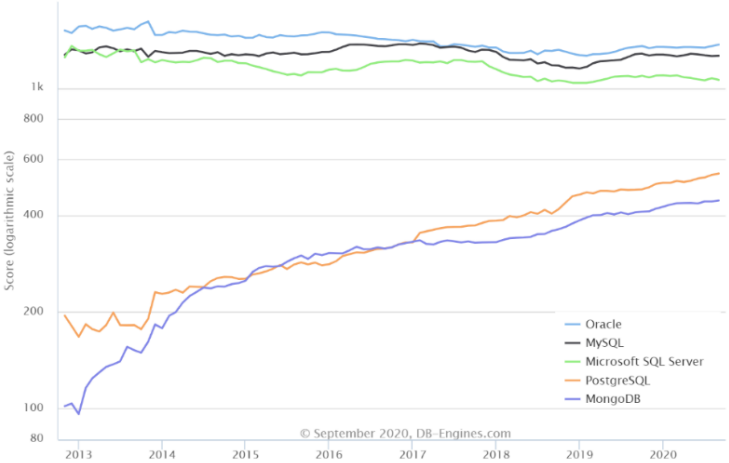


Figura 2.5 – Storico della diffusione dei 5 DBMS più diffusi

Tra i DBMS citati (escluso MongoDB, che si è rivelato inadatto), si distinguono due

categorie: prodotti su licenza gratuita (PostgreSQL e MySQL) e prodotti a pagamento (Microsoft SQL Server ed Oracle).

Oracle

Il DBMS di Oracle offre un supporto **multipiattaforma** e vanta una **sicurezza dei dati molto elevata**. Integra **procedure *PL/SQL***, permettendone un'esecuzione ottimizzata, parallelizzata, sicura e con una gestione efficace degli errori.

Microsoft SQL Server

Il DBMS di Microsoft è principalmente orientato a **piattaforme Windows** (recentemente è stato aggiunto un supporto per alcune distribuizioni Linux – totale assenza di supporto MacOS), con una forte integrazione con la piattaforma Cloud Azure.

Vanta una **sicurezza dei dati molto elevata**, ed utilizza una **versione di SQL proprietaria** di Microsoft (*Transact SQL*) che permette la realizzazione di script complessi ed offre prestazioni elevate.

MySQL

MySQL è un DBMS di proprietà di Oracle, rilasciato sia su licenza GNU che commerciale. Essendo scritto in *C++*, offre un **supporto multipiattaforma vastissimo** e molteplici linguaggi di programmazione ne permettono l'integrazione. Le versioni più recenti di MySQL aggiungono il supporto a documenti NoSQL e JSON.

Questo DBMS permette l'utilizzo di differenti *Storage Engine*, che permettono una buona verstilità e la possibilità di selezionare lo Storage Engine più appropriato per la gestione di determinate tipologie di database. Tuttavia, può avere **problemi di performance** se riceve troppe richieste nello stesso momento.

PostgreSQL

PostgreSQL è un DBMS open source con una popolarità in forte ascesa nel corso degli ultimi anni. Come MySQL, offre un vasto supporto **multipiattaforma**.

Migliora la programmabilità di SQL classico, in particolare nella costruzione di query complesse e di nuovi domini di dato. È **altamente scalabile**, ed offre prestazioni superiori rispetto a prodotti gratuiti concorrenti come MySQL.

I vincoli di Sistema Operativo

Un ulteriore vincolo, oltre alla preferenza di DBMS con licenza gratuita, è stato costituito dalla scelta del sistema operativo. Infatti, Multitraccia al momento possiede **server esclusivamente basati su MacOS**, e si è detta contraria all'utilizzo di sistemi Windows prediligendo la scelta di sistemi UNIX.

Per tale ragione, è stato effettuato un confronto tra le peculiarità e le criticità di entrambi i sistemi operativi:

- **MacOS** è un sistema operativo solido e facile da aggiornare e mantenere nel tempo; è disponibile esclusivamente per macchine Apple, le quali sono limitate nella configurabilità e presentano *single point of failure* (come la presenza di un singolo alimentatore)
- **Linux** è un sistema operativo altamente configurabile secondo le necessità del committente, ma più difficile da mantenere negli anni; i server Linux possono essere altamente personalizzati a livello componentistico, e con un costo tipicamente inferiore rispetto ai server Apple

Multitraccia, a seguito di un'analisi interna relativa ai punti di forza e le debolezze dei due sistemi operativi, unita all'esperienza consolidata nell'utilizzo di server Apple, ha scelto l'utilizzo di MacOS per il futuro server. Per tale ragione è stato scelto un prodotto compatibile con tale sistema operativo.

Scelta del DBMS

La preferenza di un DBMS su licenza gratuita e compatibile con MacOS ha fatto ricadere la scelta tra MySQL e PostgreSQL, entrambi DBMS altamente validi.

Su richiesta di Multitraccia, mi sono consultato con il professor Martoglia che, a seguito della sua esperienza con entrambi i DBMS, mi ha suggerito di optare per **PostgreSQL** in quanto più affidabile dal punto di vista prestazionale. Per tale ragione, il progetto su cui si basa la presente tesi è stato realizzato mediante l'utilizzo di tale DBMS (anziché MongoDB).

2.3.6 – PostgreSQL

PostgreSQL (logo in *Figura 2.6*), anche noto come *Postgre* (nome del vecchio progetto dello stesso), è uno dei più diffusi DBMS di tipo relazionale.

PostgreSQL si distingue dai DBMS concorrenti grazie alla propria programmabilità, più semplice ed espressiva. Il linguaggio proprietario ***PL/pgSQL*** (simile al linguaggio *PL/SQL* di Oracle) permette la **costruzione di nuovi tipi di dato** (domini) basati su quelli esistenti, con anche un meccanismo di ereditarietà dei tipi (uno dei principali concetti della programmazione orientata agli oggetti).

Figura 2.6 – Logo di PostgreSQL

Inoltre, è possibile definire funzioni (ovvero blocchi di codice SQL che è possibile richiamare) molto complesse. A differenza di SQL standard, PL/pgSQL offre il **supporto a strutture di controllo di ciclo e condizionali** tipiche dei linguaggi di programmazione. È infine possibile utilizzare *wrapper* per i più diffusi linguaggi di scripting (come Perl, Python e Ruby) e procedure in C e C++ per la realizzazione di logiche di programmazione complesse.

Per tali ragioni, PostgreSQL si sta diffondendo con popolarità sempre più crescente ed ha costituito la scelta ottimale per la realizzazione del presente progetto.

2.4 – Applicazione Web

Con Applicazione Web [18] (o, gergalmente, *web-app*) si indica una tipologia di applicazione distribuita web-based, ovvero un'applicazione distribuita accessibile tipicamente tramite un *Web Browser* (es. Google Chrome, Safari, Edge, ecc.).

Le applicazioni web non risiedono sulle macchine che le utilizzano, bensì risiedono su server remoti in un **architettura *client-server*** [19]. Tale architettura è esprimibile anche secondo la notazione *multi-tier* [20] (multi livello – *Figura 2.7* a lato): *presentation tier* (**front-end** – visualizzato sul client), *application/logic tier* (**back-end** – computazione sul server) e *data tier* (**database**).

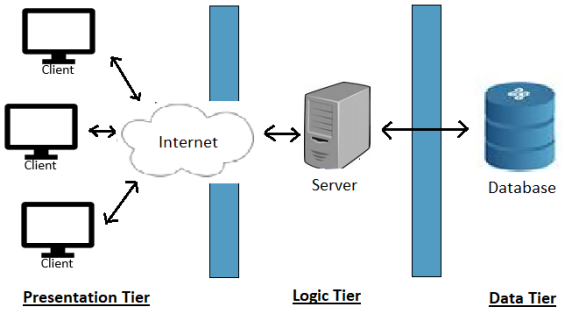


Figura 2.7 – Architettura Multi-Tier

Il progetto di cui questa tesi è oggetto richiede l'utilizzo di una web-app per consentire ai produttori dei pannelli fotovoltaici (ed agli installatori) di immettere nel sistema i dettagli relativi al pagamento dei contributi ambientali e di visualizzare i report trimestrali relativi ai moduli caricati nel sistema. Il sito web non dovrà essere progettato da zero, bensì verranno realizzate alcune pagine e sezioni per permettere l'interazione con la nuova base di dati ricalcando le funzioni ed il design del sito preesistente.

2.4.1 – Front-End

Con il termine Front-End ci si riferisce alla porzione della web-app visibile all'utente e con la quale è possibile interagire. L'interazione con l'applicazione avviene mediante una classica **pagina web**, che visualizza informazioni e permette l'acquisizione di dati in input.

Esistono molteplici linguaggi con la quale è possibile realizzare i front-end. In questo progetto, su richiesta del committente (Multitraccia), si useranno i seguenti linguaggi per garantire la piena compatibilità con il sito preesistente:

- **HTML** (HyperText Markup Language): utilizzato per definire la struttura delle pagine web
- **CSS** (Cascading Style Sheets): utilizzato per definire la struttura e la formattazione grafica delle pagine web
- **JavaScript**: linguaggio che permette di rendere dinamiche alcune sezioni della pagina, mediante l'esecuzione di script da parte del browser; mediante questo linguaggio verrà validata localmente parte dei dati in input, prima di inviarli al server
- **AJAX** (Asyncronous Javascript and XML): utilizzato per la gestione di alcuni contenuti dinamici delle pagine
- **jQuery**: libreria Javascript che semplifica la gestione dei contenuti dinamici delle pagine web, mediante una gestione modulare degli eventi della pagina

2.4.2 – Back-End

Con il termine Back-End si indica la porzione della web-app che si occupa **dell'interazione tra l'utente ed i dati memorizzati all'interno del database**.

Questa parte dell'applicazione, non visibile o accessibile dagli utenti (che interagiscono con il sistema mediante il Front-End), gestisce: l'elaborazione dei dati acquisiti mediante il front-end, le sessioni e l'autenticazione degli utenti che accedono al sistema, l'interazione con il database per l'invio all'utente di dati che ha richiesto, e molto altro.

Per garantire la compatibilità con il sito preesistente, il back-end verrà realizzato principalmente mediante l'utilizzo di **PHP**, unito ad HTML, CSS, e JavaScript che costituiranno il corpo delle pagine inviate al font-end.

Inoltre, per motivazioni che verranno descritte più dettagliatamente nel *Capitolo 3.6.3* e *Capitolo 5.2.3*, una porzione del Back-End richiederà l'utilizzo di uno script scritto mediante il linguaggio **Python**.

2.5 – Macchine Virtuali

Per poter effettuare lo studio e la successiva esportazione dei dati dal sistema originario, è stata richiesta l'installazione di macchine virtuali.

Con il termine macchina virtuale (spesso abbreviata con l'acronimo di *VM* – Virtual Machine) [21], si indica un'applicazione che, mediante un **processo di virtualizzazione** [], **emula il comportamento di una macchina fisica** (PC / Server) in un ambiente virtuale.

Questa particolare tipologia di software è realizzata, e gestita, da specifici programmi detti **HyperVisor** [22]. Gli HyperVisor gestiscono l'assegnazione di risorse hardware (risorse computazionali, memoria RAM e porzioni del disco rigido) alla macchina virtuale, "*facendo credere*" al sistema virtualizzato di essere eseguito su una macchina con tale tipologia di hardware o architettura (che, essendo emulati, può anche differire dall'hardware realmente presente sul PC/Server su cui l'HyperVisor viene eseguito).

Gli ambienti virtuali sono **totalmente indipendenti tra loro**, e permettono l'installazione di **sistemi operativi differenti rispetto** a quello presente sulla macchina sulla quale l'HyperVisor viene eseguito.

Il sistema originario, come descritto nel *Capitolo 2.2*, vede l'utilizzo di due server con sistema operativo MacOS X, che comunicano tra loro all'interno di una rete LAN. Il PC sul quale ho sviluppato questo progetto, invece, opera su un sistema Windows 10.

Nell'impossibilità di accedere fisicamente ai server, e non potendo installare MacOS sul mio computer (per ragioni di compatibilità, ma soprattutto di violazione della licenza d'uso siglata da Apple Inc.), è stata necessaria la **creazione di due macchine virtuali**.

A tale scopo, è stato utilizzato l'HyperVisor **VMware Workstation Pro 15**. Tuttavia, VMware Workstation non supporta ufficialmente la virtualizzazione del sistema operativo MacOS, per cui è stata necessaria l'installazione di una patch (realizzata da un team di esperti del settore) che ne permettesse l'utilizzo, sfruttando la compatibilità nativa di VMware con i sistemi UNIX (di cui MacOS fa parte).

Figura 2.8 – Logo di VMware

Sono state create due macchine virtuali per replicare i server di Multitraccia, con medesimo hardware virtuale (processore con 4 thread, 4GB di RAM e 40GB di disco), e con sistema operativo MacOS X (rispettivamente nelle versioni 10.6 Server e 10.5 Server). Al fine di emulare completamente il sistema originario, le due macchine virtuali sono state collegate all'interno di una rete LAN virtuale.

Tuttavia, per motivi di tempo e di compatibilità (l'installazione di FileMaker Pro è stata problematica in una delle due macchine), insieme a Multitraccia è stato deciso di utilizzare solo una macchina virtuale, sulla quale è stato installato **FileMaker Pro** (in versione 10) e sui cui è stato possibile **importare il database originario**. Attraverso questa macchina è stato, inoltre, possibile **esportare i contenuti della base di dati originaria**, che sono stati successivamente importati nel nuovo database mediante uno script scritto in linguaggio Python.

2.6 – Ambienti di Sviluppo

Per poter effettuare l'implementazione della base di dati, degli script di importazione e dell'applicazione web, è stato necessario l'utilizzo di **ambienti di sviluppo integrati** (*IDE*).

Gli IDE sono strumenti software molto potenti ed importanti per i programmatori, in quanto offrono un enorme supporto sia nella fase di programmazione (ad esempio evidenziando la sintassi del codice o facilitando l'installazione e la gestione di librerie ed API) che in quella di debug.

Per la realizzazione del progetto è stato scelto l'utilizzo degli IDE di **JetBrains** (precedentemente nota come **IntelliJ** – Logo nella *Figura 2.9* a fianco), la cui licenza d'uso è fornita gratuitamente agli studenti universitari.

Figura 2.9 – Logo di JetBrains

In particolare, sono stati utilizzati:

- **DataGrip**: IDE specializzato nell'implementazione e nella gestione di basi di dati (con anche la possibilità di eseguire query direttamente dall'IDE)
- **PyCharm**: IDE dedicato allo sviluppo di software scritti in linguaggio Python
- **WebStorm**: IDE specializzato nell'implementazione del front-end dei siti web
- **PhpStorm**: IDE specializzato per lo sviluppo del back-end dei siti web mediante il linguaggio PHP

2.7 – Strumenti di Comunicazione

La realizzazione di questo progetto ha richiesto l'utilizzo di svariati strumenti di comunicazione. La scelta e l'utilizzo degli strumenti descritti in questo capitolo è stata dettata dalla situazione di emergenza sanitaria relativa alla diffusione del virus *SARS COVID-19*, che mi ha costretto a **svolgere il progetto** (ed il mio tirocinio) **completamente in remoto**.

Figura 2.10
Logo di Skype

La comunicazione con Multitraccia è avvenuta principalmente mediante videoconferenze e messaggi tramite **Microsoft Skype** (logo in *Figura 2.8*). Parte della comunicazione, ed in particolare tutta quella avvenuta con il mio tutor Riccardo Martoglia, è avvenuta tramite **e-mail**.

Per far fronte ad alcune difficoltà riscontrate nell'utilizzo di FileMaker (DBMS che non conoscevo), ma anche per un confronto sul lavoro che stavo svolgendo, è stato utilizzato **TeamViewer** (logo in *Figura 2.9*) per la condivisione remota del mio schermo (e l'interazione con la mia macchina).

Figura 2.11 – Logo di TeamViewer

Figura 2.12
Logo di GitHub

L'intero storico dell'evoluzione del progetto è stato tracciato mediante l'utilizzo di **Git**, da linea di comando, e di **GitHub** (logo in *Figura 2.10*), la cui licenza è fornita dall'Università degli Studi di Modena e Reggio Emilia.

Infine, sono stati adoperati servizi di archiviazione cloud come **Google Drive** e **DropBox** per l'invio di alcuni file di grandi dimensioni.

2.8 – Metodo di Sviluppo Agile/XP

La realizzazione di questo progetto ha visto un'**evoluzione dei requisiti** nel corso del suo svolgimento, alcuni dei quali non erano del tutto chiari e/o definiti nel momento del suo inizio. Inoltre, come accennato nel precedente capitolo (*Capitolo 2.6*), la situazione di emergenza sanitaria in cui ci troviamo nel momento della stesura del presente documento mi ha portato allo sviluppo del progetto a distanza ed in modo abbastanza autonomo.

Per queste ragioni, è stato adottato il **modello di sviluppo Agile/XP** (eXtreme Programming) [23][24]. I metodi agili sono una famiglia di metodi di sviluppo che hanno in comune:

- **Rilasci frequenti** del prodotto sviluppato
- **Collaborazione** continua con il cliente (Multitraccia)
- **Documentazione** di sviluppo ridotta
- **Valutazione continua** dei requisiti e dei rischi dei cambiamenti

"***eXtreme Programming***" (XP) è una disciplina di sviluppo che accoglie pienamente il manifesto agile. È infatti caratterizzata da:

- **piccoli e frequenti rilasci** del prodotto che consentono un continuo confronto con il cliente
- il **cliente** (o un suo rappresentante) è **sempre disponibile per chiarimenti** relativi alle specifiche o per prendere decisioni critiche (gli sviluppatori non devono fare ipotesi od attendere)
- descrizione dei requisiti mediante "***user story***", ovvero descrizioni, tipicamente informali e senza l'utilizzo di terminologie tecniche, di ciò che il cliente si aspetta che il prodotto svolga
- lo sviluppo è guidato dai **test**
- **progettazione semplice**
- **documentazione minima**

L'utilizzo dei suddetti metodi di sviluppo ha consentito la realizzazione del progetto in tempi piuttosto rapidi (il tirocinio è stato svolto nel corso di tre mesi e mezzo) e con la soddisfazione di Multitraccia.

Tuttavia, un importante rischio intrinseco nella natura dei processi XP è relativo **all'impossibilità di effettuare alcuni test del prodotto** prima della piena integrazione del sistema. Infatti, il database e l'applicazione web descritti in questo progetto non sono stati ancora realmente integrati e messi in funzione. Ciò mi ha portato a dover affrontare delle scelte progettuali importanti e rischiose, senza la possibilità di verificarne immediatamente la correttezza.

Parte II

Progetto e Sviluppo

Capitolo 3

Progettazione

3.1 – Introduzione

Questo capitolo ha l'obiettivo di descrivere le fasi che hanno costituito la progettazione del progetto, elementi cruciali per la sua corretta realizzazione.

La prima parte della progettazione è dedicata all'analisi approfondita dei requisiti funzionali, minuziosamente descritti da Multitraccia.

La parte successiva descriverà l'analisi del database già esistente, la quale ha fatto emergere l'inconsistenza di alcuni dati ed ha portato ad una riprogettazione quasi completa della base di dati. Saranno, dunque, elencati tutti gli interventi e le scelte effettuate per garantire la creazione di un nuovo database che rispetti i requisiti di coerenza e consistenza dei dati. Inoltre, verrà dettagliatamente descritta la modellazione del nuovo database, mediante la costruzione dei diagrammi E/R (Entità-Relazione) e dello schema logico.

Infine, l'ultima parte di questo capitolo illustrerà il progetto dell'applicazione web, della sua interfaccia e della gestione del caricamento dei dati nel database.

3.2 – Requisiti Funzionali

L'analisi dei requisiti costituisce una delle fasi più importanti per la corretta realizzazione di un progetto. Il fine di questo processo è quello di stabilire:

- quali sono i **requisiti** (espliciti o impliciti) del committente (Multitraccia)
- quali sono i **vincoli** (espliciti o impliciti) imposti dal sistema

Le specifica dei requisiti è tipicamente composta dalle seguenti fasi:

- studio di **fattibilità**
- estrazione ed **analisi** dei requisiti

- **specifica** formale dei requisiti
- **validazione** dei requisiti

La specifica dei requisiti rappresenta un **accordo** tra lo sviluppatore del software ed il suo cliente / committente, e definisce le funzionalità che si dovranno ottenere mediante l'implementazione del software. I requisiti esprimono che cosa il sistema deve fare, ma non come: quest'ultimo aspetto (che non è sempre indipendente dal primo) viene deciso dall'implementazione del sistema.

I requisiti si suddividono in tre tipologie:

- requisiti **funzionali**: descrivono che il sistema dovrà offrire e come dovrà reagire agli input
- requisiti **non funzionali**: descrivono caratteristiche dei servizi offerti dal sistema, come l'affidabilità, la velocità di elaborazione, la resilienza, ecc.
- requisiti di **domino**: derivano immediatamente dal dominio applicativo

---

Durante la realizzazione del presente progetto, come già descritto nel *Capitolo 2.7*, è stato adottato il modello di sviluppo **Agile/XP**. Secondo tale modello, la specifica dei requisiti non è avvenuta mediante la redazione di appositi documenti tecnici – come il documento *SRS* (Software Requirement Specification) [25] – ma è stata realizzata mediante il susseguirsi della descrizione di ***user story***, le quali mi hanno permesso di comprendere ed analizzare tutte le richieste di Multitraccia senza l'utilizzo di termini eccessivamente tecnici e formali.

Nei prossimi due capitoli saranno descritte nel dettaglio le funzionalità principali richieste dal sistema software (requisiti di base), e le ulteriori funzionalità che sono state aggiunte e descritte durante il corso del mio tirocinio (requisiti aggiuntivi).

3.2.1 – Requisiti di Base

Si vuole progettare un database per la gestione dei contribuiti ambientali relativi a moduli fotovoltaici immessi sul mercato.

Multitraccia, come progetto del mio tirocinio, mi ha commissionato la realizzazione di una base di dati per la **memorizzazione dei contributi ambientali dei moduli fotovoltaici** (che afferiscono a specifici listini e categorie) e dei dettagli relativi all'eventuale **installazione** dei moduli. I contenuti memorizzati nel database dovranno essere accessibili mediante un'applicazione web. L'applicazione e la base di dati dovranno essere **compatibili con il sistema preesistente** (nelle modalità che saranno descritte più dettagliatamente nel corso di questo capitolo).

Memorizzazione Listini

La prima funzionalità analizzata consiste nella modalità di memorizzazione dei listini relativi ai moduli fotovoltaici.

Con il termine listino si intende la catalogazione di alcune specifiche caratteristiche relative ai moduli fotovoltaici, che la legislatura italiana (ed europea) richiede vengano adeguatamente descritte nel momento del pagamento del contributo ambientale.

Ogni listino è identificato da un id univoco interno e memorizza il **valore imponibile** (relativo al contributo ambientale), l'IVA che andrà applicata, un eventuale sconto, l'anno ed il trimestre di riferimento.

Ogni listino fotovoltaico afferisce ad una **Categoria** ed una **Sottocategoria** (descritte da un nome ed una descrizione), ed una **Tipologia** (anch'essa descritta da un nome ed una descrizione). Ad ogni categoria, sottocategoria e tipologia possono afferire più listini.

Ogni listino è identificato in modo univoco anche mediante un **codice di listino,** così composto: *idCategoria_idSottoCategoria_idTipologia*. Ad esempio, il record di listino che afferisce alla categoria 2 e sottocategoria 4, e descritto dalla tipologia 7, sarà identificato dal codice di listino 2_4_7. Il codice di listino permette ai produttori ed agli installatori di identificare un determinato listino.

Il vecchio database memorizza, inoltre, un codice composto (identico al codice di listino) ed un codice denominato "composto anno" che indica anche l'anno di riferimento del listino (nel formato *idCategoria_idSottoCategoria_idTipologia_anno_trimestre*). Tali attributi non sono più realmente utili, ma sono mantenuti per motivi di retrocompatibilità e costituiscono anch'essi degli identificatori.

Per ovviare all'errata organizzazione delle Categorie, Sottocategorie e Tipologie, nel vecchio database, che ha portato alla ridondanza e leggera inconsistenza di alcuni dati (vedere *Capitolo 3.4.1*), è stato necessario gestire diversamente tali entità. Ciò ha portato ad una **modifica nella numerazione dei codici di listino**. Multitraccia ha esplicitamente richiesto di memorizzare anche i vecchi codici di listino, composto e composto anno al fine di mantenere una piena retrocompatibilità con il vecchio sistema: i produttori e gli installatori continueranno ad utilizzarli.

Vengono inoltre indicati: un campo booleano che indica se il listino è attivo o meno, un campo booelano che indica lo stato del listino (che tipicamente coincide con il valore del campo attivo), il tipo batteria (non sempre valorizzato) ed un eventuale raggruppamento (R1, R2, R3, R4, R5) non sempre valorizzato. Infine vengono memorizzati i timestamp di creazione ed ultima modifica del record.

Listini Base e Listini Personalizzati

Esistono due tipologie di listini:

- **listino base**: si occupa della memorizzazione di tutte le caratteristiche appena descritte
- **listino personalizzato**: afferisce ai singoli produttori e specifica valori (relativi all'importo del contributo ambientale ed allo sconto) che possono essere differenti, tra i vari produttori, per via di differenti accordi commerciali con il consorzio di filiera

I listini personalizzati, oltre che da un campo id interno, sono identificati dai seguenti campi (ognuno di esso, singolarmente, costituisce un identificatore):

- composto (e vecchio composto): nel formato *idCategoria_idSottoCategoria_idTipologia_PIVAProduttore*
- composto anno (e vecchio composto anno): nel formato *idCategoria_idSottoCategoria_idTipologia_anno_trimestre_PIVAProduttore*

Memorizzazione Produttori

Al fine di poter memorizzare il pagamento dei contributi ambientali da parte dei produttori dei moduli fotovoltaici, è necessario memorizzare le seguenti informazioni: verrà innanzitutto indicata la P.IVA del produttore (che costituirà un identificatore univoco), il Codice Fiscale (che può coincidere con la P.IVA), il nome della Ragione Sociale, ed i recapiti telefonici e di posta elettronica.

Memorizzazione Moduli Fotovoltaici

La memorizzazione delle informazioni relative ai moduli fotovoltaici immessi nel mercato costituisce il fulcro intorno a cui ruota questo progetto. Infatti, ai moduli fotovoltaici è associato il pagamento del proprio contributo ambientale.

I moduli fotovoltaici (da qui in poi abbreviati come *moduli*), per ragioni prestazionali, sono identificati internamente mediante un id progressivo.

Ogni modulo possiede un **codice seriale** (da qui in poi abbreviato come *seriale*). Questo campo non rappresenterà un identificatore poiché il medesimo seriale può essere utilizzato da più produttori di pannelli fotovoltaici (non vi è alcuna organizzazione internazionale che ne regola l'utilizzo). Un seriale dovrebbe essere univoco per uno specifico produttore, ma non è garantito (per cui l'univocità non verrà considerata).

Ogni modulo viene identificato univocamente mediante un **campo composto** dalla concatenazione del suo codice seriale e della P.IVA del suo produttore, nel formato: *seriale_PIVA*.

All'interno della vecchia base di dati vi era un ulteriore identificatore detto "**codice di sistema**", il quale era univocamente generato dalla concatenazione tra il codice seriale del pannello, la P.IVA del produttore ed altri campi (non descritti da Multitraccia). Per ragioni di compatibilità, verrà mantenuto tale campo; i record provenienti dal DB originario manterranno i vecchi codici di sistema, mentre i nuovi moduli immessi nel sistema vedranno la valorizzazione di tale campo con un semplice numero progressivo, garantendo comunque l'univocità del campo.

I moduli fotovoltaici sono descritti da:

- **valore**: l'importo del contributo ambientale
- **unità di misura**
- **peso** (espresso in Kg)
- **numero celle**
- **potenza** (espressa in W)
- tipo di **comparto**: professionale o domestico
- modello del modulo
- campo **pagato**: indica l'avvenuto pagamento
- campo **stato**: tipicamente corrispondente allo stato dell'ordine (che sarà descritto nei paragrafi successivi)
- campo **attivo**: indica l'avvenuta installazione
- campo **modulo attivo**: indica l'avvenuta attivazione e messa in funzione del modulo
- **date** di **installazione** ed **attivazione**
- timestamp di inserimento ed ultima modifica del record

Memorizzazione Ordini

Con il termine *Ordine*, all'interno di questa realtà di interesse, si intende una transazione commerciale relativa all'**inserimento**, nella base di dati, **di moduli** fotovoltaici di un produttore **nell'arco di una giornata**.

Un produttore, dunque, può caricare svariati moduli fotovoltaici durante la giornata ma verranno tutti racchiusi in un solo ordine. Ogni modulo fotovoltaico viene dichiarato all'interno di un solo ordine (effettuato dal produttore del panello).

Un ordine è identificato univocamente, per motivi prestazionali, da un id progressivo. Inoltre, è identificato da un **codice ordine**, di lunghezza fissa (14 caratteri) nel formato *FTVxxxxxxxxxxx*, in cui le x indicano cifre

decimali; le cifre più a destra rappresentano la concatenazione dell'id dell'ordine e dell'anno in cui è stato effettuato, mentre le seguenti (più a sinistra) sono zeri.

Un ordine è descritto dai seguenti attributi:

- titolo
- email (del produttore)
- **imponibile**: totale (senza iva) degli importi relativi ai contributi ambientali dei moduli fotovoltaici appartenenti all'ordine
- **IVA**: valore percentuale dell'imposta al momento dell'effettuazione dell'ordine
- campo **pagato**: indica l'avvenuto pagamento
- campo **fatturato**: indica l'avvenuta fatturazione
- numero moduli
- **stato dell'ordine** (Aperto, Chiuso, Pagato, Inviato, Attivo, Inattivo)
- indirizzi email relativi ad un eventuale invio di un sollecito di pagamento (destinatari, destinatari in copia conoscenza ed in copia conoscenza nascosta) ed un campo che conteggia il numero di solleciti inviati
- **anno** e **trimestre** dell'ordine
- **data di invio** e **data di conferma**
- timestamp di creazione e di ultima modifica

Memorizzazione Installazioni

Al fine di memorizzare in maniera completa lo stato di attivazione dei moduli fotovoltaici dichiarati, Multitraccia ha richiesto la gestione della memorizzazione dell'installazione dei moduli.

Un modulo, che è stato dichiarato e caricato all'interno della base di dati, può essere installato a seguito della propria vendita. Si memorizzano dunque:

- **sede di installazione** (può essere la stessa per più moduli): memorizza l'indirizzo, il comune e la nazione della località nel cui presso è stato montato il pannello fotovoltaico
- **installatore**: rappresenta la ragione sociale che si è occupata dell'installazione di un determinato modulo (con attributi del tutti analoghi a quelli analizzati per i produttori)

Multitraccia ha come obiettivo futuro la realizzazione, all'interno del sito web, di una **mappa interattiva** che permetta ad ogni produttore e ad ogni installatore di visualizzare le località in cui sono stati installati i moduli fotovoltaici. Questo progetto sarà nuovamente descritto nella sezione conclusiva del seguente documento, che parlerà degli **sviluppi futuri** di questo progetto.

3.2.2 – Requisiti Aggiuntivi

Applicazione Web

Il più importante dei requisiti aggiuntivi, che Multitraccia ha specificato durante lo svolgimento del mio tirocinio, è stata la realizzazione di un'applicazione web che permettesse agli utenti del consorzio di filiera (produttori ed installatori) di interagire con la base di dati.

L'applicazione web **non è stata progettata da zero**, ma è stato necessario realizzare alcune pagine web (e sezioni) che ricalcavano le funzionalità, la struttura e lo stile grafico del **sito già esistente**.

Le pagine web realizzate si suddividono in due sezioni:

- **area tracciati**: permette la creazione ed il **caricamento** dei ***tracciati*** dei **moduli** fotovoltaici (descritti nel prossimo paragrafo)
- **area report**: permette la consultazione di **report trimestrali**, relativi agli ordini ed ai moduli fotovoltaici dichiarati, e dei tracciati caricati

Il progetto dell'applicazione web non ha richiesto la gestione delle sessioni degli utenti autenticati e dei relativi permessi nell'accesso a determinate sezioni riservate del sito web; la motivazione di questa scelta, decisa da Multitraccia, è dettata dal fatto che il sito web del consorzio di filiera è attualmente in fase di un completo rinnovamento e dunque non è stato possibile avere informazioni concrete sulla gestione degli utenti (ancora non definitiva); per tale ragione, è stata solo predisposta una gestione minimale degli accessi che dovrà essere modificata ed integrata a quella che sarà presente nel nuovo sito web.

Gestione dei Tracciati dei Moduli Fotovoltaici

Con il termine tracciato (o tracciato record) viene indicato un **file testuale** che contiene una **lista di seriali di moduli fotovoltaici** che il relativo produttore vuole dichiarare mediante la loro immissione nella base di dati del consorzio di filiera.

Ogni riga del file è strutturata nel seguente modo:

P.IVA Produttore	Codice Seriale Modulo	Codice Listino	Peso	Potenza	Comparto
12345678912	432432443	1_1_3	30,5	40	P

Il file testuale, in formato TXT, è codificato con il set di caratteri **Unicode UTF-8** [26]. I sei campi che compongono le righe del file sono separati mediante il carattere di tabulazione.

Il caricamento di un tracciato record afferisce alla **creazione di un ordine**: ogni ordine (di ogni singolo produttore), infatti, viene creato nel momento del caricamento, sul sito web, del primo tracciato in una determinata data. Il produttore, nell'orario di attività del sito (8:00 – 18:00), può caricare più tracciati record in momenti differenti, che comporranno un **unico ordine giornaliero** (chiaramente, ad ogni produttore corrispondere un ordine differente); è inoltre possibile eliminare i tracciati record nel corso della giornata in cui sono stati caricati.

Gestione degli Utenti

La realtà di interesse di cui il presente progetto è oggetto individua quattro tipologie di utenti:

- utente **produttore**: può caricare i tracciati record dei moduli fotovoltaici e consultare i report trimestrali relativi ai moduli dichiarati
- utente **installatore**: carica e visualizza informazioni relative alle installazioni dei moduli fotovoltaici
- utente **amministratore**: ha accesso completo alla base di dati ed al sistema, per operazioni di lettura, modifica ed inserimento di nuove funzionalità (nel pieno rispetto delle normative sulla privacy e sulla riservatezza dei dati)
- utente in **lettura**: ha accesso completo o parziale alla base di dati ed al sistema per operazioni di lettura (nel pieno rispetto delle normative sulla privacy e sulla riservatezza dei dati)

Ogni tipologia di utente memorizza le seguenti informazioni: **username** univoco, una password, e l'**indirizzo IP** [27] ed il timestamp dell'**ultimo accesso effettuato**.

Per ragioni di privacy e di rinnovamento del sito del consorzio di filiera (attualmente in costruzione), Multitraccia non ha fornito informazioni precise riguardo la memorizzazione e la gestione di tali dati. Per tale motivo, sono state effettuate una progettazione ed un'implementazione minimali che dovranno successivamente essere espanse per soddisfare i futuri requisiti dell'applicazione. Non verranno gestite l'autenticazione dell'utente e la sua registrazione al sistema.

Gestione dei Log

L'ultimo dei requisiti aggiuntivi descritto da Multitraccia riguarda la gestione dei log [28], ovvero della cronologia delle operazioni effettuate all'interno dell'ambito applicativo e della base di dati.

Si memorizzano i log relativi alle operazioni di tutte le tipologie di utenti. Ad essi si aggiungono i log riguardante l'esecuzione di **script** sul server.

Un log memorizza:

- l'**id** dell'**utente** che ha effettuato l'operazione
- l'**operazione** ed il suo **tipo**
- i **timestamp** dell'inizio e del termine dello svolgimento dell'operazione
- gli **script** utilizzati
- alcuni campi mantenuti per retrocompatibilità: i file utilizzati, il progetto di riferimento e la modifica effettuata

A tale tipologia di log è stata successivamente aggiunta la gestione dei log riguardanti le **modifiche ai record** delle tabelle memorizzate nel database. Tali log, mediante campi di tipo **JSON** [29], permettono di preservare la cronologia dei valori dei record memorizzando i vecchi valori ed eventualmente i nuovi.

Per le ragioni relative all'incompleta gestione degli utenti (descritte nel paragrafo precedente), la gestione dei log per gli utenti è stata progettata ed implementata in modo minimale – su indicazione del committente – al fine di predisporre il database ad una futura gestione di tali funzionalità.

3.3 – Diagramma dei Casi d'Uso

Con il termine **caso d'uso**, nell'ambito della progettazione e della specifica dei requisiti di un progetto, si indicano le modalità di **utilizzo del sistema** (che si vuole realizzare) da parte dei suoi **utenti** (denominati ***attori***). Il diagramma dei casi d'uso permette di rappresentare i diversi scenari di interazione con il sistema, dal "punto di vista" di un determinato attore, in modo schematico e mediante l'utilizzo di una notazione standardizzata.

La notazione standard descrive i seguenti componenti:

- **sistema**: delimita l'argomento del diagramma, specificando i confini del sistema descritto mediante il diagramma; è graficamente definito da un rettangolo, al cui interno verranno inseriti i casi d'uso
- **attore**: l'utente del sistema, che esegue le azioni che verranno schematizzate mediante il diagramma (gli attori non identificano soltanto esseri umani, ma anche dispositivi hardware o software che interagiscono con il sistema); la notazione tipicamente utilizzata per la loro rappresentazione è costituita da un omino stilizzato

- **caso d'uso**: esprime un comportamento desiderato o offerto dal sistema; i casi d'uso devono essere non ambigui e non scomponibili in altre attività; la notazione grafica consiste nel disegno di un ovale, al cui interno sarà sinteticamente indicata l'azione oggetto del caso d'suo
- **relazione associazione**: collega gli attori ai casi d'uso; è graficamente rappresentata da una linea continua, e può opzionalmente avere un nome, molteplicità e dei ruoli
- **relazione generalizzazione**: collega un attore, o caso d'uso, ad un altro più generale; la notazione grafica è simile a quella dell'associazione, ma con una freccetta piena che ne indica il verso
- **relazione inclusione**: indica una dipendenza tra casi d'uso; l'inclusione non è opzionale: la corretta esecuzione di un caso d'uso dipende da quella del caso incluso; la notazione grafica è una linea tratteggiata, con una freccia ad indicare il verso, e la scritta *<<include>>*
- **relazione estensione**: indica una dipendenza tra casi d'uso; l'inclusione è opzionale: il caso d'uso che estende (detto anche *client*) specifica un comportamento aggiuntivo per il caso che viene esteso (detto *supplier*); la notazione grafica è una linea tratteggiata, con una freccia ad indicare il verso, e la scritta *<<extends>>*

I diagrammi dei casi d'uso sono uno strumento ampiamente usato nella fase di **analisi dei requisiti** e di **progettazione** di un sistema (non necessariamente di natura informatica), perché permettono di ottenere una rappresentazione grafica e non ambigua della realtà di interesse che si vuole analizzare (o di una sua parte).

I seguenti due paragrafi illustreranno i casi d'uso dell'utilizzo del sistema da parte dei **produttori** dei moduli fotovoltaici e dei loro **installatori**. Non verranno descritti i casi d'uso della gestione dei log e degli utenti in quanto si tratta di requisiti aggiuntivi che sono stati implementati solo in maniera parziale per le ragioni descritte nel precedente capitolo (*Capitolo 3.2*).

3.3.1 – Diagramma dei Casi d'Uso: Produttore

Di seguito (*Figura 3.1*) viene riportato il diagramma dei casi d'uso del sistema, relativo all'interazione del Produttore con l'applicazione web del consorzio di filiera. Sono schematizzate le azioni che possono essere svolte all'interno dell'applicazione, con condizioni che escludono il verificarsi di situazioni non consentite.

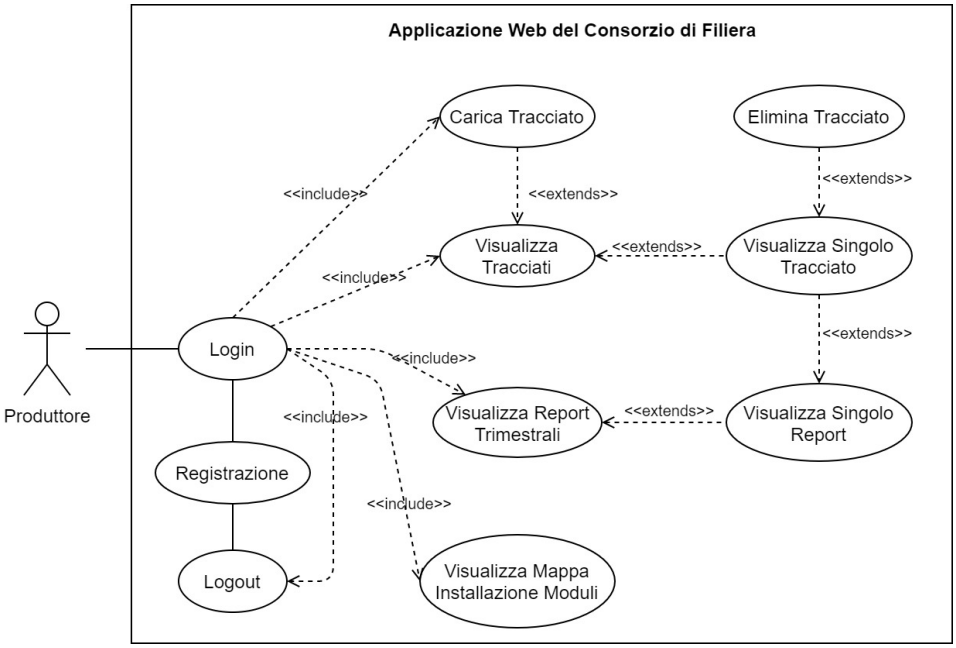


Figura 3.1 – Diagramma dei Casi d'Uso dell'Attore Produttore

3.3.2 – Diagramma dei Casi d'Uso: Installatore

Nella seguente figura (*Figura 3.1*) viene riportato il diagramma dei casi d'uso del sistema, relativo all'interazione dell'Installatore dei moduli fotovoltaici con l'applicazione web del consorzio di filiera. Sono schematizzate le azioni che possono essere svolte all'interno dell'applicazione, con condizioni che escludono il verificarsi di situazioni non consentite.

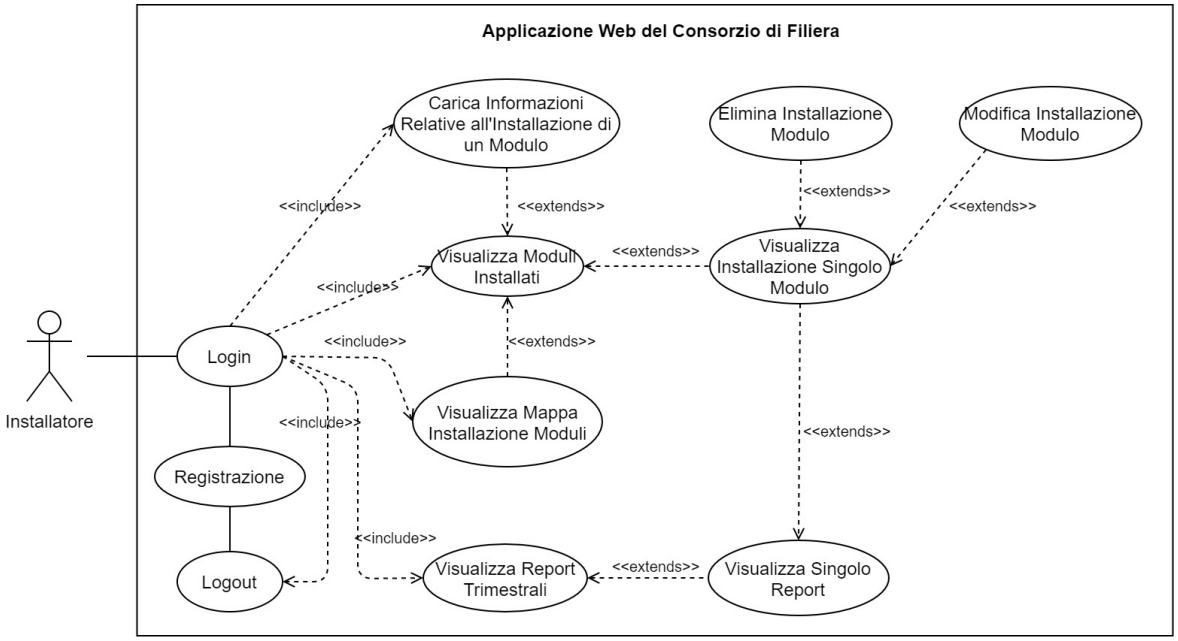


Figura 3.2 – Diagramma dei Casi d'Uso dell'Attore Installatore

3.4 – Progettazione Database: Analisi del Database Originario e Costruzione del Diagramma E/R

La progettazione della base di dati ha costituito il fulcro di questo progetto. La struttura del database è responsabile dell'organizzazione logica (ed in parte fisica) dei dati, ed ha lo scopo di evitare la creazione di ridondanza dei dati con il rischio di una loro incoerenza ed inconsistenza.

La progettazione del database è stato il passo immediatamente successivo all'analisi dei requisiti e dei casi d'uso del sistema software. È composta di tre fasi principali:

- **analisi del database originario**: per comprendere a pieno lo stato dell'arte e verificare l'eventuale presenza di problemi
- realizzazione del **Diagramma E/R** [30]: definisce la struttura logica dei dati che saranno memorizzati
- traduzione in **Schema Logico** [31] (*Capitolo 3.5*): definisce e normalizza la scelta delle entità, e dei relativi attributi ed attributi chiave, per la successiva implementazione mediante il linguaggio SQL (che sarà oggetto del *Capitolo 4.2*)

In questo capitolo verranno descritte le prime due fasi relative alla progettazione della base di dati, unite a considerazioni sulla scelta del DBMS.

3.4.1 – Analisi Database Originario

Prima di procedere alla progettazione vera e propria della nuova base di dati, è stato effettuato uno studio del database preesistente al fine di comprendere a pieno la sua struttura ed avere un **confronto con le scelte implementative** che furono **effettuate da Multitraccia** nel momento della sua realizzazione.

È stato di cruciale importanza il confronto con il mio tutor aziendale, Francesco Gregori, che mi ha dettagliatamente illustrato ogni dettaglio del database originario. Inoltre, sono stati discussi gli aspetti legati alla presenza di campi necessari per la corretta indicizzazione e per calcoli all'interno di FileMaker, che non sono stati inseriti nella nuova base di dati.

Il database originario, su cui codesto progetto si basa, è composto dalle seguenti tabelle (il cui significato è stato ampiamente descritto nel *Capitolo 3.2*, dedicato alla specifica dei requisiti del sistema):

- **listino base** e **listino personalizzato**
- **ordine**
- **dettaglio ordine**
- **produttore**
- **installatore**
- **elenco tracciati**
- **log**

Di seguito verranno analizzate le problematiche relative ad alcune tabelle ed alla ridondanza di alcuni campi e valori.

Attenzione: gli screenshot che seguiranno nel corso di questo capitolo rappresentano dei *facsimili*. L'**accordo per la riservatezza dei dati**, stipulato con Multitraccia, non mi consente di mostrare il reale contenuto della base di dati; al fine di illustrare alcune problematiche relative alle tabelle, verranno utilizzati dati inventati.

Listino Base e Personalizzato

Le due tabelle, fondamentalmente identiche nella loro struttura, memorizzano il contenuto dei listini dei moduli fotovoltaici.

All'interno di queste tabelle al valore di listino, l'eventuale sconto e l'IVA, sono memorizzate le Categorie, le SottoCategorie e la Tipologie. I dati relativi alle Categorie, le Sottocategorie e le Tipologie sono estremamente **ridondanti** e risulterà di fondamentale importanza memorizzarli in tabelle esterne, collegate ai listini mediante appositi vincoli di chiavi esterne.

La ridondanza dei dati, oltre a rendere meno performante il sistema, ha causato nel tempo ad una leggera **incoerenza** ed **inconsistenza** dei dati: infatti, i valori relativi ai campi sopra citati presentano una differente spaziatura e punteggiature ed un differente *case* dei caratteri. Questo rende estremamente difficoltose e le ricerche che richiedono un match esatto del contenuto di ricerca, e richiedono l'utilizzo di costose query con espressioni regolari.

Segue uno screenshot della tabella Listino Base (*Figura 3.3* nella pagina seguente) che mostra le problematiche descritte.

a_codice_listino	a_cod_categoria	a_categoria	a_cod_sottocategoria	a_sottocategoria	a_cod_tipo...	a_tipologia
2_1_0	2	2. NomeCategoria2	1	a. Sottocategoria1	0	
2_1_1	2		1		1	1. Mq da 0,50 a 1,00
2_1_2	2		1		2	2. Mq da 1,01 a 1,50
2_1_3	2		1		3	3. Mq > 1,50
2_2_0	2	2. NomeCategoria2	2	b. SottoCategoria2	0	
2_2_1	2		2		1	1. Mq da 0,50 a 1,00
2_2_2	2		2		2	2. Mq da 1,01 a 1,50
2_2_3	2		2		3	3. Mq > 1,50
2_3_0	2	2. NomeCategoria2	3	b. SottoCategoria2	0	
2_3_1	2		3		1	1. Mq da 0,50 a 1,00
2_3_2	2		3		2	2. Mq da 1,01 a 1,50
2_3_3	2		3		3	3. Mq > 1,50
2_4_0	2	2. NomeCategoria2	4	d. SottoCategoria 4	0	
2_4_1	2		4		1	Modulo Tipologia xxxx
2_4_2	2		4		2	Modulo

Figura 3.3 – Screenshot Tabella Listino Base

Ordine

La tabella Ordine memorizza tutte le informazioni relative ad un ordine da parte di un Produttore di moduli fotovoltaici.

Il primo problema riscontrato è l'enorme ridondanza di alcuni campi numerici: infatti, viene duplicata la memorizzazione dei valori decimali per rappresentarli sia nella notazione con la virgola, come separatore tra le cifre intere e quelle decimali, che in quella con il punto.

Il secondo problema è relativo alla memorizzazione di alcune informazioni relative all'anagrafica dei produttori e degli installatori (come il CF, la Ragione Sociale, la P.IVA, ecc.) che già risiede all'interno delle tabelle relative a tali entità. Inoltre, è stata riscontrata la mancata o errata valorizzazione di alcuni di questi campi, che non rispettano il formato prestabilito del campo. È fortunatamente possibile risalire ai dati corretti nelle apposite tabelle, ma la tabella ordine risulta **inconsistente**.

Segue uno screenshot della tabella Ordine (*Figura 3.4*) che mostra le problematiche descritte.

a_codice_ordine	a_anno	a_trimestre	a_stato	cf	p_iva	rag_sociale	z_id
FTV000[illegible]	20[illegible]	[illegible]	inattivo	[illegible]	[illegible]	[illegible]	[illegible]
FTV000[illegible]	20[illegible]	[illegible]	inattivo	[illegible]	[illegible]	[illegible]	[illegible]
FTV000[illegible]	20[illegible]	[illegible]	pagato	[illegible]	[illegible]	[illegible]	[illegible]
FTV000[illegible]	20[illegible]	[illegible]	inattivo		1_1_3		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		7°		[illegible]
FTV000[illegible]			inattivo		0		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]			inattivo		[illegible]		[illegible]
FTV000[illegible]	20[illegible]	[illegible]	inattivo		1		[illegible]
FTV000[illegible]			inattivo		PARTITA IVA		[illegible]
FTV000[illegible]			inattivo				[illegible]
FTV000[illegible]	20[illegible]	[illegible]	inviato	[illegible]	[illegible]	[illegible]	[illegible]
FTV000[illegible]	20[illegible]	[illegible]	pagato	[illegible]	[illegible]	[illegible]	[illegible]
FTV000[illegible]	20[illegible]	[illegible]	pagato	[illegible]	[illegible]	[illegible]	[illegible]

Figura 3.4 – Screenshot Tabella Ordine

Dettaglio Ordine

Nella tabella dettaglio ordine (che memorizza i dettagli relativi ai moduli fotovoltaici) è stata riscontrata la **medesima problematica** presente nella tabella ordine, appena descritta.

Oltre ai problemi relativi alla ridondanza e l'inconsistenza dell'anagrafica dei produttori e dei campi numerici, lo stesso problema si presenta anche con la memorizzazione di parte dell'anagrafica degli installatori che è comunque accessibile, tramite vincolo di chiave esterna, nella relativa tabella.

Segue uno screenshot della tabella Dettaglio Ordine (*Figura 3.5*) che mostra le problematiche descritte.

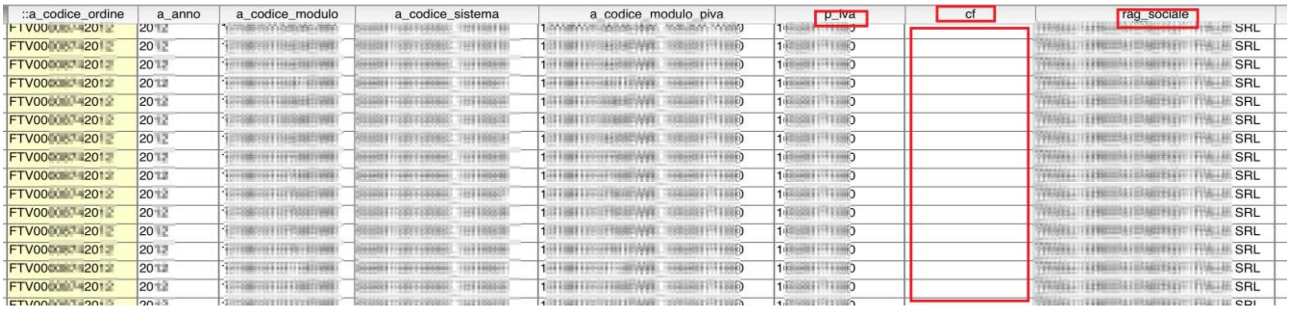


Figura 3.5 – Screenshot Tabella Dettaglio Ordine

3.4.2 – Scelta del DBMS

Il processo di analisi del database originario e dei requisiti del nuovo, unito ad un consulto con il mio tutor aziendale Francesco Gregori ed il suo collaboratore Roberto D'Autilio, mi ha permesso di comprendere che **MongoDB** (DBMS proposto da Multitraccia) **non risultava essere la scelta opportuna** per la realizzazione di questo progetto.

Analizzando dettagliatamente il database ed i requisiti del nuovo, è emersa un'**organizzazione dei dati molto rigida** e che ha subito **poche modifiche negli anni** (dovute principalmente all'aggiunta di alcuni campi per la memorizzazione di alcune caratteristiche dei moduli richieste dalle recenti normative vigenti). Tale struttura si presta decisamente bene per la **memorizzazione in tabelle**, caratteristica fondamentale dei **database di tipo relazionale**; i database di tipo non relazionale, a cui MongoDB appartiene, non utilizzano tale costrutto.

Per tale ragione, ricevuta l'approvazione di Multitraccia, ho effettuato una minuziosa ricerca dei principali DBMS di tipo relazionale. Dal loro confronto (oggetto del *Capitolo 2.3*), unito ad alcuni vincoli di sistema, alla preferenza di Multitraccia ed al parere del mio tutor Riccardo Martoglia, **PostgreSQL** è stato ritenuto il **DBMS più adatto** alla realizzazione di questo progetto.

3.4.3 – Schema Scheletro Macroscopico

In questa fase della progettazione, considerata l'estensione della realtà di interesse, si opta per una strategia modulare. La prima fase di progettazione del database consiste nella sua **osservazione macroscopica** mediante la realizzazione di uno schema scheletro, che evidenzierà le parti e le **entità principali** della realtà di interesse.

Nella pagina seguente è mostrato lo schema scheletro macroscopico (*Figura 3.6*)

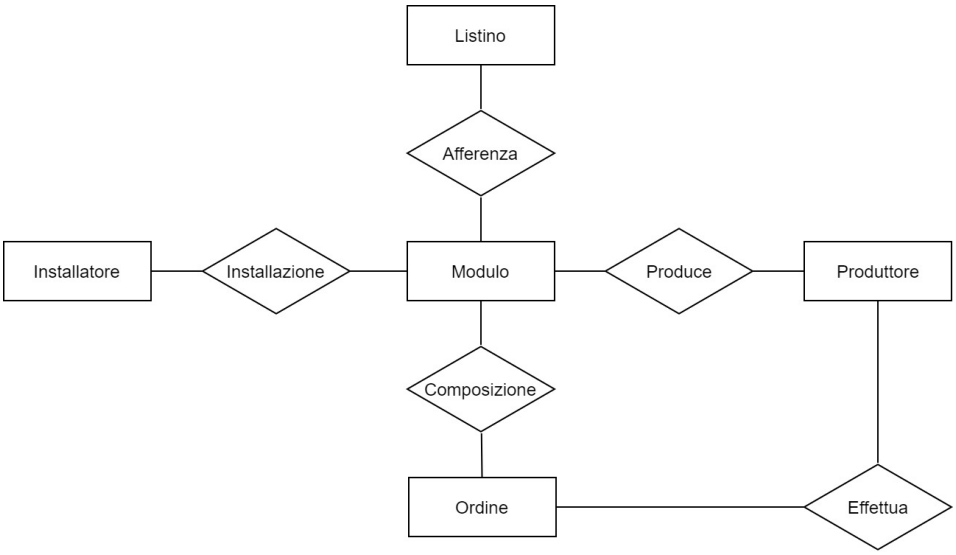


Figura 3.6 – Schema Scheletro Macroscopico

È possibile scomporre lo schema scheletro, sopra riportato, in tre macro sezioni:

- **gestione dei listini**: si occupa della gestione dei listini relativi ai moduli fotovoltaici, e della memorizzazione delle caratteristiche dei moduli stessi
- **gestione ordini**: si occupa della gestione dei dati relativi al pagamento dei contributi ambientali moduli fotovoltaici da parte dei produttori, mediante il caricamento di tracciati record
- **gestione installazioni**: si occupa della memorizzazione delle avvenute installazioni dei moduli, da parte dei relativi installatori

3.4.4 – Gestione Listini

Schema Scheletro

Dall'analisi dei requisiti, emerge la necessità di memorizzare dettagliatamente le proprietà relative alle Categorie (e Sottocategorie) ed alle Tipologie di moduli; inoltre, è necessario distinguere il Listino Base da quelli Personalizzati per i produttori. Segue lo schema scheletro che riassume tali caratteristiche (*Figura 3.7*).

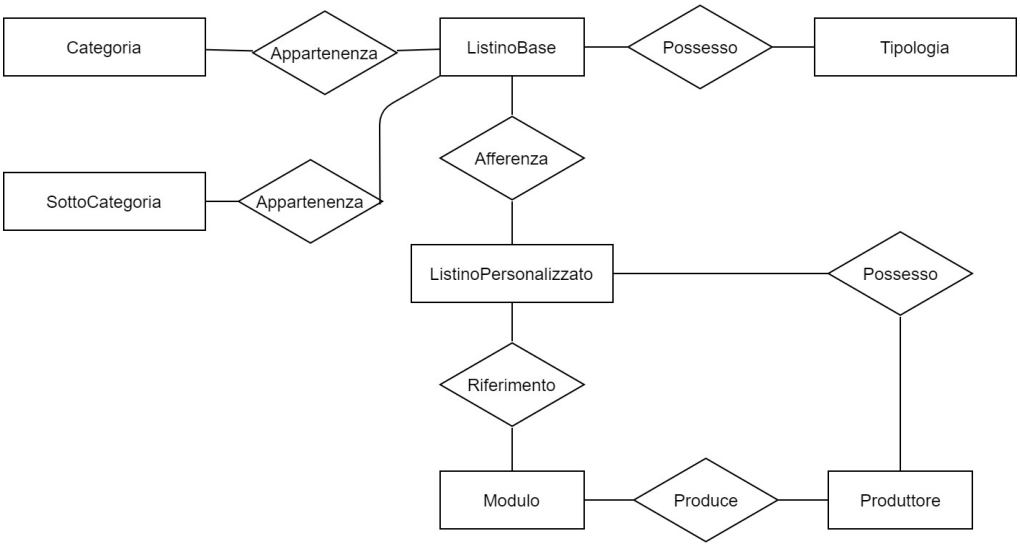


Figura 3.7 – Schema Scheletro Gestione Listini

Associazioni e Cardinalità

Sono dunque aggiunte le cardinalità alle relazioni con le seguenti considerazioni:

- una sottocategoria, altro non è che una categoria; può esserci al più un livello di ricorsione, non ammettendo la presenza di sottocategorie di sottocategorie (non esprimibile nello schema ER, ma oggetto di controllo nella fase di implementazione).
- con i termini *listino base* e *listino personalizzato* ci si riferisce al concetto di record del singolo listino; dunque, un produttore è associato a più record dell'entità listino personalizzato che, insieme, costituiranno il listino di tale produttore.

Segue diagramma E-R con le cardinalità delle associazioni (*Figura 3.8*):

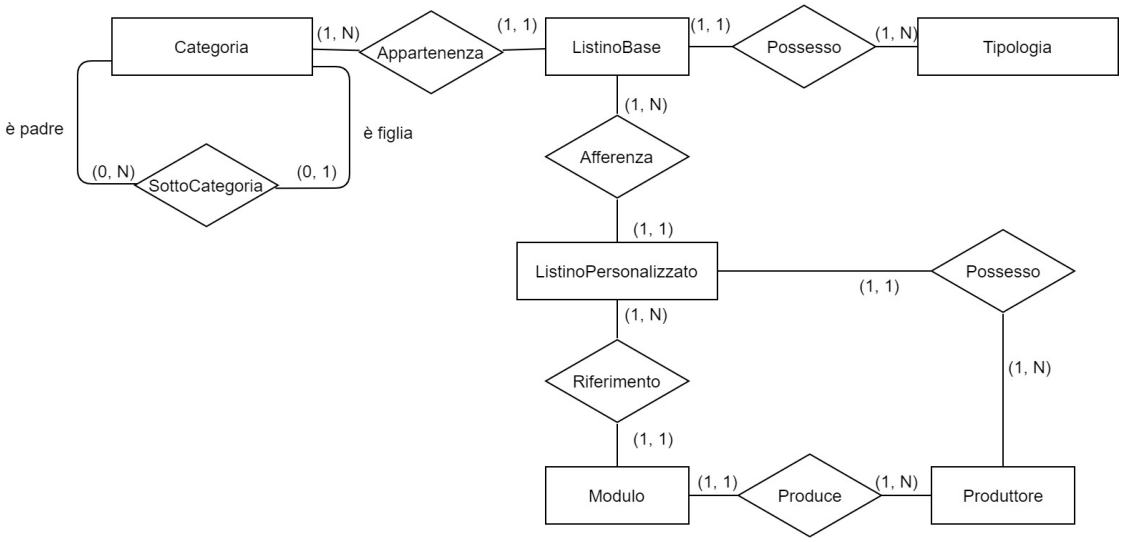


Figura 3.8 – Diagramma ER Gestione Listini: Associazioni e Cardinalità

Attributi e Chiavi

Vengono riportati gli attributi delle varie entità, con le seguenti considerazioni:

- nelle entità ListinoBase e ListinoPersonalizzato, i campi VecchioCodListino, VecchioComposto e VecchioCompostoAnno sono campi unique che costituivano gli identificatori nel database originario; tali campi sono stati mantenuti per compatibilità con il vecchio database
- tra gli identificatori dell'entità ListinoPersonalizzato ne è presente uno composto dal campo VecchioCodListino e dalla chiave esterna proveniente dall'associazione con l'entità Produttore: tale chiave è stata mantenuta per garantire piena compatibilità con il vecchio sistema
- il codice fiscale del produttore non può costituire un identificatore dell'entità in quanto alcuni Produttori, pur avendo P.IVA differente, vedono l'indicazione del medesimo CF

La scelta degli altri identificatori e degli attributi è stata motivata nel capitolo dedicato all'analisi dei requisiti del database (*Capitolo 3.2*).

Nella pagina seguente è mostrato il diagramma E-R completo relativo a questa porzione della realtà di interesse (*Figura 3.9*).

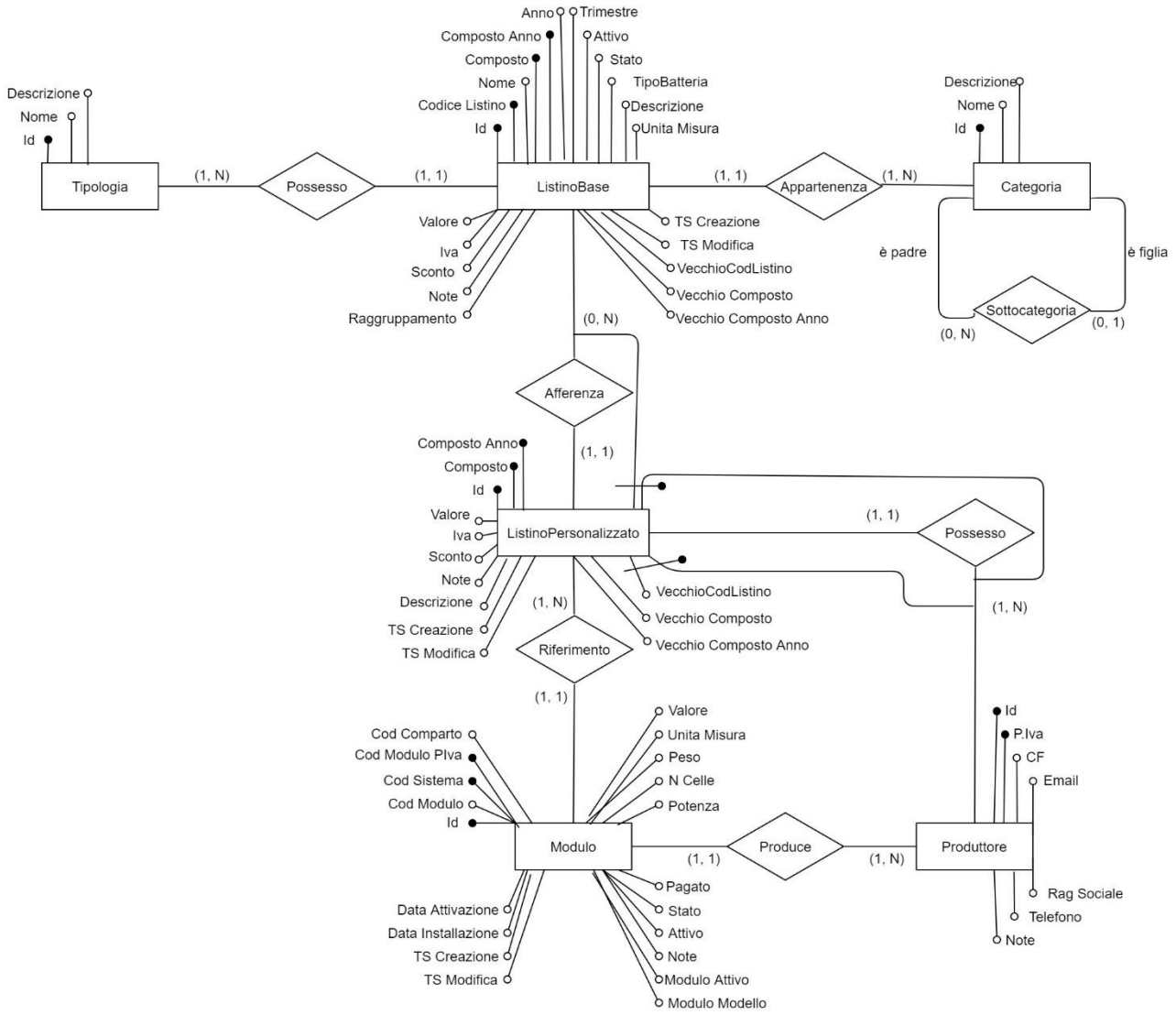


Figura 3.9 – Diagramma ER Gestione Listini: Attributi e Chiavi

3.4.5 – Gestione Ordini

Schema Scheletro

Dall'analisi dei requisiti, emerge la necessità di memorizzare i dettagli relativi al **caricamento dei tracciati**, che contengono le informazioni sui moduli fotovoltaici che costituiranno l'ordine del produttore. Segue lo schema scheletro (*Figura 3.10*) che riassume le caratteristiche della gestione degli ordini.

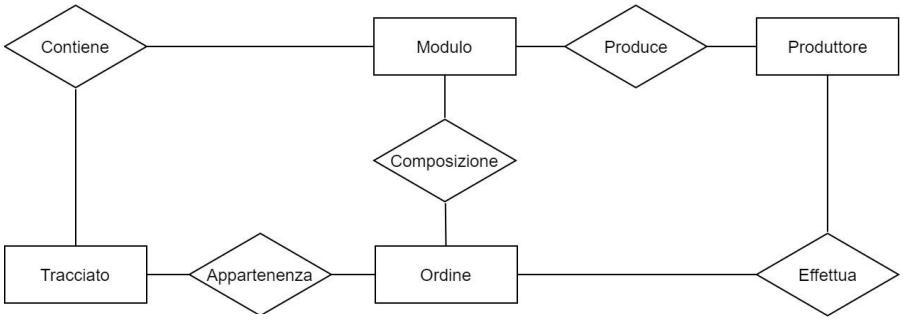


Figura 3.10 – Schema Scheletro Gestione Ordini

Associazioni e Cardinalità

Il seguente diagramma E-R (*Figura 3.11*) mostra la gestione degli ordini con la cardinalità delle associazioni. Non è emersa la necessità di modifiche rispetto allo schema scheletro riportato nel paragrafo precedente.

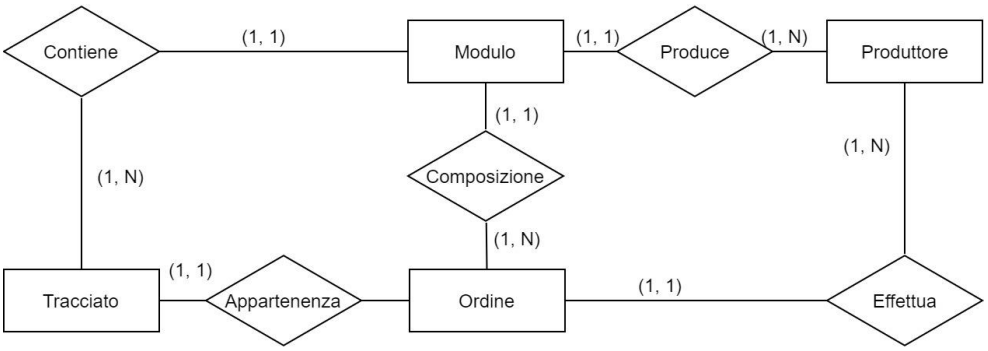


Figura 3.11 – Diagramma ER Gestione Ordini: Associazioni e Cardinalità

Attributi e Chiavi

Nella seguente figura (*Figura 3.12*) sono riportati gli attributi delle varie entità, con la seguente considerazione:

- i tracciati sono descritti dal nome del file (caricato dal produttore), dall'eventuale nome della cartella in cui è stato memorizzato sul server (tipicamente corrispondente alla P.IVA del Produttore) e di un campo per memorizzare il file stesso; la memorizzazione del file all'interno del database rende superflua la gestione delle cartelle, ma è stata inserita per motivi di compatibilità con il vecchio sistema

La scelta degli altri identificatori e degli attributi è stata motivata nel capitolo dedicato all'analisi dei requisiti del database (*Capitolo 3.2*).

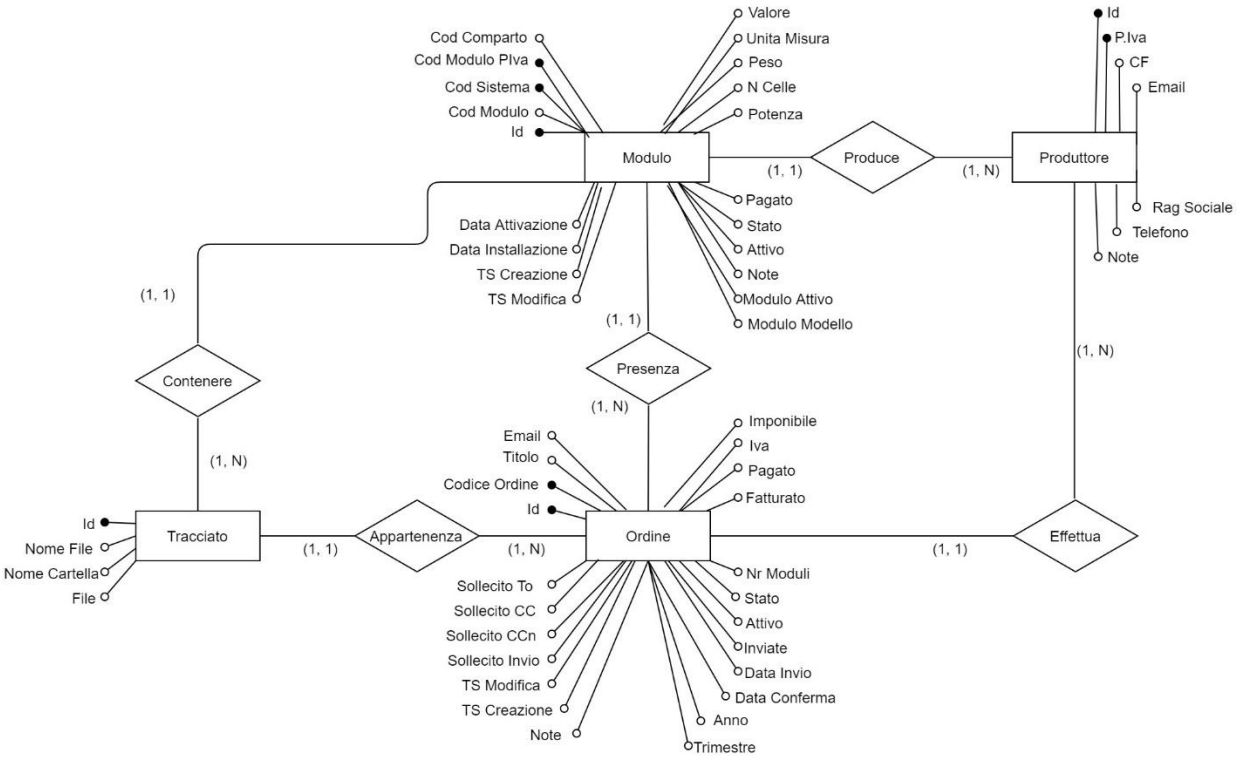


Figura 3.12 – Diagramma ER Gestione Ordini: Attributi e Chiavi

Attributi Derivati

Si effettua uno studio degli attributi derivati, che ha portato alla scelta di due attributi derivati per l'entità Ordine. Lo studio ha riguardato i seguenti dati derivati (calcolati mediante l'associazione con l'entità Modulo):

- totale imponibile
- nr. moduli

Le seguenti tabelle dei volumi sono state calcolate utilizzando **dati reali**. Il volume delle operazioni è stato calcolato considerando il numero di record inserito nel corso dei dieci anni di attività del database originario ed offre quindi una stima piuttosto veritiera. Per esempio: oltre 4 milioni di moduli, in 3.650 giorni, porta ad una media di 1.100 moduli caricati al giorno.

Tabella Volume Dati:

Concetto	Tipo	Volume dei Dati
Ordine	E	4.000
Modulo	E	4.000.000
Presenza	R	1.000

Tabella Volume Operazioni:

Operazione	Tipo	Volume delle Operazioni
1 – Aggiunta di un modulo ad un ordine	I	1.100/gg
2 – Visualizzazione del totale di un ordine	I	5/gg
3 – Visualizzazione del numero di moduli di un ordine	I	5/gg

Si associa alle operazioni di lettura un costo unitario, mentre a quelle di scrittura un costo doppio. Il costo di accesso alla relazione Presenza non viene considerato in quanto mediante la foreign key è possibile accedere direttamente ai record corrispondenti.

Si procede al calcolo dei costi delle operazioni **senza l'ausilio dei dati derivati**.

Costo Totale	Operazione	Concetti	Accessi	Tipo
1 * 2 * 1.100 = 1.100/gg	Aggiunta di un modulo	Modulo	1	S
1.101 * 5 = 5.505/gg	Visualizzazione totale ordine	Ordine	1	L
		Modulo	1.100	L
1.101 * 5 = 5.505/gg	Visualizzazione numero moduli di un ordine	Ordine	1	L
		Modulo	1.100	L

Il **costo totale senza dati derivati** corrisponde a 12.110 operazioni al giorno.

Si procede ora al calcolo dei costi delle operazioni **con l'ausilio dei dati derivati.** Il costo di accesso alla relazione Presenza non viene considerato in quanto mediante la foreign key è possibile accedere direttamente ai record corrispondenti.

Costo Totale	Operazione	Concetti	Accessi	Tipo
(2 * 2 + 1) * 1.100 = 5.500/gg	Aggiunta di un modulo	Modulo	1	S
		Ordine	1	L
		Ordine	1	S
1 * 5 = 5/gg	Visualizzazione totale ordine	Ordine	1	L
1 * 5 = 5/gg	Visualizzazione numero moduli di un ordine	Ordine	1	L

Il **costo totale con i dati derivati** corrisponde a 5.510 operazioni al giorno. **È evidente il beneficio derivato dall'utilizzo dei dati derivati** per la memorizzazione del numero dei moduli e del totale imponibile.

3.4.6 – Gestione Installazioni

Schema Scheletro

La gestione delle installazioni dei moduli fotovoltaici richiede la memorizzazione delle relative sedi di installazioni (abitazioni private, aziende, ecc.). Viene di seguito riportato lo schema scheletro relativo a questa porzione del database (*Figura 3.13*).

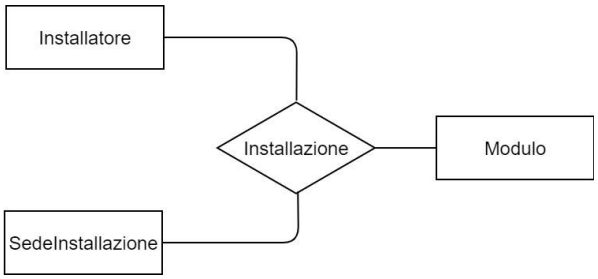


Figura 3.13 – Schema Scheletro Gestione Installazioni

Associazioni e Cardinalità

La gestione delle sedi delle installazioni richiede la memorizzazione di campi come il Comune e la Nazione (viene, infatti, prevista anche la gestione di moduli fotovoltaici a livello internazionale). Al fine di evitare la ridondanza dei dati, tali campi vengono memorizzati in specifiche entità. Si prevede la possibilità che nella stessa sede di installazione possano aver lavorato installatori differenti.

Il seguente diagramma ER (*Figura 3.14*) mostra lo schema risultante:

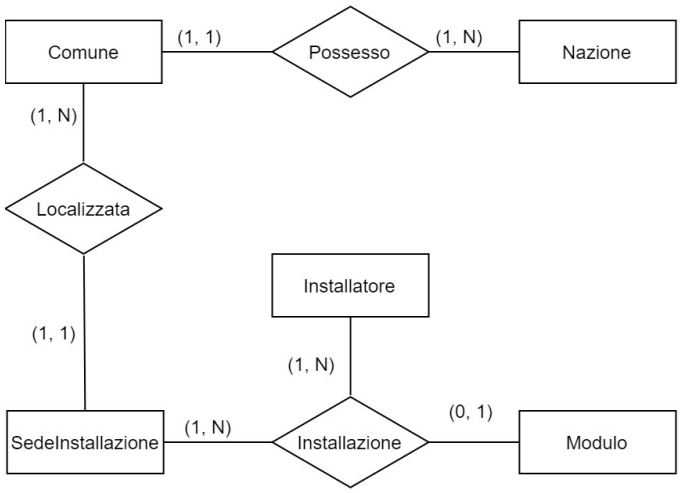


Figura 3.14 – Diagramma ER Gestione Installazioni: Associazioni e Cardinalità

Attributi e Chiavi

Nella seguente figura (*Figura 3.15*) sono riportati gli attributi delle varie entità, con le seguenti considerazioni:

- un installatore può installare moduli differenti, e può effettuare installazioni in sedi differenti
- comune e nazione sono identificati anche dal codice Istat
- il CAP tra comuni di differenti nazioni potrebbe, in un caso molto sfortunato, coincidere; tuttavia, considerando che gli stati adottano formati differenti tra loro e che il consorzio di filiera si occupa principalmente delle installazioni che avvengono nel territorio italiano, viene esclusa tale possibilità; per questa ragione, il campo CAP risulta identificatore dell'entità comune

La scelta degli altri identificatori e degli attributi è stata motivata nel capitolo dedicato all'analisi dei requisiti del database (*Capitolo 3.2*).

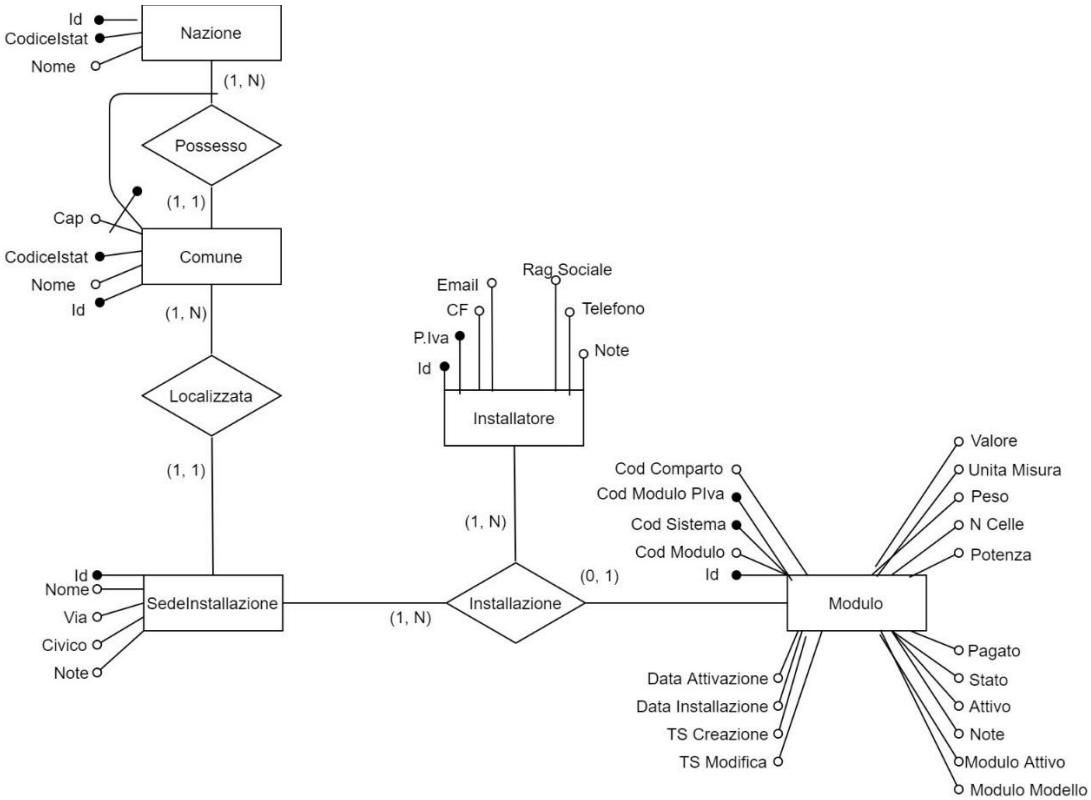


Figura 3.15 – Diagramma ER Gestione Installazioni: Attributi e Chiavi

3.4.7 – Gestione Utenti

Schema Scheletro

La gestione degli utenti, descritta nei *Capitolo 3.2.2* come requisito aggiuntivo, identifica quattro tipologie di utenti: Amministratore, utente in Lettura, Installatore e Produttore. Viene di seguito riportato lo schema scheletro relativo a questa porzione del database (*Figura 3.16*).

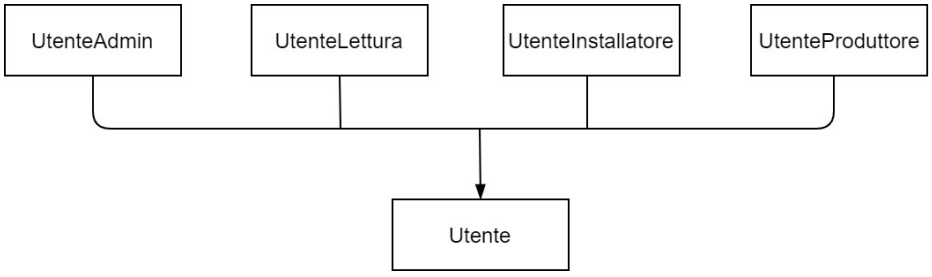


Figura 3.16 – Schema Scheletro Gestione Utenti

Associazioni e Cardinalità

La gestione degli utenti vede la presenza di una **generalizzazione**. Tale generalizzazione risulta essere di tipo **totale** (in quanto sono descritte tutte le tipologie di utenti presenti nella realtà di interesse) ed **esclusiva** (in quanto le tipologie di utenti sono distinte – un produttore non è anche amministratore)

Il seguente diagramma ER (*Figura 3.17*) mostra lo schema risultante:

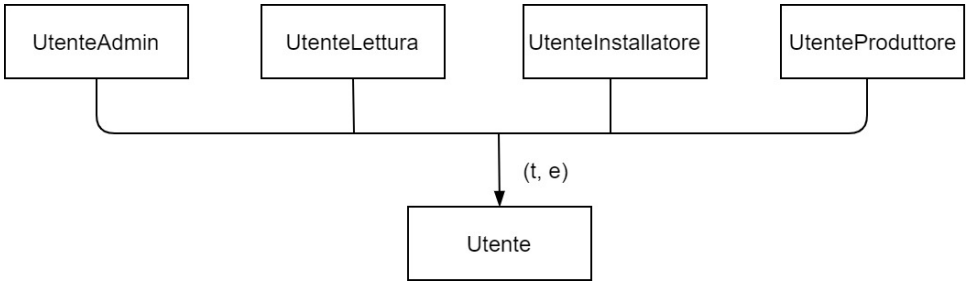


Figura 3.17 – Diagramma ER Gestione Utenti: Associazioni e Cardinalità

Attributi e Chiavi

Nella seguente figura (*Figura 3.18*) sono riportati gli attributi e le chiavi delle entità. Gli attributi delle entità *figlie* sono semplicemente quelli ereditati dall'entità *padre*.

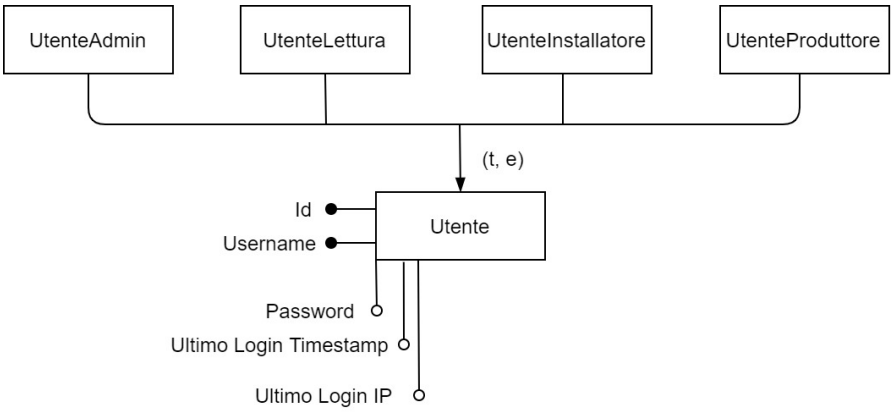


Figura 3.18 – Diagramma ER Gestione Utenti: Attributi e Chiavi

La scelta degli identificatori e degli attributi è stata motivata nel capitolo dedicato all'analisi dei requisiti del database (*Capitolo 3.2*).

È importante specificare che ad ogni produttore ed installatore presente nel sistema è associato un relativo utente, che memorizza le informazioni e le credenziali di accesso all'Applicazione Web del consorzio di filiera. Il diagramma ER sopra mostrato non descrive tali associazioni per motivi di semplicità; questi vincoli saranno adeguatamente illustrati dal Diagramma ER completo (di cui il *Capitolo 3.4.9* è oggetto), che offrirà una visione complessiva di tutto il database e della realtà di interesse.

3.4.8 – Gestione Log

Schema Scheletro

La gestione dei log, descritta nei *Capitolo 3.2.2* come requisito aggiuntivo, identifica sei tipologie di log: Modifiche (alle tabelle del DB), Utente Admin, Utente Lettura, Utente Produttore, Utente Installatore, Scripts.

Viene di seguito riportato lo schema scheletro relativo a questa porzione del database (*Figura 3.19*).

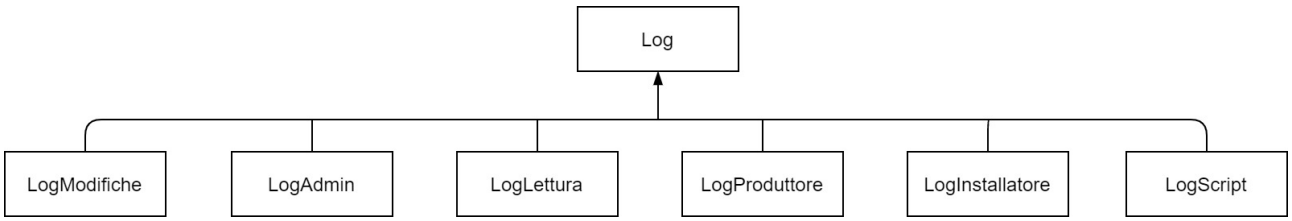


Figura 3.19 – Schema Scheletro Gestione Log

Associazioni e Cardinalità

La gestione dei log individua due differenti macrocategorie di Log:

- **log utenti e script**: memorizzano lo storico delle operazioni effettuate dagli utenti e dall'esecuzione degli script (anche automatica)
- **log modifiche**: memorizza lo storico delle modifiche (e cancellazioni) ai record delle tabelle del database

I log degli amministratori e degli script sono, inoltre, classificati come **log interni**.

Il seguente diagramma ER (*Figura 3.20*) mostra lo schema risultante:

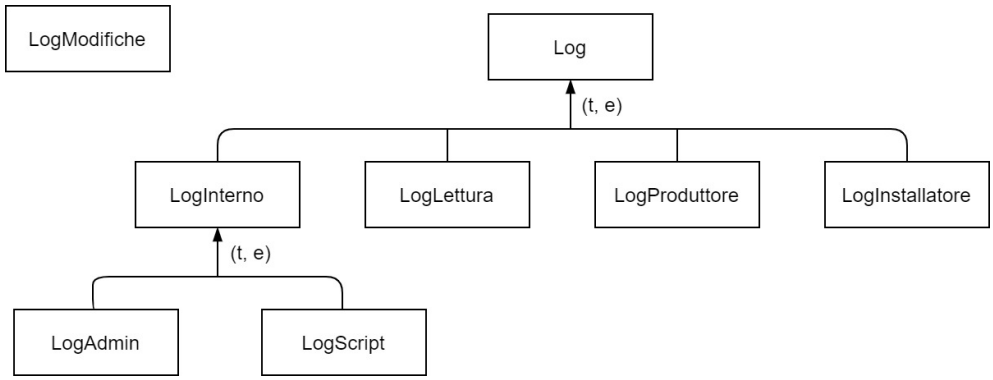


Figura 3.20 – Diagramma ER Gestione Log: Associazioni e Cardinalità

Attributi e Chiavi

Entrambe le generalizzazioni sono di tipo totale (descrivono tutte le tipologie di log presenti nella realtà di interesse) ed *esclusive* (in quanto non è possibile che lo stesso log appartenga, ad esempio, sia ad un produttore che ad un installatore, dato che rappresentano utenti distinti). Gli attributi delle entità *figlie* sono semplicemente quelli ereditati dall'entità *padre*.

L'entità Log Modifiche è stata distinta dai log che memorizzano le generiche operazioni degli utenti e degli scripts in quanto presenta attributi molto differenti ed incompatibili con la generalizzazione (a meno dell'utilizzo di valori NULL e campi selettori, che tuttavia complicherebbero la gestione e potrebbero portare al rischio di errori).

Segue, nella pagina seguente, lo schema ER completo (*Figura 3.20*) relativo alla gestione dei Log. La scelta degli attributi e delle chiavi è stata descritta e motivata nella fase di specifica dei requisiti (*Capitolo 3.2.2*).

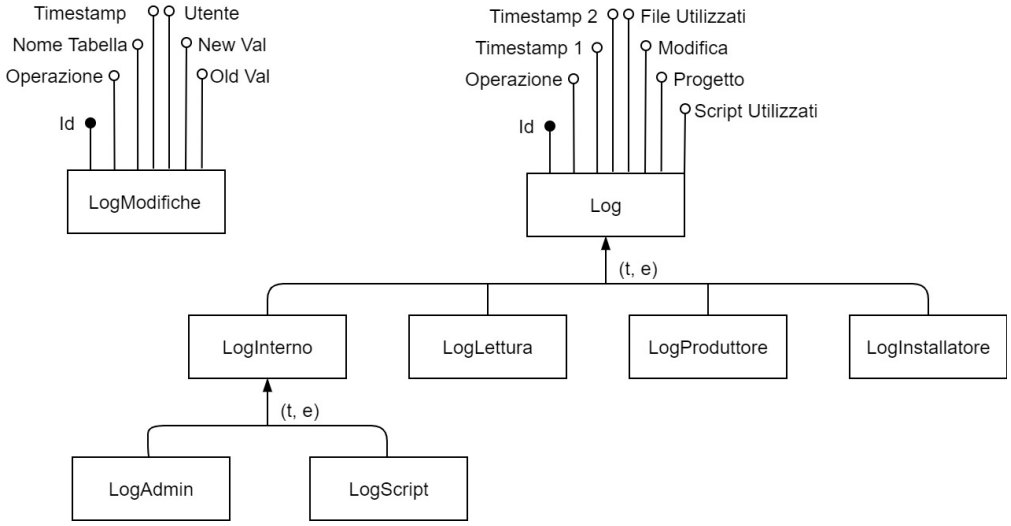


Figura 3.21 – Diagramma ER Gestione Log: Attributi e Chiavi

3.4.9 – Diagramma E/R Completo

Dopo aver analizzato tutte le macro componenti in cui si scompone il database del consorzio di filiera, è stata fondamentale la costruzione di un diagramma ER completo che permettesse una visione di insieme di tutte le parti del DB.

In aggiunta alle tabelle descritte ed analizzate nel corso dei capitoli precedenti, nella fase di Implementazione del database (di cui il *Capitolo 4* sarà oggetto) è stata realizzata un ulteriore tabella necessaria per la memorizzazione di alcuni **parametri di configurazione del database**. La tabella DBConfigurazioni è identificata da un id di tipo testuale e memorizza il valore di una costante con visibilità globale alle *procedure* e *trigger* del database.

Sebbene nel diagramma ER completo, di seguito mostrato, tale tabella non risulta collegata alle altre, ciò non risulta essere un errore poiché i dati in essa contenuta saranno, in realtà, accessibili mediante le *procedure* ed i *trigger*. Segue, brevemente, il diagramma ER dell'entità DB Configurazioni (*Figura 3.22*):

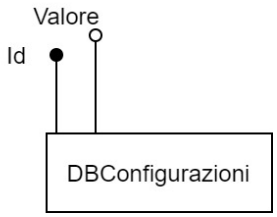


Figura 3.22 – Diagramma ER Gestione Configurazioni DB

Segue il diagramma ER completo del database (*Figura 3.23*):

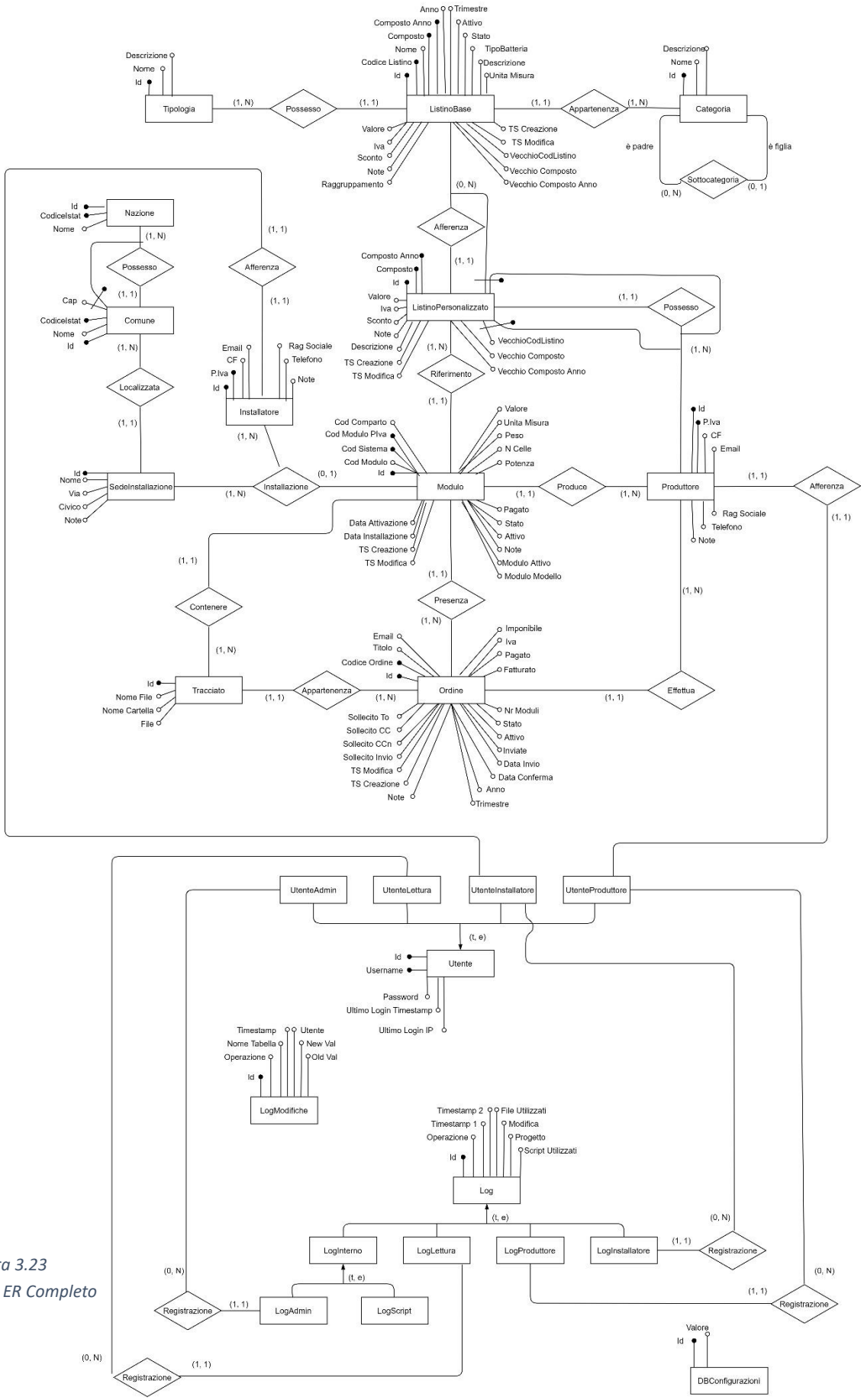


Figura 3.23
Diagramma ER Completo

3.5 – Progettazione Database: Schema Logico

In questa fase della progettazione, si tradurrà lo schema E/R (analizzato nel precedente capitolo – *Capitolo 3.4*) nello schema logico, al fine di procedere con la progettazione fisica. Saranno effettuate scelte dovute a chiarezza concettuale, robustezza infrastrutturale e migliorie prestazionali.

3.5.1 – Eliminazione Gerarchie ISA

Prima di tutto è necessario rimuovere le gerarchie ISA (*Is-A*). Per risolvere questo problema sono possibili le seguenti soluzioni:

- **mantenimento delle entità**: vengono mantenute tutte le entità, le figlie vengono associate al padre e identificate esternamente tramite l'associazione
- **collasso verso l'alto**: le entità figlie vengono rimosse, tutti i loro attributi diventano attributi opzionali per l'entità padre ed è necessario l'utilizzo di selettori per indicare di quale sotto entità si tratta
- **collasso verso il basso**: viene eliminata l'entità padre e tutti gli attributi e le associazioni legate ad essa vengono replicate per ogni entità figlia

È importante sottolineare che tutte le gerarchie coinvolte nel presente progetto sono di tipo *totale* (coinvolgono tutte le casistiche previste dalla realtà di interesse) ed *esclusivo* (le entità figlie sono distinte tra loro); le scelte descritte in seguito hanno tenuto in considerazione questo aspetto.

Gerarchia Log Interno

La gerarchia che sussiste tra le entità Log Interno (*padre*), LogAdmin e Log Script (*figlie*), definisce un raggruppamento logico tra i log riguardanti gli interventi interni – ovvero da parte degli amministratori, eventualmente mediante l'esecuzione di script.

Analizzando lo storico dei Log Interni del database originario, è emerso che le operazioni effettuate dagli utenti amministratori costituiscono solo una piccolissima parte del totale; infatti, i log interni sono principalmente relativa all'esecuzione di script automatici (di backup e di importazione dei tracciati record).

Per tali ragioni, si opta per un collasso verso l'alto, mediante l'aggiunta di un campo TipoLog atto a distinguere le due tipologie. Gli altri attributi dell'entità Log Interno derivano, in realtà, da un'altra gerarchia (sull'entità Log), che verrà analizzata nel seguente paragrafo.

Segue il diagramma ER risultante (*Figura 3.24*):

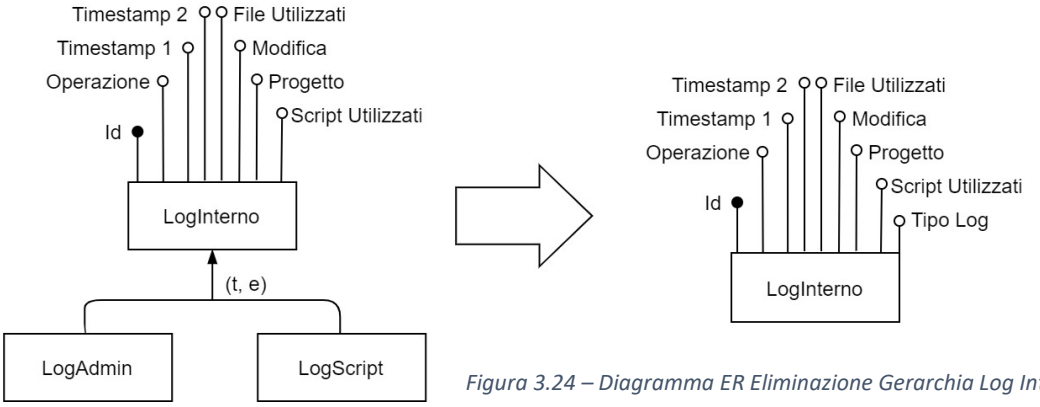


Figura 3.24 – Diagramma ER Eliminazione Gerarchia Log Interno

Gerarchia Log

La gerarchia che sussiste tra le entità Log (*padre*), LogInterno (derivanti dall'eliminazione della gerarchia su tale entità), LogInstallatore, LogProdotture e LogLettura (*figlie*), definisce un raggruppamento logico tra i log riguardante le operazioni effettuate da utenti e script.

Gli attributi delle entità figlie derivano dall'entità padre, Log; le entità figlie non presentano ulteriori attributi.

Appare evidente che il collasso verso il basso è la soluzione più appropriata per questa gerarchia. Segue il diagramma ER risultante (*Figura 3.25*):

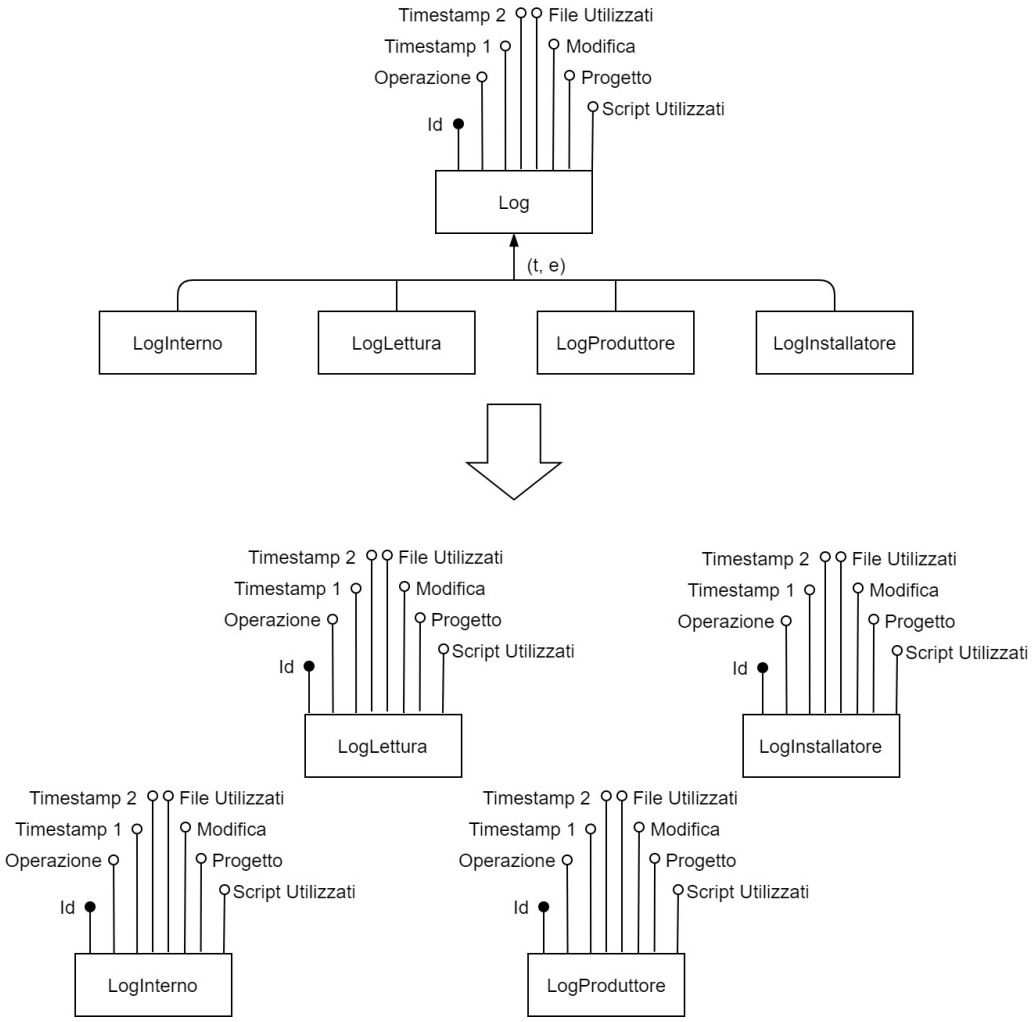


Figura 3.25 – Diagramma ER Eliminazione Gerarchia Log

Gerarchia Utenti

La gerarchia che sussiste tra le entità Utente (*padre*), UtenteAdmin, UtenteLettura, UtenteInstallatore ed UtenteProduttore (*figlie*), definisce un raggruppamento logico tra le tipologie di utenti che possono interagire con il sistema software.

Gli attributi delle entità figlie derivano dall'entità padre, Utente; le entità figlie non presentano attributi aggiuntivi.

Il collasso verso il basso risulta essere la soluzione più appropriata per questa gerarchia. Segue il diagramma ER risultante (*Figura 3.26*):

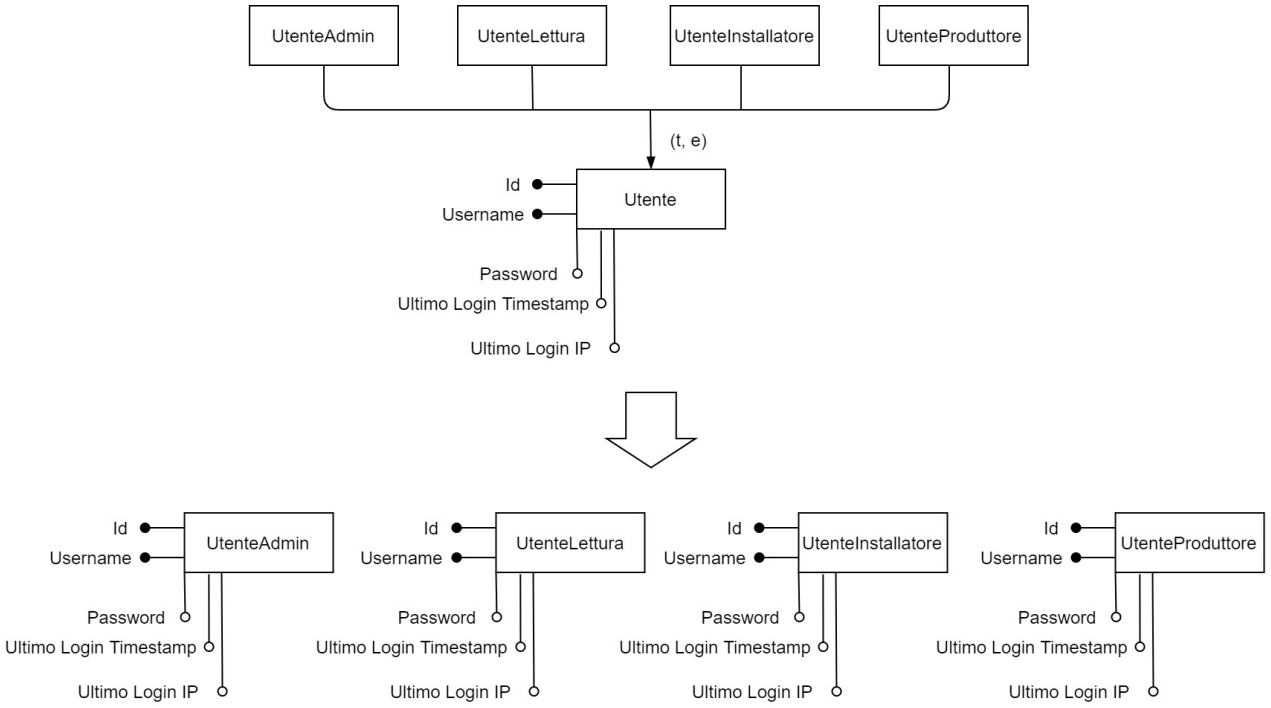


Figura 3.26 – Diagramma ER Eliminazione Gerarchia Utenti

3.5.2 – Selezione delle Chiavi Primarie ed Eliminazione degli Identificatori Esterni

In questa sezione della presente documentazione sono descritte le motivazioni che hanno portato alla scelta delle chiavi primarie per le varie entità.

È importante sottolineare che la scelta degli identificatori primari è stata effettuata prediligendo la scelta di **campi numerici seriali** per **motivi prestazionali**; verranno, in ogni caso, mantenuti gli identificatori scelti durante la fase di analisi dei requisiti (oggetto del *Capitolo 3.2*) e durante la realizzazione del Diagramma ER (*Capitolo 3.4*) poiché permettono agli utenti del sistema di approcciarsi ai dati mediante l'utilizzo di identificatori e nomi più semplici ed espressivi (come i codici di listino – ad esempio, il codice *1_4_2*).

Verranno, di seguito, analizzate solo le entità che presentano più identificatori, chiavi composte ed identificatori esterni.

Categoria e SottoCategoria

L'entità Categoria presenta un'associazione ricorsiva su sé stessa, con i ruoli di Categoria Padre e Categoria Figlia, che descrive le SottoCategorie.

Per motivi prestazionali, e per una esplicita richiesta del committente (Multitraccia), si opta per la scomposizione di Categoria e SottoCategoria in due entità distinte. SottoCategoria avrà gli stessi attributi dell'entità Categoria, più un identificatore esterno che farà riferimento alla Categoria Padre. Questa scelta evita la possibilità della formazione di cicli tra le sottocategorie, ed impone (mediante il vincolo di chiave esterna) che una SottoCategoria possa fare riferimento ad una sola Categoria padre.

Segue lo schema risultante (*Figura 3.27*):

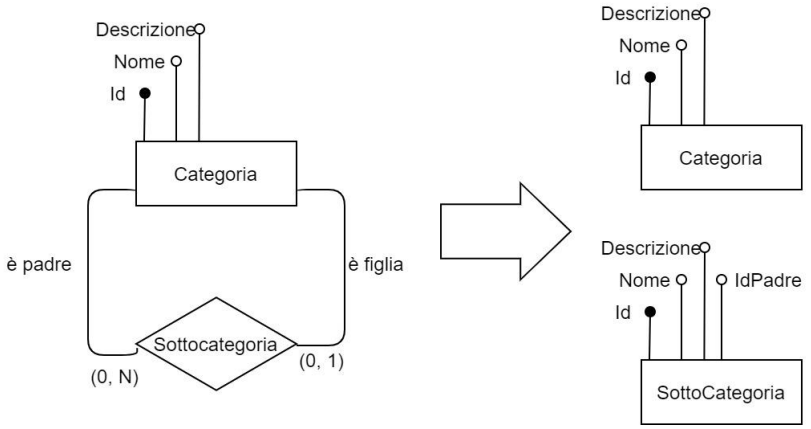


Figura 3.27 – Diagramma ER Chiavi: Categoria e SottoCategoria

---

Listino Base

L'entità Listino Base presenta ben quattro identificatori: un id numerico seriale interno, un codice di listino, un codice composto ed un codice composto anno (di tipo stringa).

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Gli altri tre campi vengono utilizzati dai Produttori e dagli Installatori per identificare i record, e rimarranno quindi campi univoci a valori non nulli.

Segue lo schema risultante (*Figura 3.28*):

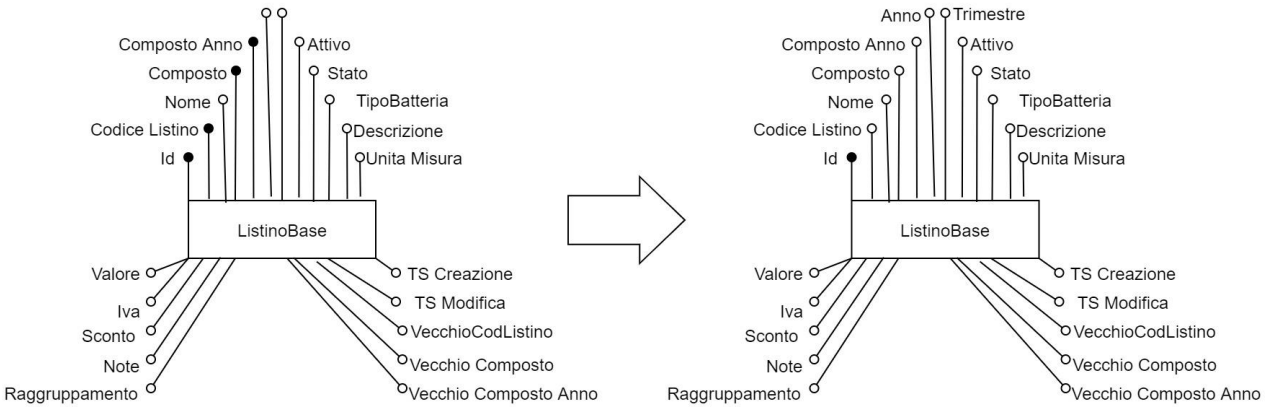


Figura 3.28 – Diagramma ER Chiavi: Listino Base

Produttore

L'entità Produttore presenta due identificatori: un id numerico seriale interno e la P.IVA.

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Il campo P.IVA rimarrà univoco a valori non nulli.

Segue lo schema risultante (*Figura 3.29*):

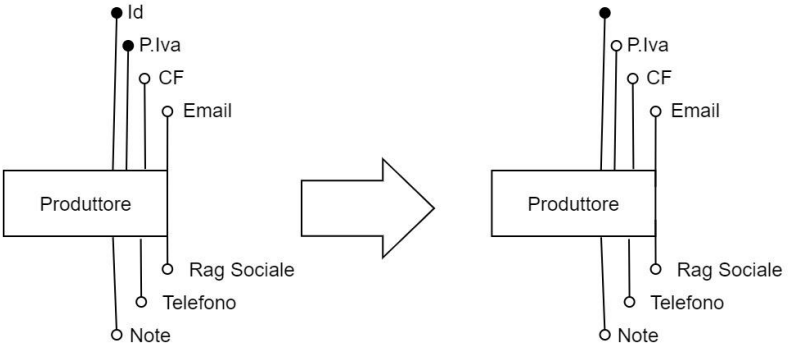


Figura 3.29 – Diagramma ER Chiavi: Produttore

Listino Personalizzato

L'entità Listino Personalizzato presenta ben cinque identificatori: un id numerico seriale interno, un codice composto, un codice composto anno, la composizione degli identificatori delle entità Listino Base e Produttore (chiavi esterne), e la composizione dell'identificatore dell'entità Produttore (chiave esterna) e del Vecchio Codice di Listino.

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Gli altri campi vengono utilizzati dai Produttori e dagli Installatori per identificare i record, e rimarranno quindi campi univoci a valori non nulli.

Segue lo schema risultante (*Figura 3.30*):

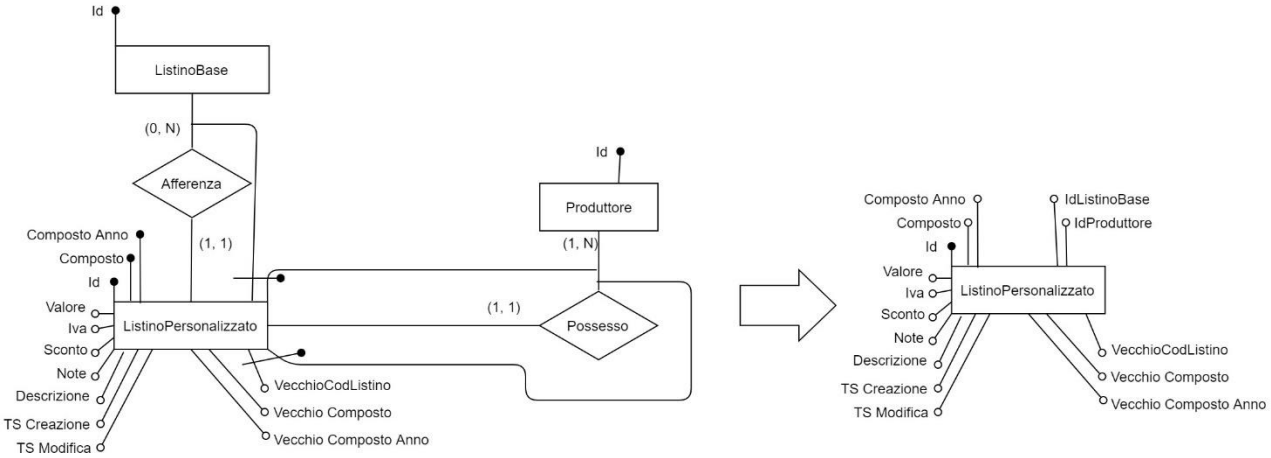


Figura 3.30 – Diagramma ER Chiavi: Listino Personalizzato

Ordine

L'entità Ordine presenta due identificatori: un id numerico seriale interno ed un codice ordine (di tipo stringa) utilizzato da Produttori ed Installatori per identificare gli ordini.

Per ragioni prestazionali, si opta per la scelta del campo id come chiave primaria dell'entità. Il campo codice ordine rimarrà a valore univoco e non nullo.

Di seguito (*Figura 3.31*) lo schema risultante:

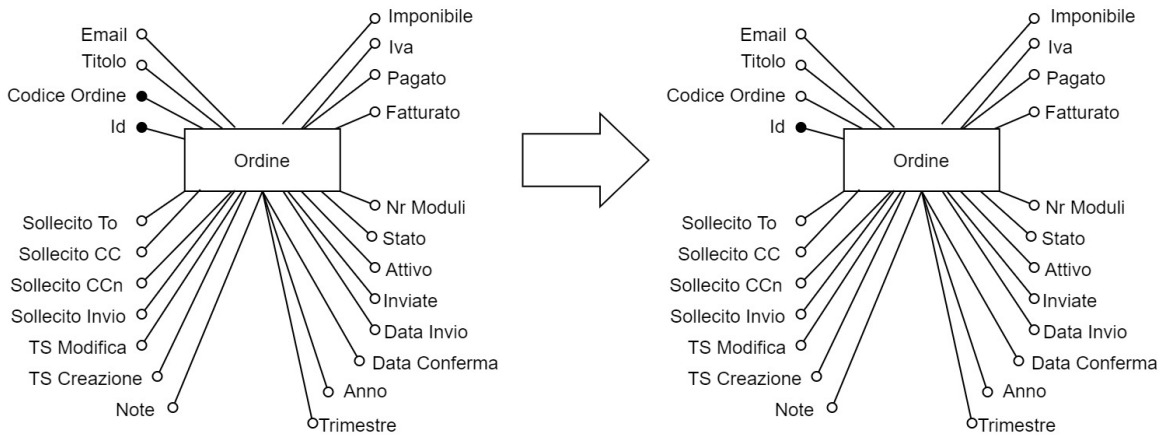


Figura 3.31 – Diagramma ER Chiavi: Ordine

Modulo

L'entità Modulo presenta tre identificatori: un id numerico seriale interno, un codice di sistema ed un codice modulo PIVA.

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Gli altri campi vengono utilizzati dai Produttori e dagli Installatori per identificare i record, e rimarranno quindi campi univoci a valori non nulli.

Segue lo schema risultante (*Figura 3.32*):

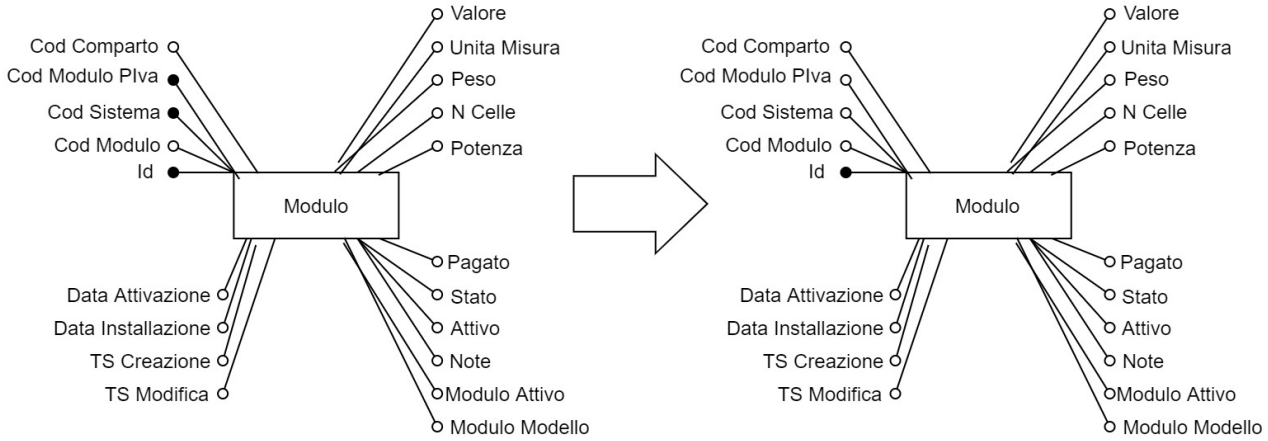


Figura 3.32 – Diagramma ER Chiavi: Modulo

Installatore

L'entità Installatore presenta due identificatori: un id numerico seriale interno e la P.IVA.

Si procede in modo analogo a quanto fatto con l'entità Produttore: si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Il campo P.IVA rimarrà univoco a valori non nulli.

Segue lo schema risultante (*Figura 3.33*):

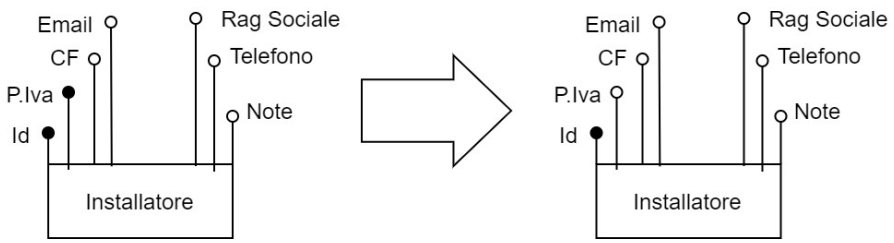


Figura 3.33 – Diagramma ER Chiavi: Installatore

Nazione

L'entità Nazione presenta due identificatori: un id numerico seriale interno ed il codice ISTAT.

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Il campo codice ISTAT rimarrà univoco a valori non nulli.

Segue lo schema risultante (*Figura 3.34*):

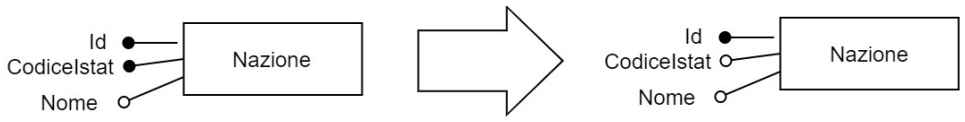


Figura 3.34 – Diagramma ER Chiavi: Nazione

Comune

L'entità Comune presenta tre identificatori: un id numerico seriale interno, il codice ISTAT, ed un identificatore composto dal CAP e dall'identificatore dell'entità Nazione (mediante vincolo di chiave esterna).

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Gli altri campi vengono utilizzati rimarranno campi univoci a valori non nulli.

Segue lo schema risultante (*Figura 3.35*):

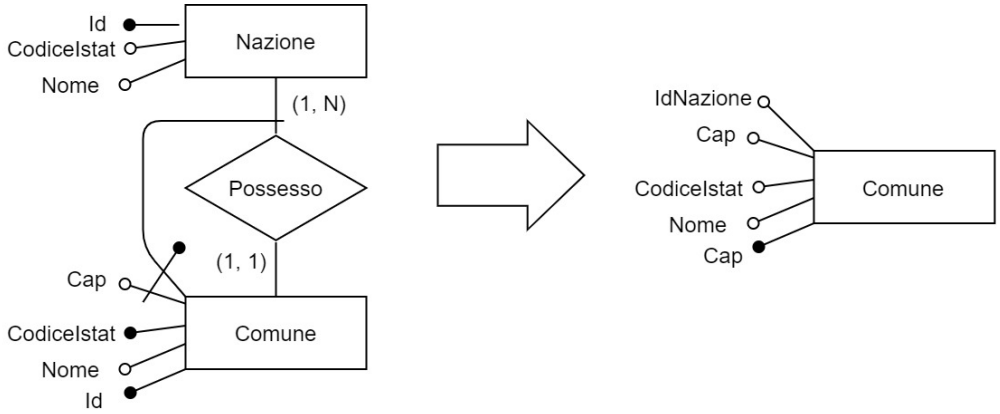


Figura 3.35 – Diagramma ER Chiavi: Comune

Utenti

L'entità Utente Admin, Utente Installatore, Utente Produttore ed Utente Lettura (derivate dal collasso verso il basso della gerarchia sulla tabella Utenti – *Capitolo 3.5.1*) presentano, ciascuna due identificatori: un id numerico seriale interno, ed uno username (di tipo stringa).

Si sceglie di utilizzare l'id come chiave primaria dell'entità, per motivi prestazionali. Il campo username resterà univoco a valori non nulli.

Segue lo schema risultante per l'entità Utente Admin (*Figura 3.36*), del tutto identico per le altre tre entità:

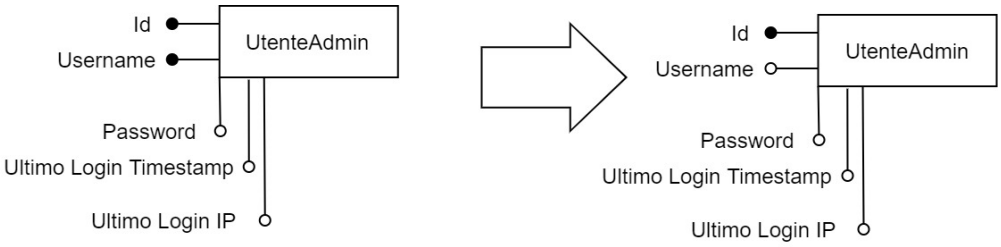


Figura 3.36 – Diagramma ER Chiavi: Utenti

3.5.3 – Trasformazione degli Attributi Multipli e/o Composti

La realizzazione del diagramma E/R non ha richiesto l'utilizzo di attributi multipli o composti. Non sono stati necessari interventi relativi alla loro scomposizione.

3.5.4 – Traduzione di Entità ed Associazioni in Schema Logico

In questa sezione si analizzerà la traduzione effettiva di entità ed associazioni in schema logico. La costruzione dello schema logico è di cruciale importanza per garantire una corretta implementazione del database in SQL.

Si effettueranno scelte per garantire la corretta futura implementazione, la maggiore chiarezza, e, ove possibile, le migliori prestazioni.

La traduzione in schema logico inizia dall'analisi delle entità "*esterne*", ovvero quelle non coinvolte in particolari associazioni e non contenenti vincoli di chiavi esterne, per poi procedere con le restanti.

Entità DBConfigurazioni, LogModifiche, Tipologia, Produttore, Installatore e Nazione

Lo schema logico di queste entità, il cui diagramma ER è riportato nella pagina seguente (*Figura 3.37*) è:

DBConfigurazioni (Id, Valore)

LogModifiche (Id, Operazione, NomeTabella, TImestamp, Utente, NewVal, OldVal)

Tipologia (Id, Nome, Descrizione)

Produttore (Id, P.Iva, CF, Email, RagSociale, Telefono, Note)

AK: P.Iva

Installatore (Id, P.Iva, CF, Email, RagSociale, Telefono, Note)

AK: P.Iva

Nazione (Id, CodiceIstat, Nome)

AK: CodiceIstat

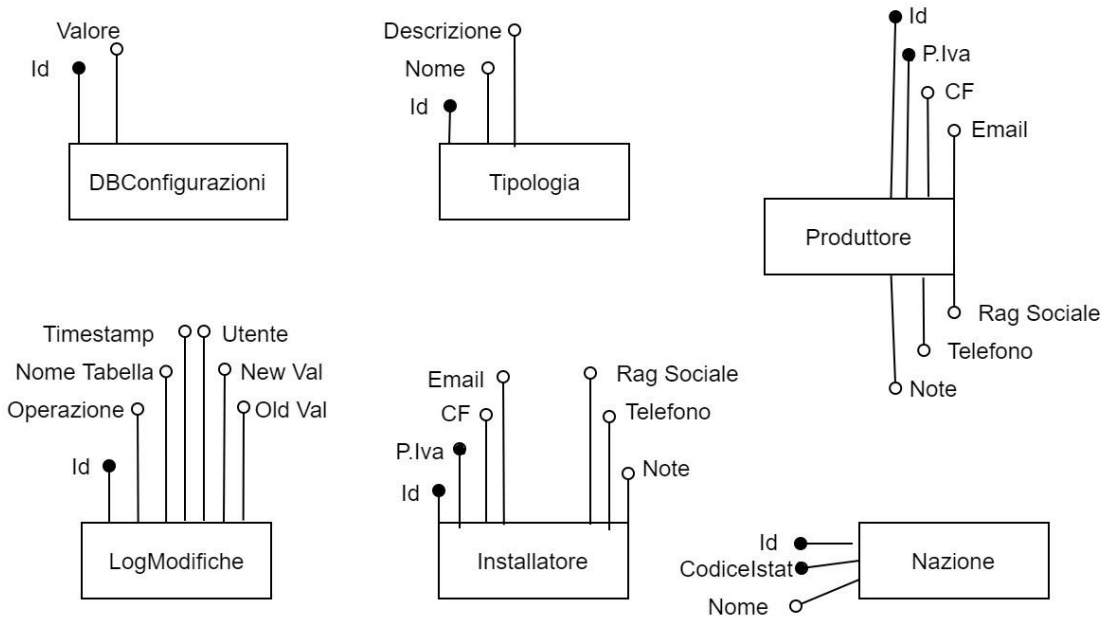
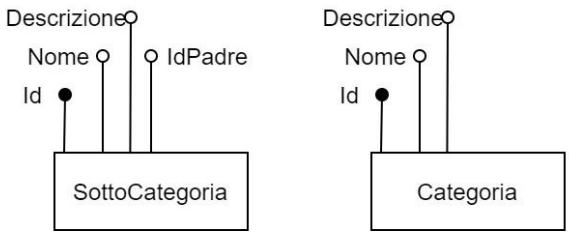


Figura 3.37 – Diagramma ER Traduzione Schema Logico 1

Categoria e SottoCategoria

L'entità Categoria presentava un'associazione ricorsiva su sé stessa, che esprimeva il concetto di sottocategoria. Tale associazione, come analizzato nel *Capitolo 3.5.2*, è stata scomposta in due entità distinte. Segue lo schema logico delle due entità (diagramma ER nella *Figura 3.38*).



Categoria (Id, Nome, Descrizione)

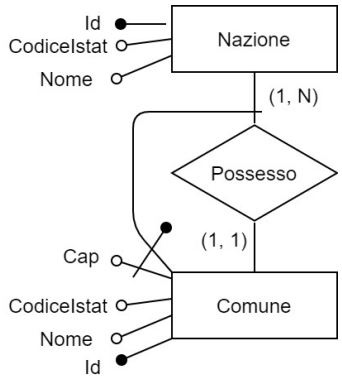
SottoCategoria (Id, Nome, Descrizione, IdPadre)

FK: (IdPadre) REFERENCES Categoria

Figura 3.38 – Diagramma ER Traduzione Schema Logico 2

Comune

L'entità comune riceve la chiave dell'entità Nazione tramite vincolo di foreign key (digramma ER nella *Figura 3.39* seguente). Tale chiave costituisce, insieme al campo CAP, un identificatore secondario per l'entità.



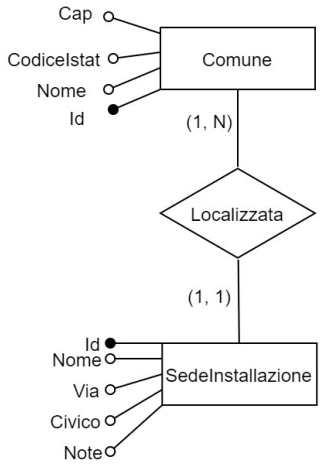
Comune (Id, Nome, CodiceIstat, Cap, IdNazione)

FK: (IdNazione) REFERENCES Nazione
AK: (Cap, IdNazione)
AK: CodiceIstat

Figura 3.39 – Diagramma ER Traduzione Schema Logico 3

SedeInstallazione

L'entità Sede Installazione riceve l'identificatore dell'entità Comune mediante vincolo di chiave esterna (diagramma nella *Figura 3.40* seguente).



SedeInstallazione (Id, Nome, Via, Civico, Note, IdComune)

FK: (IdComune) REFERENCES Comune

Figura 3.40 – Diagramma ER Traduzione Schema Logico 4

Listino Base

L'entità Listino Base riceve, medianti vincoli di chiavi esterne (da associazioni differenti), gli identificatori delle entità Tipologia, Categoria e SottoCategoria (diagramma nella *Figura 3.41* seguente).

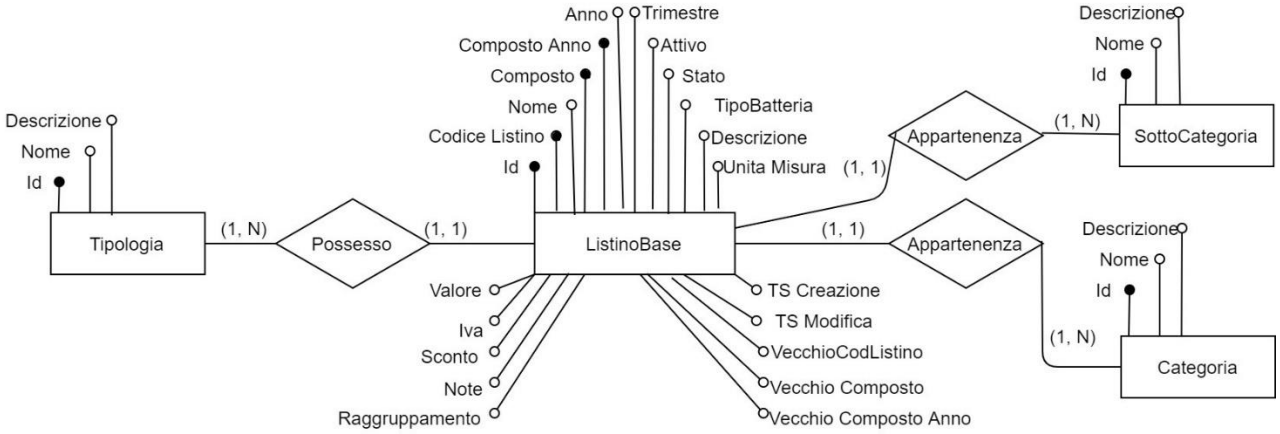


Figura 3.41 – Diagramma ER Traduzione Schema Logico 5

ListinoBase (Id, CodiceListino, Nome, Composto, CompostoAnno, Anno, Trimestre, Attivo, Stato, TipoBatteria, Descrizione, UnitaMisura, Valore, Iva, Sconto, Note, Raggruppamento, TSCreazione, TSModifica, VecchioCodListino, VecchioComposto, VecchioCompostoAnno, IdTipologia, IdCategoria, IdSottoCategoria)

FK: (IdTipologia) REFERENCES Tipologia

FK: (IdCategoria) REFERENCES Categoria

FK: (IdSottoCategoria) REFERENCES SottoCategoria

AK: CodiceListino

AK: Composto

AK: CompostoAnno

Listino Personalizzato

L'entità Listino Personalizzato riceve due identificatori esterni, mediante vincolo di chiave esterna, rispettivamente dalle entità Listino Base e Produttore (diagramma nella *Figura 3.42* seguente).

Gli identificatori delle entità ListinoBase e Produttore costituiscono, insieme, una chiave dell'entità; una seconda chiave è costituita dall'identificatore dell'entità Produttore e l'attributo VecchioCodListino.

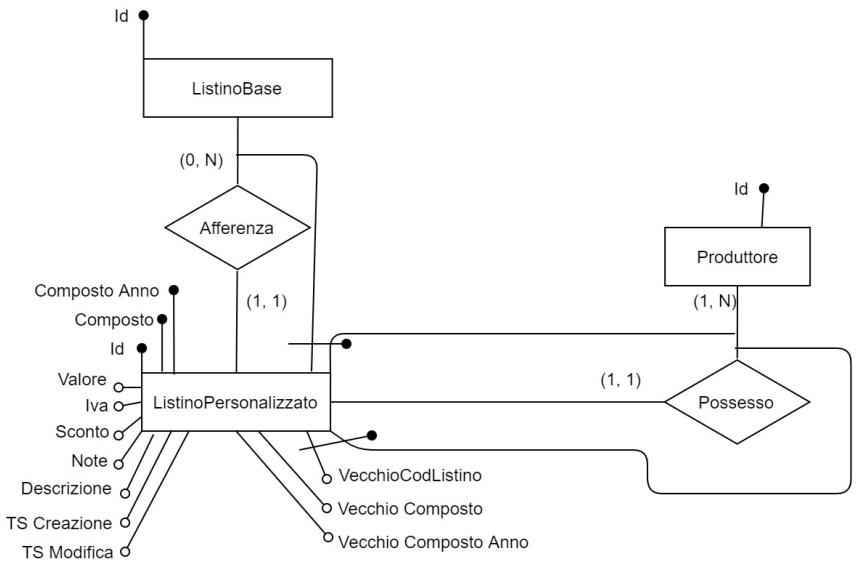


Figura 3.42 – Diagramma ER Traduzione Schema Logico 6

ListinoPersonalizzato (Id, Composto, CompostoAnno, Valore, Iva, Sconto; Note, Descrizione, TS Creazione, TS Modifica, IdProduttore, IdListinoBase, VecchioCodListino, VecchioComposto, VecchioCompostoAnno)

FK: (IdProduttore) REFERENCES Produttore

FK: (IdListinoBase) REFERENCES ListinoBase

AK: Composto

AK: CompostoAnno

AK: (IdProduttore, IdListinoBase)

AK: (IdProduttore, VecchioCodListino)

Ordine

L'entità Ordine riceve, mediante vincolo di chiave esterna, l'identificatore dell'entità Produttore (diagramma nella *Figura 3.43* seguente).

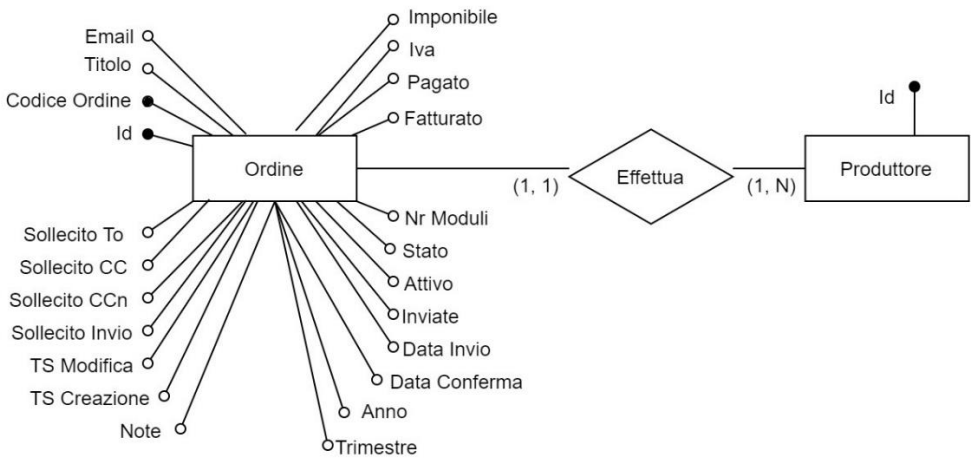


Figura 3.43 – Diagramma ER Traduzione Schema Logico 7

Ordine (Id, CodiceOrdine, Titolo, Email, Imponibile, Iva, Pagato, Fatturato, NrModuli, Stato, Attivo, Inviata, DataInvio, DataConferma, Anno, Trimestre, SollecitoTo, SollecitoCC, SollecitoCCn, SollecitoInvio, TSModifica, TSCreazione, Note, IdProduttore)

FK: (IdProduttore) REFERENCES Produttore

AK: CodiceOrdine

Tracciato

L'entità Tracciato, mediante vincolo di chiave esterna, riceve l'identificatore dell'entità Ordine (diagramma nella *Figura 3.44* seguente).

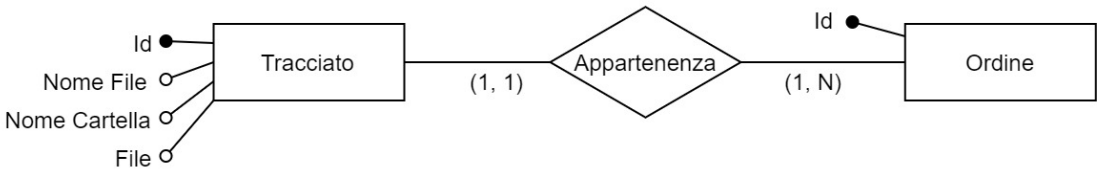


Figura 3.44 – Diagramma ER Traduzione Schema Logico 8

Tracciato (Id, NomeFile, NomeCartella, File, IdOrdine)

FK: (IdOrdine) REFERENCES Ordine

Modulo

L'entità Modulo, mediante vincoli di chiave esterna, riceve l'identificatore delle entità Ordine, Produttore, Tracciato, ListinoPersonalizzato, Installatore e SedeInstallazione (diagramma nella *Figura 3.45* seguente).

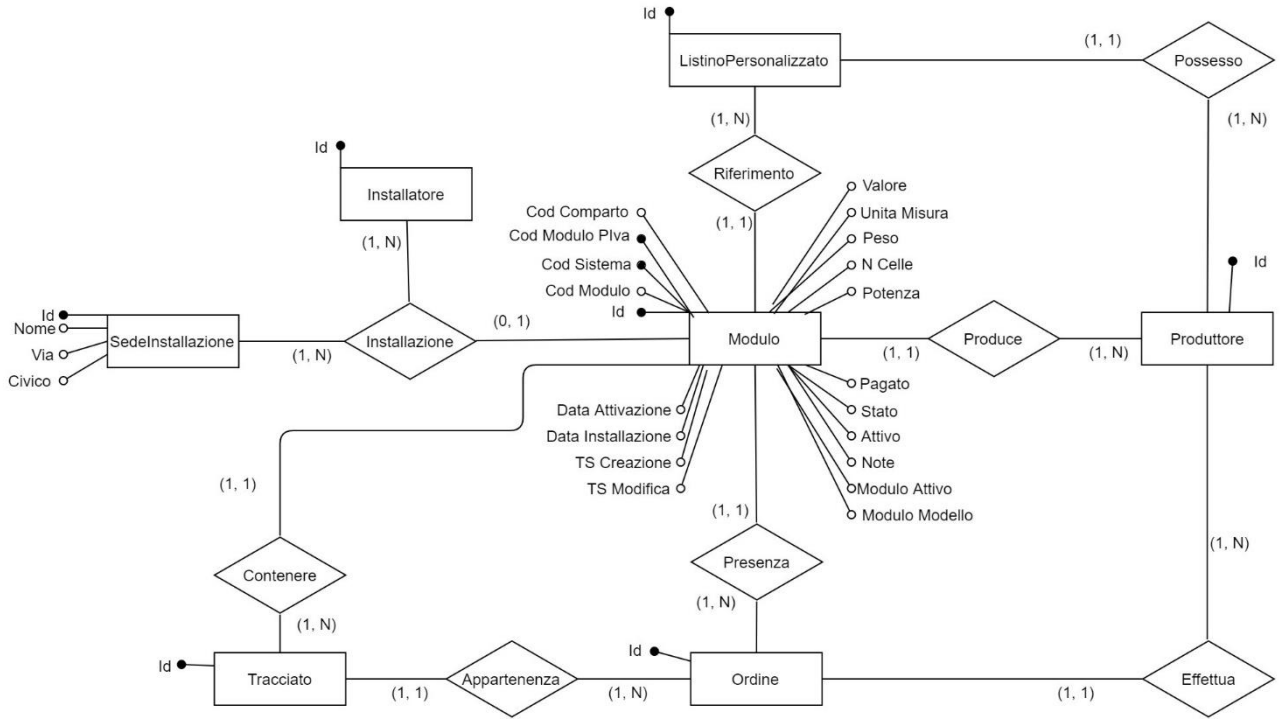


Figura 3.45 – Diagramma ER Traduzione Schema Logico 9

Modulo (Id, CodModulo, CodSistema, CodModuloPIva, CodComparto, Valore, UnitaMisura, Peso, NCelle, Potenza, Pagato, Stato, Attivo, Note, ModuloAttivo, ModuloModello, DataAttivazione, DataInstallazione, TSCreazione, TSModifica, IdProduttore, IdOrdine, IdTracciato, IdInstallatore, IdSedeInstallazione, IdListinoPersonalizzato)

FK: (IdOrdine) REFERENCES Ordine

FK: (IdProduttore) REFERENCES Produttore

FK: (IdTracciato) REFERENCES Tracciato

FK: (IdInstallatore) REFERENCES Installatore

FK: (IdSedeInstallazione) REFERENCES SedeInstallazione

FK: (IdListinoPersonalizzato) REFERENCES ListinoPersonalizzato

AK: CodSistema

AK: CodModuloPIva

UtenteAdmin, UtenteLettura, UtenteInstallatore, UtenteProduttore

Le entità UtenteAdmin, UtenteLettura, UtenteInstallatore ed UtenteProduttore (*Figura 3.46* seguente) possiedono gli stessi attributi e verranno, dunque, analizzate insieme.

Le entità UtenteInstallatore ed UtenteProduttore ricevono, mediante vincoli di chiave esterna, rispettivamente l'identificatore dell'entità Installatore e dell'entità Produttore.

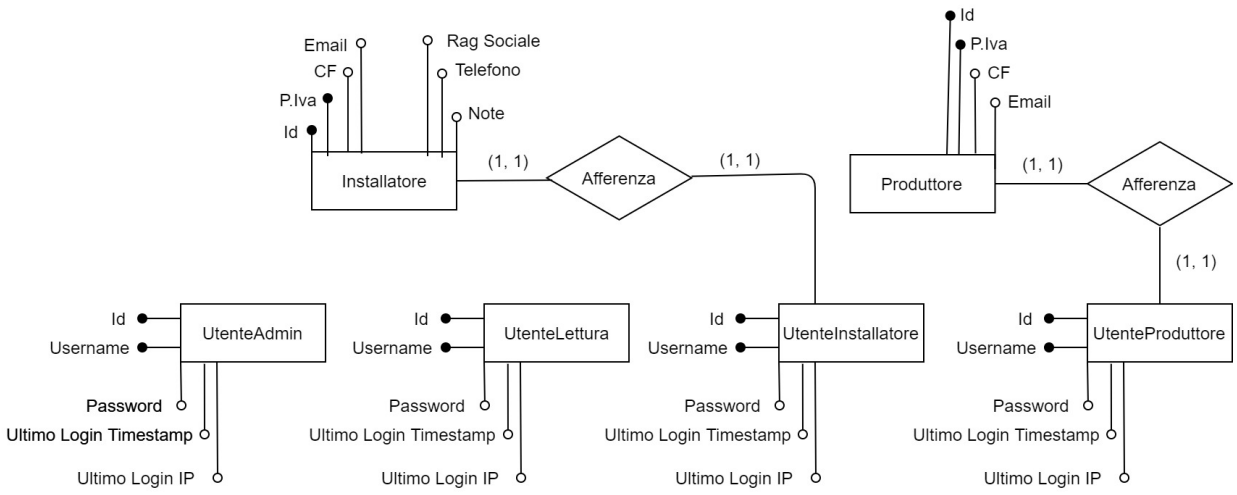


Figura 3.46 – Diagramma ER Traduzione Schema Logico 10

UtenteAdmin / UtenteLettura (Id, Username, Password, UltimoLoginTimestamp, UltimoLoginIP)

UtenteInstallatore (Id, Username, Password, UltimoLoginTimestamp, UltimoLoginIP, IdInstallatore)

FK: (IdInstallatore) REFERENCES Installatore

AK: Username

UtenteProduttore (Id, Username, Password, UltimoLoginTimestamp, UltimoLoginIP, IdProduttore)

FK: (IdProduttore) REFERENCES Produttore

AK: Username

LogInterno, LogProduttore, LogInstallatore, LogLettura

Le entità LogInterno, LogProduttore, LogInstallatore, LogLettura (*Figura 3.47* seguente) possiedono gli stessi attributi e verranno, dunque, analizzate insieme.

Le entità LogProduttore e LogInstallatore ricevono, mediante vincoli di chiave esterna, rispettivamente l'identificatore dell'entità Produttore e dell'entità Installatore.

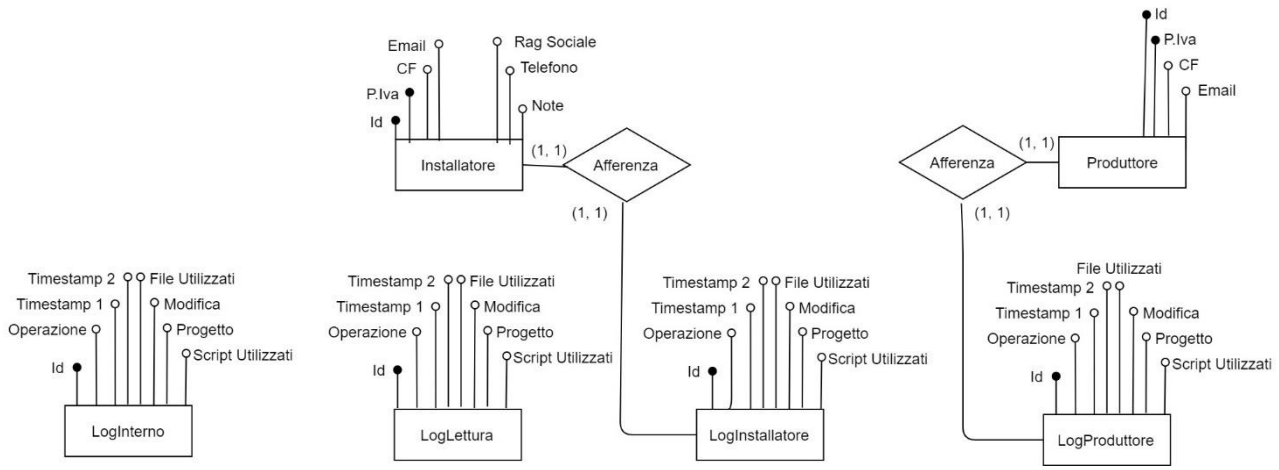


Figura 3.47 – Diagramma ER Traduzione Schema Logico 11

LogInterno / LogLettura (Id, Operazione, Timestamp1, Timestamp2, FileUtilizzati, Modifica, Profetto, ScriptUtilizzati)

LogInstallatore (Id, Operazione, Timestamp1, Timestamp2, FileUtilizzati, Modifica, Profetto, ScriptUtilizzati, IdInstallatore)

FK: (IdInstallatore) REFERENCES Installatore

LogProduttore (Id, Operazione, Timestamp1, Timestamp2, FileUtilizzati, Modifica, Profetto, ScriptUtilizzati, IdProduttore)

FK: (IdProduttore) REFERENCES Produttore

3.5.5 – Verifica della Normalizzazione

A seguito della verifica della normalizzazione, non è stato ritenuto necessario apportare modifiche.

3.6 – Progettazione Sito Web

La realizzazione di una porzione del sito web del consorzio di filiera appartiene ai requisiti aggiuntivi (descritti nel *Capitolo 3.2.2*) emersi nel corso del mio tirocinio presso Multitraccia. Per motivazioni esclusivamente tempistiche, è stata progettata (e realizzata) solo una porzione del nuovo sito web, al fine di mostrare un "modello" di interfacciamento alla base di dati.

Sono state realizzate due aree del sito, accessibili ai Produttori dei moduli fotovoltaici:

- **area di dichiarazione contributi ambientali**: dichiarazione dei moduli e dei relativi contributi ambientali
- **area report**: visualizzazione di report trimestrali relativi ai moduli dichiarati

Seguono l'analisi e la progettazione di tali aree del sito.

3.6.1 – Analisi dei Requisiti e del Sito Esistente

Multitraccia ha richiesto la realizzazione di due aree per il nuovo sito web, dedicate alla **dichiarazione dei contributi ambientali dei moduli** ed alla **visualizzazione di report trimestrali** riguardanti tali dichiarazioni; entrambe le aree sono **accessibili** esclusivamente **dai Produttori** dei moduli fotovoltaici, previa autenticazione al sistema.

L'autenticazione degli utenti, tuttavia, non rientra nello scopo di questo progetto (su scelta di Multitraccia), per cui non è stata gestita (se non predisponendo controlli minimali).

Entrambe le pagine web hanno dovuto replicare le funzionalità e l'aspetto delle aree del sito preesistente del consorzio. Per ragioni di riservatezza, Multitraccia non mi ha consentito di mostrare le pagine originali del sito, per cui ne verranno soltanto descritte le funzionalità.

Area Dichiarazione Contributi Ambientali

Questa Area consente l'accesso a due pagine, che consentono la generazione ed il caricamento dei tracciati dei moduli.

La prima pagina ha lo scopo di costituire un **Help Online per la generazione dei tracciati**, che consenta ai Produttori di comprendere le linee guida per la realizzazione dei tracciati nonché la loro struttura; sarà inoltre possibile generare veri e propri tracciati mediante un form interattivo.

La seconda pagina consente il **caricamento dei tracciati moduli nel sistema**, le cui modalità di gestione saranno analizzate nel *Capitolo 3.6.3*.

Area Report

Questa Area consente l'accesso ai report dei moduli dichiarati. In particolare, è prevista la realizzazione delle seguenti pagine:

- **Report Generale Dichiarazioni**: elenca le dichiarazioni (anche dette ordini) effettuate; permette, inoltre, la visualizzazione del contenuto delle singole dichiarazioni
- **Estratto Conto Trimestrale**: permette la visualizzazione di report trimestrali delle dichiarazioni effettuate
- **Tracciati Moduli Caricati**: elenca i tracciati caricati; permette la visualizzazione del contenuto dei singoli tracciati
- **Mappa Moduli Attivati**: sezione la cui realizzazione è prevista in futuro e che consentirà la visualizzazione di una mappa interattiva dei moduli installati ed attivati

3.6.2 – Progetto dell'Interfaccia

La progettazione dell'interfaccia del sito web è risultata di grande semplicità, in quanto ha dovuto banalmente replicare quella del sito preesistente. Tuttavia, per motivi di riservatezza, non mi è stato concesso di inserire nel documento screenshot del sito originale; nel *Capitolo 4.4* verranno mostrate le nuove schermate.

La struttura delle pagine del sito web è la seguente (*Figura 3.48*):

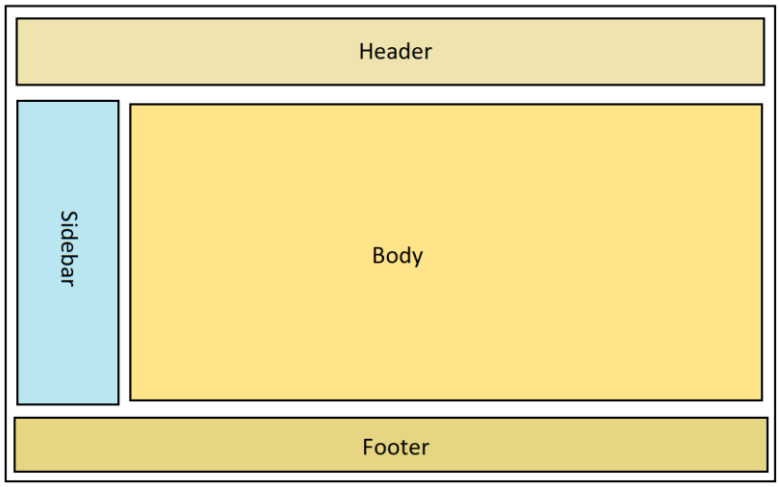


Figura 3.48 Struttura Pagina Web del Sito

Le pagine web realizzate costituiscono solo la parte centrale di tale struttura, ovvero il *body*. Non è stata prevista la realizzazione delle altre porzioni della pagina in quanto si prevede l'integrazione del body all'interno del *tema* del futuro sito web del consorzio.

3.6.3 – Gestione del Caricamento dei Tracciati dei Moduli Fotovoltaici

Il caricamento dei tracciati dei moduli fotovoltaici è una delle parti centrali del funzionamento del sistema, e risulta di enorme importanza la sua corretta gestione.

Insieme al mio tutor aziendale, Francesco Gregori, sono state analizzate **due modalità per la gestione del caricamento dei tracciati**: **caricamento diretto** e **caricamento a fine giornata**.

Caricamento Diretto

La prima modalità di gestione analizzata permette di inserire i record del tracciato (relativi alle dichiarazioni dei contributi ambientali dei moduli fotovoltaici) direttamente nel database. Al termine del caricamento, al Produttore sarà inviata una **ricevuta** in diretta.

È prevista una breve **validazione del file in locale**, mediante funzioni JavaScript, al fine di controllarne il corretto formato ed estensione prima dell'invio del file al server.

Questa modalità di gestione, più semplice, è stata la scelta di gestione per cui si è optato inizialmente (seppur consci del rischio di lunghi tempi di caricamento); tuttavia ha mostrato dei limiti importanti durante la fase di test (oggetto del *Capitolo 5.3*).

Caricamento a Fine Giornata

La seconda modalità di gestione permette un caricamento differito dei record dei tracciati.

I file dei tracciati possono essere caricati nella fascia oraria di apertura del sito (8:00 – 18:00) e vengono salvati sul server; a fine giornata, i file vengono letti sequenzialmente ed i record sono immessi nel database. L'invio delle ricevute di caricamento, ai Produttori, avviene solo a fine giornata.

È chiaramente prevista una modalità di validazione locale del file analoga a quella descritta per la gestione mediante il caricamento diretto.

Questa modalità di caricamento era quella adottata dal sito preesistente del consorzio di filiera. Per il presente progetto, è stata inizialmente preferita la gestione del caricamento in diretta al fine di rendere possibile l'invio in diretta delle ricevute di caricamento (che, con la gestione a fine giornata, chiaramente venivano mandate solo dopo l'orario di chiusura del sito). Tuttavia, a seguito di test con dati reali (oggetto del *Capitolo 5.3*), la scelta è stata rivalutata.

Capitolo 4

Implementazione

4.1 – Introduzione

Questo capitolo ha l'obiettivo di descrivere le fasi che hanno costituito l'implementazione del progetto, sulla base di quanto analizzato e descritto nel precedente capitolo dedicato alla progettazione del sistema (*Capitolo 3*).

La prima parte della progettazione è dedicata all'implementazione della base di dati in PostgreSQL, con la creazione delle tabelle, degli indici, e delle procedure di controllo ed automazione.

La parte successiva descriverà il processo di migrazione dei dati dal database originario (con DBMS FileMaker). Saranno mostrate le procedure di esportazione dei dati ed analizzate le problematiche riscontrate in fase di importazione dei dati (causate dall'inconsistenza ed incoerenza di alcuni record del DB originario).

Infine, l'ultima parte di questo capitolo illustrerà la realizzazione dell'applicazione web e della sua interfaccia, analizzando tutte le funzionalità implementate.

4.2 – Implementazione Database in PostgreSQL

L'implementazione del Database in PostgreSQL è stato un passo cruciale per la corretta realizzazione del progetto di cui la presente tesi è oggetto.

L'implementazione del database si basa sullo schema logico analizzato e realizzato nel precedente *Capitolo 3.5*. L'implementazione è suddivisa in tre fasi: la prima riguarda la creazione delle tabelle e dei domini dei dati, la seconda la creazione delle procedure per i controlli sui dati inseriti e per la generazione dei valori di alcuni campi, mentre l'ultima fase riguarda la creazione degli indici sugli attributi delle tabelle.

In questo capitolo verranno mostrati solo esempi cruciali al fine di descrivere la sintassi di PostgreSQL. Il codice completo è inserito nell'*Appendice* del documento.

4.2.1 – Creazione delle Tabelle

La creazione delle tabelle è stata la prima – e la più importante – fase dell'implementazione della base di dati.

Per prima cosa, sono stati creati un utente admin per il database ed il database stesso, mediante i seguenti comandi (*Figura 4.1*):

```
/* Creazione di un utente amministratore per il database */
DROP USER IF EXISTS multitracciaadmin;
CREATE USER multitracciaadmin SUPERUSER PASSWORD 'root';

/* Creazione del database ed assegnamento del proprietario */
DROP DATABASE IF EXISTS ftv_db;
CREATE DATABASE ftv_db OWNER multitracciaadmin;
```

Figura 4.1 – Comandi Implementazione DB 1

Il passo successivo è stato legato alla creazione dei domini dei dati, al fine di semplificare la scrittura dei tipi dei campi usati più frequentemente nelle tabelle, nonché per specificare particolari formati.

Il seguente comando (*Figura 4.2*) mostra creazione di un dominio "di base":

```
CREATE DOMAIN DomNome
    AS varchar(128);
```

Figura 4.2 – Comandi Implementazione DB 2

Il seguente comando (*Figura 4.3*) mostra creazione di un dominio con restrizione sui valori assumibili dal campo:

```
CREATE DOMAIN DomTipoLog
    AS varchar(6)
    CHECK (VALUE IN ('Script', 'Admin'));
```

Figura 4.3 – Comandi Implementazione DB 3

Il seguente comando (*Figura 4.4*) mostra creazione di un dominio con controllo sui valori mediante espressione regolare:

```
CREATE DOMAIN DomCodiceListino
    AS varchar(8)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}$'); /* 1_2_13 */
```

Figura 4.4 – Comandi Implementazione DB 4

Si procede, dunque, alla creazione vera e propria delle tabelle della base di dati; su richiesta di Multitraccia, i campi rappresentati attributi veri e propri della tabella useranno la **nomenclatura** *a_nome*, mentre quelli riguardanti campi di controllo (es. i timestamp) avranno la nomenclatura *z_nome*.

Nella seguente figura (*Figura 4.5*) è mostrata la creazione di due tabelle: la seconda tabella, SottoCategoria, presenta una chiave primaria definita su due attributi e specifica, inoltre, un attributo come chiave esterna.

```
CREATE TABLE Categoria /* ok */
(
    a_id_categoria smallserial PRIMARY KEY,
    CHECK (a_id_categoria > 0),
    a_nome         DomNome UNIQUE NOT NULL,

    a_descrizione  DomDescrizione
);

CREATE TABLE SottoCategoria /* ok */
(
    a_id_sottocategoria     smallint,
    CHECK (a_id_sottocategoria > 0),
    a_nome                  DomNome UNIQUE NOT NULL,

    a_descrizione           DomDescrizione,
    a_id_categoria_padre_fk smallint,

    PRIMARY KEY (a_id_sottocategoria, a_id_categoria_padre_fk),
    FOREIGN KEY (a_id_categoria_padre_fk) REFERENCES Categoria (a_id_categoria)
        ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

Figura 4.5 – Comandi Implementazione DB 5

4.2.2 – Creazione delle Procedure

Successivamente alla creazione delle tabelle e dei domini dei dati, è stata di fondamentale importanza la definizione di *stored procedure* mediante l'utilizzo di *trigger*.

Le stored procedure vengono utilizzate per effettuare molteplici controlli ed operazioni sulle tabelle ed i suoi dati, ad esempio:

- **Controllo della validità delle chiavi esterne**: ad esempio, un Modulo, associato al suo Produttore, deve appartenere ad un Ordine dello stesso Produttore
- **Composizione di campi:** ad esempio per la costruzione dei codici di listino, mediante la concatenazione dell'id della categoria, della sottocategoria e della tipologia a cui il record del listino afferisce
- **Salvataggio Log:** salvataggio automatico dei log a seguito di operazioni effettuate sulle tabelle e la base di dati (ad esempio eliminazione o modifica di record)

Segue un esempio di procedura mediante trigger per la generazione di alcuni campi della tabella Listino Base *Figura 4.6*, nella pagina seguente). Si evidenziano: la creazione della funzione (procedura - (costrutto tipico dei linguaggi di programmazione), la dichiarazione di una variabile locale ed infine la creazione del trigger che si attiverà (ed eseguirà la sua relativa procedura) prima dell'inizio di operazioni che riguardano l'inserimento o la modifica di record per la tabella.

```
/* T2) All'aggiunta di un record nel listino base, i campi codice listino, composto e composto anno vengono generati,
   rispettivamente come:
   - id_categoria_sottocategoria
   - id_categoria_sottocategoria
   - id_categoria_sottocategoria_anno_trimestre */
CREATE OR REPLACE FUNCTION insert_listino_base()
RETURNS TRIGGER AS $insert_listino_base$
    DECLARE
        local_id_tipologia smallint = 0;
    BEGIN

        IF NEW.a_id_tipologia_fk IS NOT NULL
        THEN
            local_id_tipologia = NEW.a_id_tipologia_fk;
        END IF;
        NEW.a_codice_listino = concat(NEW.a_id_categoria_fk, '_', NEW.a_id_sottocategoria_fk, '_', local_id_tipologia);
        NEW.a_composto = NEW.a_codice_listino;
        NEW.a_composto_anno = concat(NEW.a_codice_listino, '_', NEW.a_anno, '_', NEW.a_trimestre);

        RETURN NEW;
    END;
$insert_listino_base$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_listino_base /* in caso esista già */
ON listinobase;

CREATE TRIGGER insert_listino_base
BEFORE INSERT OR UPDATE ON listinobase
FOR EACH ROW EXECUTE PROCEDURE insert_listino_base();
```

Figura 4.6 – Comandi Implementazione DB 6

Si elencano le procedure realizzate:

1. **Sottocategoria** – genera l'id seriale all'inserimento / modifica di un record
2. **Listino Base** – genera i campi *codice listino, composto* e *composto anno* all'aggiunta / modifica di un record
3. **Listino Personalizzato** – genera i campi *composto* e *composto anno* all'aggiunta / modifica di un record; inoltre, si verifica che i timestamp di creazione sia successivo al rispettivo valore del record del listino base a cui si fa riferimento (mediante chiave esterna)
4. **Ordine** – all'inserimento di un Ordine, il campo *IVA* viene popolato con il valore attuale ottenuto dalla tabella di configurazioni *DBConfigurazioni*
5. **Ordine** – aggiornamento del valore del campo seriale in caso di inserimento manuale di un record (al fine di ripristinare la corretta numerazione ed evitare la generazione di duplicati e di un conseguente errore); generazione del *codice ordine*
6. **Ordine** – generazione del codice ordine in caso di update ad un record della tabella
7. **Modulo** – si controllano che il listino personalizzato e l'ordine a cui afferisce (mediante chiavi esterne) facciano riferimento allo stesso Produttore (indicato mediante l'apposita chiave esterna); si verifica che il timestamp di creazione del record sia successivo a quello dell'ordine
8. **Modulo** – Calcolo del totale dell'ordine all'inserimento dei moduli che lo compongono, ed incremento del contatore del numero dei moduli (campo della tabella ordine)

9. **Modulo** – Aggiornamento del totale dell'ordine e del contatore del numero dei moduli alla rimozione di un record dalla tabella modulo
10. **Modulo** – Aggiornamento del *totale* dell'ordine in caso di modifica del *valore* (prezzo del contributo ambientale) dei moduli che lo compongono; questo caso non si dovrebbe mai verificare in quanto il valore è ottenuto, al momento dell'inserimento del record, dal relativo listino di riferimento
11. **Log** – Registrazione dello storico delle operazioni sulle tabelle del db; si memorizzano le operazioni di modifica e cancellazione di record (ed i valori di tali record)

Non sono stati gestiti i log delle operazioni degli utenti in quanto riguardano una porzione dell'applicazione web che non è stata realizzata.

4.2.3 – Creazione degli Indici

La creazione degli indici sui campi delle tabelle risulta essere di fondamentale importanza al fine di garantire le prestazioni del database (in particolar modo considerando che la tabella Modulo conterrà oltre 4 milioni di record).

È importante sottolineare che la creazione degli indici dovrà avvenire soltanto a seguito dell'importazione dei dati dal database originario: infatti, questo garantirà una strutturazione più efficiente (e compatta) degli indici, nonché un minor *overhead* computazionale durante l'importazione.

Inoltre, è bene precisare che PostgreSQL crea automaticamente gli indici per i campi di tipo *unique* (dunque, anche per i campi *primary key*); questo avviene poiché lo storage engine di PostgreSQL garantisce l'univocità di tali campi proprio mediante l'utilizzo degli indici.

Sono stati creati gli indici sugli attributi più importanti delle tabelle, al fine di garantire l'efficienza delle operazioni di ricerca dei dati (per esempio, dei codici seriali dei moduli fotovoltaici – i quali non sono univoci). La selezione dei campi da indicizzare ha seguito, principalmente, le scelte adottate da Multitraccia nel database originario.

La seguente figura (*Figura 4.7*) illustra il comando per la creazione di un indice su un campo di una tabella:

```
CREATE INDEX produttore_index_a_cf ON Produttore (a_cf);
```

Figura 4.7 – Comandi Implementazione DB 7

4.3 – Migrazione dei Dati da FileMaker

Il sistema software realizzato ha lo scopo futuro di sostituire l'attuale sistema del consorzio di filiera. È dunque di vitale importanza la corretta esportazione dei dati dal database originario (con DBMS FileMaker) e la conseguente importazione nel nuovo sistema, al fine di preservare l'intero storico della dichiarazione dei contributi ambientali relativi ai moduli fotovoltaici e le installazioni di questi ultimi.

I prossimi capitoli descriveranno le procedure di esportazione ed importazione dei dati, e le scelte effettuate per la risoluzione di alcuni problemi di importazione garantendo, però, di non perdere dati.

4.3.1 – Esportazione dei Dati

La prima fase della migrazione dei dati ha riguardato l'esportazione dei dati dal database originario (che usa il DBMS FileMaker).

FileMaker (nella versione utilizzata – *Pro 10*) offre la possibilità di esportare le tabelle del database nei seguenti formati (*Figura 4.8* nella pagina seguente):

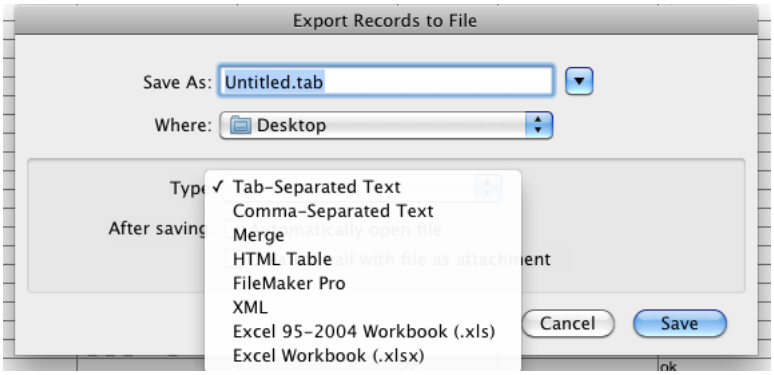


Figura 4.8 – Formati Esportazione di FileMaker

Il primo formato scelto è quello di Microsoft Excel (*.xls*). Tuttavia, tale foglio di calcolo prevede un **numero massimo di righe** pari a 65.536 (fino alla versione 2004) e pari a 1.048.576 per le versioni più recenti (formato *.xlsx*).

Appare evidente che si tratta di un numero di righe tutt'altro che sufficiente per consentire l'esportazione della tabella contenente i moduli fotovoltaici con oltre 4 milioni di record (FileMaker non permette la selezione dei record da esportare oppure l'esportazione su più file, bensì tronca il contenuto delle tabelle al raggiungimento della lunghezza massima consentita dal formato selezionato).

Per tale ragione, si è successivamente optato per l'**utilizzo di file CSV** (*Comma-Separated Text*) che hanno un limite di righe ben superiore.

4.3.2 – Script di Importazione

È stato successivamente realizzato uno script che permettesse l'importazione dei dati esportati nel nuovo database.

Per la sua realizzazione è stato scelto l'utilizzo del linguaggio Python in quanto, non essendo tipizzato (è tipizzato solo debolmente, ma comunque rientra nella categoria dei linguaggi non tipizzati), risulta essere uno strumento molto comodo per la manipolazione di dati di tipo eterogeneo; inoltre, Python è stato scelto in quanto è un linguaggio ampiamente utilizzato per la prototipazione e per la scrittura di codice in maniera rapida (cosa fondamentale visto il ristretto tempo a disposizione).

Lo script ha utilizzato le seguenti librerie:

- **Pandas**: per la lettura dei file XLS e CSV
- **Numpy**: per la corretta manipolazione e gestione dei tipi numerici
- **Psycopg2**: per la connessione alla base di dati su PostgreSQL e l'interazione con essa

4.3.3 – Risoluzione dei Problemi nei Dati Esportati

Durante l'importazione dei dati sono emersi tutti i problemi di inconsistenza ed incoerenza dei dati (descritti nel *Capitolo 3.4.1*).

Infatti, sono stati rilevati:

- **campi non valorizzati**
- **campi erroneamente valorizzati**: es. il valore "200W" nel campo numerico relativo alla potenza dei moduli fotovoltaici (che non dovrebbe contenere il carattere 'W')
- **formato dei timestamp differente** in alcuni campi
- **differente spaziatura, case e punteggiatura** nei nomi di categorie, sottocategorie e tipologia

Tali problemi sono stati risolti analizzando attentamente i record ed implementando minuziose procedure di controllo e conversione dei dati (facendo anche uso di espressioni regolari).

Per alcuni casi, tuttavia, è stato necessario l'intervento del mio tutor aziendale, Francesco Gregori. Infatti, alcuni record riguardanti l'anagrafica dei produttori non erano valorizzati (riportando solo la P.IVA, o CF) ed è dunque stato necessario risalire ai dati corretti mediante il confronto con Multitraccia.

4.4 – Implementazione Sito Web

L'implementazione del sito web ha costituito l'ultima fase dell'implementazione del progetto.

Sono state realizzate alcune pagine del sito su modello di quelle del sito esistente del consorzio di filiera, con lo scopo di costituire un esempio per l'interfacciamento alla base di dati costruita. Infatti, il sito web del consorzio è in fase di completo rinnovamento e sarebbe, quindi, stato superfluo realizzare pagine che probabilmente sarebbe stato necessario riprogettare.

Premessa: gli screenshot che saranno mostrati nei seguenti capitoli sono in toni di grigi e con parti censurate in quanto riproducono lo stile ed i contenuti del sito originale del consorzio di filiera; per motivi di privacy, non mi è consentito mostrare parti riconducibili al nome del consorzio.

4.4.1 – Pagine per la Generazione ed il Caricamento dei Tracciati Fotovoltaici

La prima area del sito realizzata è dedicata alla generazione ed al caricamento dei tracciati dei moduli fotovoltaici, da parte dai produttori che intendono dichiarare il versamento dei contributi ambientali.

Il seguente screenshot (*Figura 4.9*) mostra la pagina principale della suddetta area.

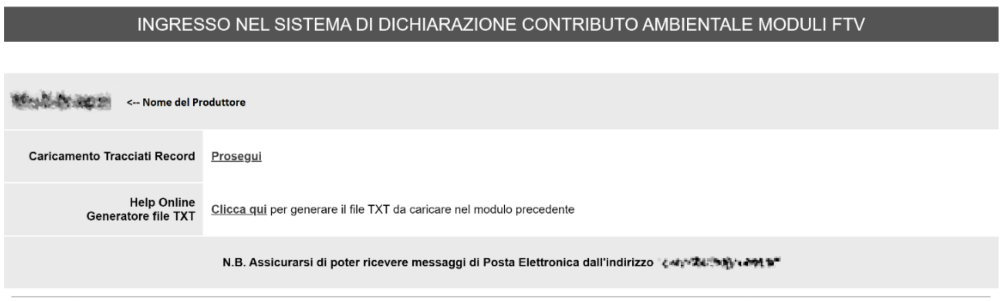


Figura 4.9 – Home Area Dichiarazione Moduli

Da questa pagina è possibile accedere alla sezione dedicata al caricamento dei tracciati di moduli fotovoltaici. Un form permette il caricamento del file, in formato TXT, che sarà validato localmente mediante funzioni Javascript ed inviato al server mediante metodo *POST*.

Segue uno screenshot della pagina per il caricamento dei tracciati (*Figura 4.10*).

Figura 4.10 – Caricamento Tracciati Moduli

È, inoltre, possibile accedere ad un Help Online (*Figura 4.11* seguente) che illustra il formato dei file per i tracciati record, e permette anche la creazione di un tracciato di prova direttamente dalla pagina. La tabella del form è stata realizzata mediante l'utilizzo della libreria *DataTables* [32] ed è gestita mediante *Ajax* e *jQuery*.

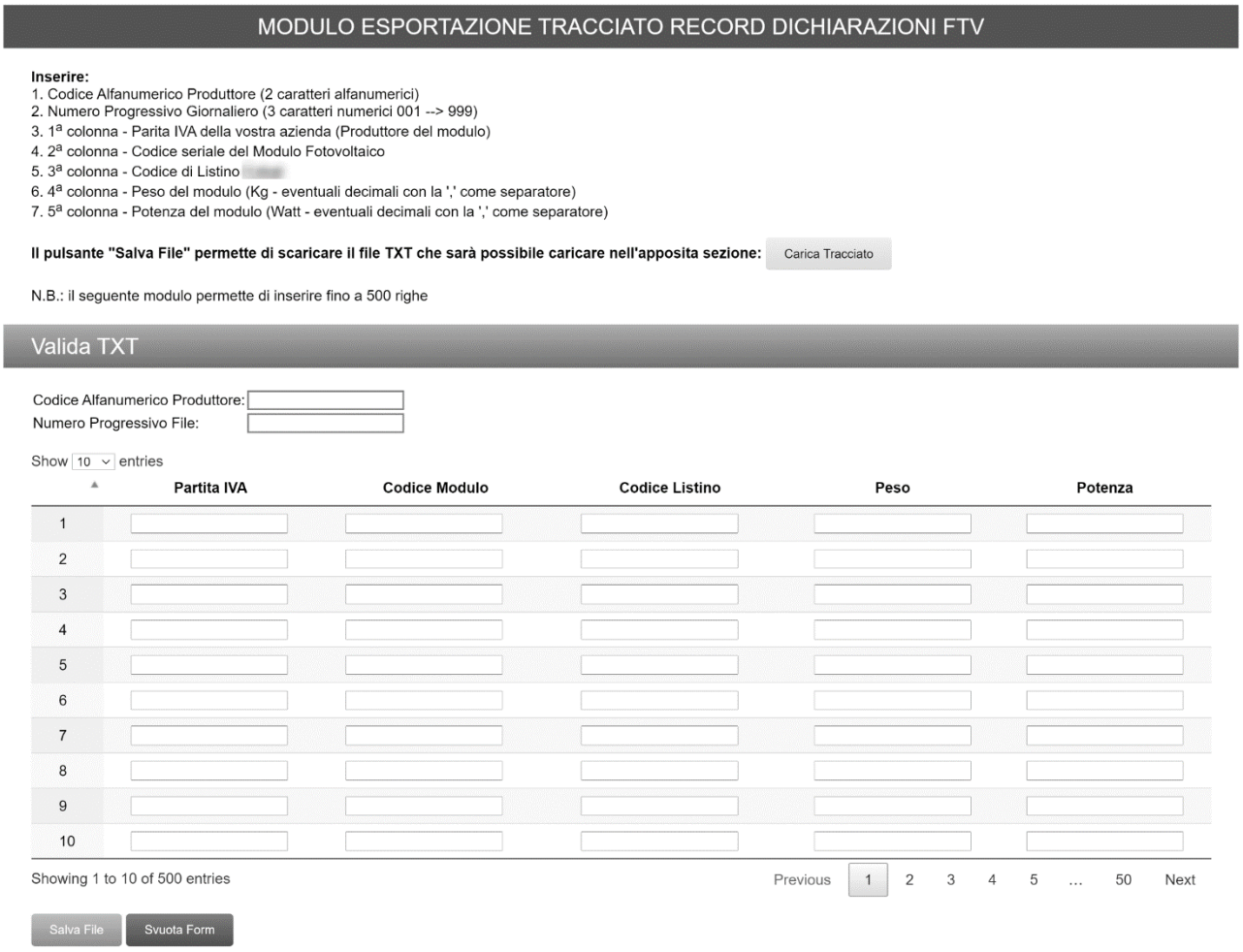


Figura 4.11 – Help Online Creazione Tracciati Moduli

4.4.2 – Gestione Generazione ed Invio delle Ricevute di Caricamento dei Tracciati Fotovoltaici

La prima implementazione del sito web, come descritto dettagliatamente nel *Capitolo 3.6.3*, prevedeva il caricamento in diretta dei tracciati record all'interno del database. Tuttavia, a seguito di prove sperimentali sui tempi di caricamento dei tracciati (che verranno adeguatamente commentate nel *Capitolo 5.3*), è stato deciso di abbandonare questa gestione e di implementare un caricamento differito (a fine giornata) dei tracciati. Di conseguenza, la generazione delle ricevute è cumulativa ed avviene solo a fine giornata.

Mediante l'esecuzione di uno script scritto in linguaggio Python, vengono letti sequenzialmente i file dei tracciati record - organizzati in cartelle, una per produttore. Al termine del caricamento dei tracciati di un produttore, viene generata una ricevuta di caricamento che sarà inviata, tramite email, al produttore.

Nella ricevuta vengono indicati: l'importo totale dell'ordine, il numero di moduli caricati, e gli eventuali codici seriali di record che non è stato possibile caricare (ad esempio per dati in un formato errato).

Segue (*Figura 4.12*) il facsimile di una ricevuta:

```
[illegible] - INVIO CALCOLO DICHIARAZIONE MODULI FOTOVOLTAICI

RAGIONE SOCIALE: [illegible]
Codice Fiscale: [illegible]
Partita IVA: [illegible]

CODICE SISTEMA: FTV00[illegible] del 09/05/2020
PERIODO: 2/2020

Riepilogo Dichiarazione:
Nr. Moduli: 18
Imponibile Euro: 157,80

----------------------------------------
[illegible] - www.[illegible]
```

Figura 4.12 – Facsimile Ricevuta di Caricamento

4.4.3 – Pagine per la Consultazione dei Tracciati Fotovoltaici Caricati

Le seguenti pagine permettono ai produttori di consultare i tracciati caricati nel corso degli anni, nonché di visualizzare i dettagli dei singoli moduli dichiarati (come il codice di listino a cui afferiscono, le categorie, tipologie, ed alcune specifiche tecniche del pannello fotovoltaico).

Sono previste due tipologie di pagine per la consultazione dei tracciati:

- **elenco di tutti i tracciati record caricati** (*Figura 4.13*); possibilità di **eliminare i tracciati** non ancora inseriti nel database (all'interno dell'orario di apertura giornaliero del sito web)
- **visualizzazione dei moduli di un tracciato** e dei relativi dettagli (*Figura 4.14*)

Nella pagina seguente seguono gli screenshot delle schermate.

TRACCIATI MODULI CARICATI

Codice Fiscale	
Partita IVA	

Tracciati Moduli Caricati

Nome File	Nr. Dichiarazione Codice Sistema	Data Caricamento	Elimina Tracciato
2.txt	Non disponibile	05/10/2020	✖ Rimuovi
tracciato_test_1000.txt >>	FTV00043462020 >>	05/10/2020	

Torna all'Area Report

Figura 4.13 – Schermata Tracciati Caricati

DETTAGLIO TRACCIATO tracciato_test_1000.txt

Codice Fiscale	
Partita IVA	

Dettaglio Tracciato tracciato_test_1000.txt

Seriale Modulo	Codice Listino	Categoria	Sottocategoria	Tipologia	Valore	Sconto	Potenza [W]	Peso [Kg]	Note
0	1_1_3				2.14	5.00	4.00	3.00	
1	1_1_3				2.14	5.00	4.00	3.00	
2	1_1_3				2.14	5.00	4.00	3.00	
3	1_1_3				2.14	5.00	4.00	3.00	
4	1_1_3				2.14	5.00	4.00	3.00	

Figura 4.14 – Schermata Dettaglio Tracciato

4.4.4 – Pagine per la Visualizzazione dei Report

La seconda area principale del sito è dedicata alla consultazione dei report relativi alle dichiarazioni dei contributi ambientali da parte dei produttori (*Figura 4.15* nella pagina seguente).

Questa area permette di accedere alle seguenti sezioni:

- **report generale dichiarazioni** (*Figura 4.16)*: elenco delle dichiarazioni (ordini) effettuate nel tempo, che è possibile consultare singolarmente
- **mappa moduli attivati**: sezione non implementata che permetterà, in futuro, di visualizzare una mappa interattiva dei moduli installati ed attivati
- **estratto conto trimestrale** (*Figura 4.17)*: permette di visualizzare l'estratto conto relativo ad uno specifico trimestre, nonché l'elenco dei moduli dichiarati in tale periodo
- **tracciati moduli caricati** (descritta nel precedente capitolo): elenco dei tracciati caricati nel tempo

AREA REPORT DICHIARAZIONI FOTOVOLTAICI

Codice Fiscale	[illegible]
Partita IVA	[illegible]

Report Generale Dichiarazioni	Prosegui
Mappa Moduli ATTIVATI	Prosegui
Estratto Conto Trimestrale	Anno: 2020 Trimestre: 4 Prosegui
Tracciati Moduli Caricati	Prosegui

Figura 4.15 – Home Area Report Dichiarazioni

REPORT GENERALE DICHIARAZIONI

Codice Fiscale	[illegible]
Partita IVA	[illegible]

Riepilogo Dichiarazioni Moduli Fotovoltaici Inviate

Nr. Dichiarazione Codice Sistema	Data Dichiarazione	Data Invio	Imponibile	Nr. Moduli	Pagato	Fatturato	Stato
FTV00043462020 >>	05/10/2020	05/10/2020	2140.00	1000	No	No	Aperto

Torna all'Area Report

Figura 4.16 – Schermata Report Generale Dichiarazioni

ESTRATTO CONTO TRIMESTRE 4 2020

Codice Fiscale	[illegible]
Partita IVA	[illegible]

Estratto Conto 4 2020

Seriale Modulo	Codice Listino	Categoria	Sottocategoria	Tipologia	Valore	Sconto	Potenza [W]	Peso [Kg]	Note
0	1_1_3	[illegible]	[illegible]	[illegible]	2.14	5.00	4.00	3.00	
1	1_1_3	[illegible]	[illegible]	[illegible]	2.14	5.00	4.00	3.00	
2	1_1_3	[illegible]	[illegible]	[illegible]	2.14	5.00	4.00	3.00	
3	1_1_3	[illegible]	[illegible]	[illegible]	2.14	5.00	4.00	3.00	
4	1_1_3	[illegible]	[illegible]	[illegible]	2.14	5.00	4.00	3.00	

Figura 4.17 – Schermata Estratto Conto Trimestrale

Capitolo 5

Test

5.1 – Introduzione

Questo capitolo ha l'obiettivo di descrivere i test effettuati sul sistema implementato, con lo scopo di verificarne il corretto funzionamento nonché le sue prestazioni.

I test effettuati, infatti, hanno permesso di comprendere meglio alcune **criticità del sistema software** (in modo particolare la gestione del caricamento dei tracciati). Per questa ragione, sono state progettate ed intraprese strade alternative al fine di eliminare/ridurre tali problematiche.

Premessa: a causa della situazione di emergenza sanitaria in corso durante lo svolgimento del mio tirocinio (e tuttora in corso nel momento della scrittura del presente documento), **non è stato possibile integrare il software realizzato** nei sistemi di Multitraccia e del consorzio di filiera. Per questa ragione, **tutti i test sono stati effettuati sulla mia macchina** ed i risultati ottenuti potrebbero differire rispetto a quelli ottenibili sul futuro hardware per il quale questo software è destinato; nonostante ciò, **i test mi hanno permesso di stimare l'ordine di grandezza** dei tempi richiesti per l'esecuzione di alcune operazioni, consentendomi di effettuare un'analisi ed una correzione dei punti critici del sistema.

5.2 – Descrizione del Sistema di Test

Come descritto nel precedente capitolo introduttivo (*Capitolo 5.1*), non è stato possibile integrare il sistema software realizzato nei sistemi di Multitraccia e del consorzio di filiera. I test sono stati eseguiti sulla mia macchina.

Si elencano le caratteristiche della mia macchina, sulla quale sono stati effettuati i test del software realizzato:

- Processore: Intel Core i7 4770K (Overclock con frequenza a 4.6GHz)
- RAM: 24GB DDR3 2000Mhz

- Scheda Madre: ASUS Z97-K
- Scheda Video: Nvidia GTX 1080TI
- Storage: **SSD** Sabrent NVME (PCI-E 3.0) 1TB
- Sistema Operativo: **Windows 10** versione 1903

Per quanto riguarda il software utilizzato per i test:

- Browser: Google Chrome
- DBMS: **PostgreSQL versione 9.3**
- Server PHP: mediante XAMPP (versione 3.2.4) e **Server Apache** (versione 2.4.43)
- Python: **Python versione 3.8**

Al fine di rendere i risultati i più veritieri possibili, tutti i test sono stati eseguiti a seguito dell'importazione dei dati dal database originario (mediante gli script descritti nel *Capitolo 4.3*). La tabella *Modulo*, dunque, conteneva oltre 4.000.000 di record.

5.3 – Caricamento Tracciati Fotovoltaici

Il test sul caricamento dei tracciati è stato il test di importanza maggiore ed ha consentito di confrontare le due modalità di gestione del caricamento dei tracciati fotovoltaici (analizzate nel *Capitolo 3.6.3*).

A seguito dei test è emerso che la modalità di caricamento scelta (caricamento *in diretta*) offre prestazioni tutt'altro che sufficienti, e sono dunque state effettuate scelte per migliorare le performance del sistema.

5.3.1 – Modalità di Caricamento dei Tracciati Fotovoltaici a Confronto

Come descritto nel *Capitolo 3.6.3* dedicato alla progettazione dell'applicazione web, sono emerse due differenti modalità per la gestione del caricamento dei tracciati dei moduli fotovoltaici.

La prima modalità permette il **caricamento in diretta dei tracciati**: dopo una breve validazione in locale del file per verificarne l'integrità ed il formato (mediante l'utilizzo di funzioni JavaScript), il tracciato verrà inviato al server e processato immediatamente. Il produttore potrà ricevere in tempo reale l'esito del caricamento del tracciato, nonché la sua ricevuta.

La seconda modalità permette di **processare i tracciati a fine giornata**: dopo una breve validazione in locale del file (come per la prima modalità descritta), il tracciato verrà inviato al server e memorizzato all'interno di una cartella remota (saranno utilizzate cartelle distinte per i differenti produttori). I tracciati dei produttori **verranno processati soltanto dopo l'orario di chiusura del sito web** (ore 18.00), in maniera sequenziale, e solo a questo punto verrà generata la ricevuta.

5.3.2 – Test di Caricamento

Al fine di simulare casi d'uso tipici e reali del sistema, sono stati realizzati dei tracciati di prova, con un numero di record prefissato, al fine di determinare l'ordine di grandezza del tempo necessario al processamento dei tracciati caricati.

La stima del tempo è stata utile solo per la gestione dei caricamenti in diretta, in quanto solo in questo caso il produttore deve attendere il caricamento (sullo schermo sarà visualizzata una progress bar). Il caricamento a fine giornata, invece, non ha vincoli di questo genere.

Sono stati generati, e caricati, tracciati con i seguenti numeri di record (seriali): 1000, 10.000, 100.000. Si elencano i tempi di caricamento (grafico nella *Figura 5.1* seguente):

- 1000 record: caricamento immediato
- 10.000 record: 3-4 secondi
- 100.000 record: oltre 1 minuto

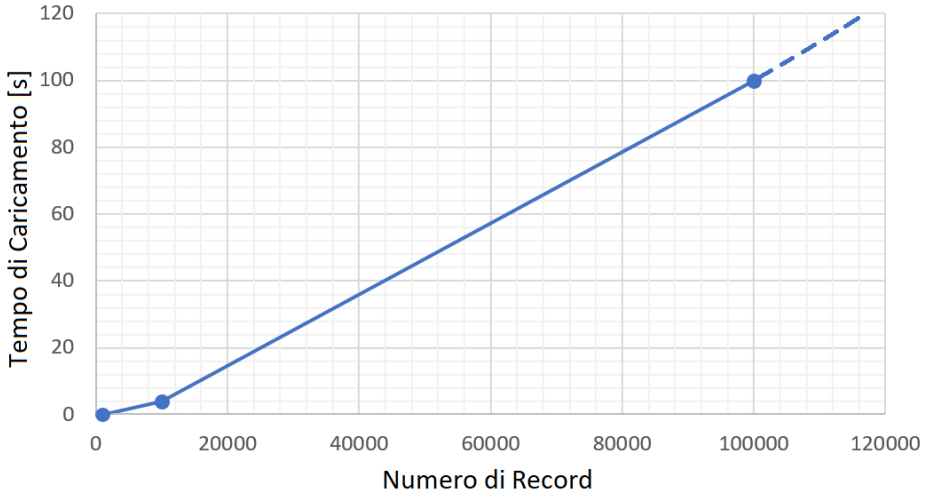


Figura 5.1 – Grafico Tempo Caricamento Tracciati

5.3.3 – Considerazioni e Modifiche alla Gestione del Caricamento dei Tracciati Fotovoltaici

I test di caricamento hanno evidenziato una latenza accettabile per tracciati con un numero di record con ordine di grandezza minore/uguale a 10.000, ma tutt'altro che tollerabile per tracciati con 100.000 record.

Infatti, il produttore rischierebbe di **attendere interi minuti** prima che il tracciato venga caricato, **senza la possibilità** di cambiare pagina; un crash del browser, il refresh della pagina oppure la chiusura di quest'ultima potrebbero portare a risultati del tutto indesiderati ed imprevisti.

Prima di scartare a priori la modalità di caricamento in diretta, si è cercato di comprendere le cause della latenza. Il colpevole più probabile individuato è rappresentato dagli indici sulla tabella Modulo, che devono essere continuamente aggiornati in fase di inserimento dei nuovi record.

La prima soluzione testata ha consistito nell'utilizzo di una tabella "*temporanea*", ModuloTMP, con indici solo sugli attributi indispensabili (ovvero i campi *unique* e *primary key*, la cui univocità è garantita da PostgreSQL proprio mediante l'utilizzo di indici). Tuttavia, questa soluzione si è rivelata inefficace.

Al fine di ovviare al problema di estrema latenza, insieme al mio tutor aziendale, Francesco Gregori, si è scelto di optare nuovamente per la **gestione del caricamento dei tracciati a fine giornata**, come nel vecchio sistema.

In questo modo **il produttore potrà immediatamente riprendere la navigazione** all'interno del sito web ed eventualmente scollegarsi, ed i tracciati saranno processati solo quando il sistema non sarà in uso da parte dei propri utenti (ovvero a fine giornata). Infine, le ricevute di caricamento delle dichiarazioni dei contributi ambientali saranno inviate, tramite email, nuovamente a fine giornata.

5.4 – Visualizzazione dei Tracciati Fotovoltaici

Durante i test del sistema, è emerso un aspetto critico nella pagina del sito web dedicata alla visualizzazione dei tracciati dei moduli fotovoltaici (mostrate nel *Capitolo 4.4.3*).

È stato riscontrato un problema di caricamento della pagina web in caso di un numero di record molto elevato. Nei prossimi sotto capitoli saranno analizzati nel dettaglio il problema e la sua causa.

5.4.1 – Problemi nel Caricamento della Pagina Web

La pagina web per la visualizzazione dei tracciati dei moduli fotovoltaici presenta **problemi di caricamento con un numero di record molto elevato**.

Con 1.000 record il caricamento è piuttosto immediato, mentre con un numero di record superiore (ordine di grandezza dei 10.000) il browser non riesce a caricare l'intera pagina in tempi celeri.

Il problema, in realtà, non risiede nell'impossibilità di visualizzare il contenuto per intero (perlomeno sul browser utilizzato per i test), bensì nell'**eccessivo tempo richiesto dal browser** per mostrare a video l'intera pagina. Infatti, il browser del client riceve immediatamente il risultato della richiesta dal server, ma non riesce a stamparlo a video in tempi brevi.

5.4.2 – Considerazioni sulla Risoluzione del Problema

La soluzione più semplice per porre rimedio al problema consiste nella **suddivisione dei record in *chunk***, ovvero blocchi di record (magari da 100 righe) che verranno visualizzati singolarmente sulla schermata; mediante il click su appositi pulsanti di navigazione, sarà possibile visualizzare gli altri record (sempre a blocchi da 100). In questo modo, le pagine potranno essere mostrate a video istantaneamente.

Tuttavia, come già descritto nei capitoli *4.4* e *3.3.2*, le pagine del sito web realizzate costituiscono un **semplice modello di interazione con la base di dati** e non saranno integrate direttamente nel sito del consorzio di filiera; in futuro, infatti, si farà riferimento alle modalità di gestione delle pagine già attualmente in uso nel sito del consorzio. Per questa ragione, unita alla mancanza di tempo sufficiente nel corso del tirocinio, **questa soluzione non è stata implementata**.

5.5 – Analisi Prestazioni degli Indici

L'ultima categoria di test effettuati sul sistema ha analizzato le prestazioni degli indici nell'esecuzione delle query, al fine di determinare la corretta reattività del sistema.

Di seguito, verranno riportate e descritte alcune delle query più importanti e comuni utilizzate nell'ambito della realtà di interesse del sistema software. Per tutte le query sono state effettuati sei benchmark (tre con indice, tre senza) al fine di calcolare una media più veritiera.

È possibile consultare l'elenco degli indici creati (oltre quelli di default per i campi *unique* e *primary key*) nell'*Appendice* del documento.

5.5.1 – Query 1

La prima query analizzata riguarda un tipico caso d'uso del sistema: un produttore vuole ricercare un ordine effettuato in una particolare data.

Questa semplice query (mostrata nella *Figura 5.1*, nella pagina seguente) è esprimibile mediante una ricerca effettuata direttamente sulla tabella ordine, supponendo di conoscere già l'id del produttore loggato nel sistema. In caso contrario, sarà sufficiente aggiungere, mediante il costrutto *JOIN*, le tabelle Produttore ed UtenteProduttore.

```
SELECT *
FROM Ordine
WHERE a_id_produttore_fk = 1 AND a_data_invio = '2020-10-05';
```

Figura 5.2 – Query 1

Si riportano i tempi di esecuzione della query (riportati dall'IDE DataGrip), che ha restituito un record:

- Senza indici: 55ms (esecuzione: 6ms, stampa a video: 49ms)
- Con indici: 50ms (esecuzione: 3ms, stampa a video: 47ms)

Segue il grafico dei tempi impiegati (*Figura 5.3*):

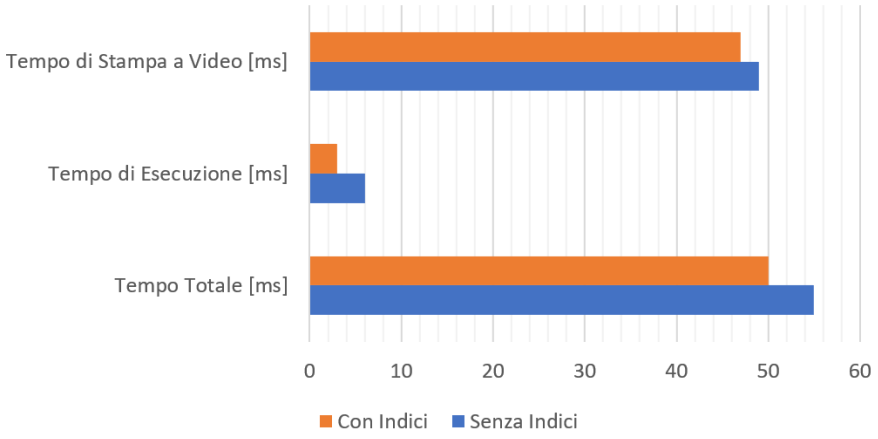


Figura 5.3 – Grafico Query 1

Sebbene il tempo totale sia simile, è importante notare che il **tempo di esecuzione** della query sia **dimezzato**. Infatti, nella prima esecuzione della query nessun campo è indicizzato, mentre nella seconda esecuzione entrambi i campi presenti nella clausola *where* sono indicizzati.

Tuttavia, è bene considerare che questo risultato rappresenta solo un piccolo campione e che la leggera differenza di tempo di esecuzione può essere stata causata da risorse di sistema occupate.

5.5.2 – Query 2

La seconda query analizzata (*Figura 5.4*) rappresenta un tipico caso d'uso: un produttore vuole elencare i tracciati delle dichiarazioni effettuate nell'ultimo periodo di riferimento (indicato mediante un intervallo di date).

```
SELECT TRA.a_nome_file, ORD.a_codice_ordine
FROM Tracciato TRA, Ordine ORD
WHERE TRA.a_id_ordine_fk = ORD.a_id_ordine AND
      ORD.a_id_produttore_fk = 1 AND
      ORD.a_data_invio BETWEEN '2020-09-15' AND '2020-10-05'
ORDER BY ORD.a_codice_ordine DESC, TRA.a_nome_file;
```

Figura 5.4 – Query 2

Si riportano i tempi di esecuzione della query (riportati dall'IDE DataGrip), che ha restituito un record:

- Senza indici: 30ms (esecuzione: 5ms, stampa a video: 25ms)
- Con indici: 27ms (esecuzione: 3ms, stampa a video: 24ms)

Segue il grafico dei tempi impiegati (*Figura 5.5*):

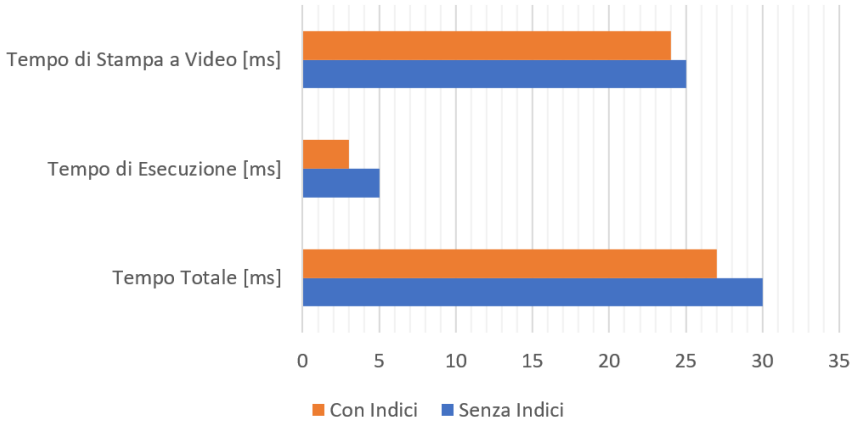


Figura 5.5 – Grafico Query 2

Sebbene il tempo totale sia simile, è importante notare che il **tempo di esecuzione** della query sia **quasi dimezzato**. Infatti, nella prima esecuzione della query nessun campo è indicizzato, mentre nella seconda esecuzione tutti i campi presenti nella clausola *where* sono indicizzati.

Tuttavia, è bene considerare che questo risultato rappresenta solo un piccolo campione e che la leggera differenza di tempo di esecuzione può essere stata causata da risorse di sistema occupate.

5.5.3 – Query 3

La seguente query (*Figura 5.6* nella pagina seguente) permette di elencare tutti i tracciati dei moduli dichiarati in un trimestre ed anno di riferimento. Questa interrogazione verrà utilizzata per la consultazione dei report trimestrali da parte dei produttori, mediante l'apposita sezione del sito web (consultare il *Capitolo 4.4.4* per maggiori dettagli).

Questa query risulta avere una **complessità computazionale superiore** rispetto a quelle precedentemente analizzate, in quanto, al fine di visualizzare tutti i dettagli dei moduli, richiede il *JOIN* con le tabelle:

- Categoria
- Sottocategoria
- Tipologia
- Ordine
- Listino Base
- Listino Personalizzato

```
SELECT MOD.a_codice_modulo AS seriale, LPE.z_vecchio_codice_listino AS listino,CAT.a_nome AS categoria,
       SCA.a_nome as sottocategoria, TIP.a_nome as tipologia, MOD.a_valore as valore, LPE.a_sconto as sconto,
       MOD.a_potenza as potenza, MOD.a_peso as peso, MOD.a_note as note
FROM Modulo AS MOD, Categoria AS CAT, SottoCategoria AS SCA, Tipologia AS TIP,
     ListinoPersonalizzato AS LPE, ListinoBase AS LBA, Ordine AS ORD
WHERE MOD.a_id_listino_personalizzato_fk = LPE.a_id_listino_personalizzato AND
      LPE.a_id_listino_base_fk = LBA.a_id_listino_base AND
      LBA.a_id_sottocategoria_fk = SCA.a_id_sottocategoria AND
      LBA.a_id_categoria_fk = SCA.a_id_categoria_padre_fk AND
      SCA.a_id_categoria_padre_fk = CAT.a_id_categoria AND
      LBA.a_id_tipologia_fk = TIP.a_id_tipologia AND
      MOD.a_id_ordine_fk = ORD.a_id_ordine AND
      ORD.a_id_produttore_fk = 68 AND
      ORD.a_trimestre = 3 AND
      ORD.a_anno = 2019
ORDER BY MOD.a_id_modulo;
```

Figura 5.6 – Query 3

Si riportano i tempi di esecuzione della query (riportati dall'IDE DataGrip), che ha restituito 239 record:

- Senza indici: 479ms (esecuzione: 449ms, stampa a video: 30ms)
- Con indici: 47ms (esecuzione: 8ms, stampa a video: 39ms)

Segue il grafico dei tempi impiegati (*Figura 5.7*):

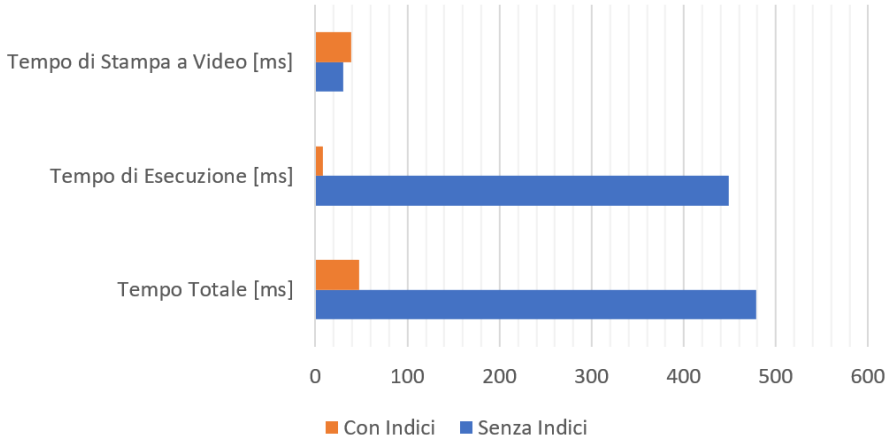


Figura 5.7 – Grafico Query 3

L'esecuzione della query, grazie all'aggiunta di indici su tutti i campi presenti nella clausola *WHERE* (prima nessuno di essi risultava indicizzato – ad esclusione delle chiavi primarie), è ben 56 volte più veloce (10 volte più veloce se si considera anche il tempo di stampa a video).

In questa query, che coinvolge ben sette tabelle mediante *JOIN*, è ben evidente l'**enorme miglioramento prestazionale** garantito dall'utilizzo degli indici.

5.5.4 – Query 4

La query seguente (*Figura 5.8*) rappresenta un tipico caso d'uso: un produttore vuole visualizzare la dichiarazione (ordine) con il valore (imponibile) maggiore da lui effettuata.

```
SELECT ORD.a_codice_ordine, ORD.a_imponibile, ORD.a_nr_moduli, ORD.a_trimestre, ORD.a_anno, ORD.a_data_invio
FROM Ordine AS ORD
WHERE ORD.a_id_produttore_fk = 68 AND ORD.a_imponibile = (
                                                          SELECT MAX(ORD2.a_imponibile)
                                                          FROM Ordine AS ORD2
                                                          WHERE ORD2.a_id_produttore_fk = 68
                                                        );
```

Figura 5.8 – Query 4

Si riportano i tempi di esecuzione della query (riportati dall'IDE DataGrip), che ha restituito un record:

- Senza indici: 45ms (esecuzione: 6ms, stampa a video: 39ms)
- Con indici: 32ms (esecuzione: 6ms, stampa a video: 26ms)

Segue il grafico dei tempi impiegati (*Figura 5.9*):

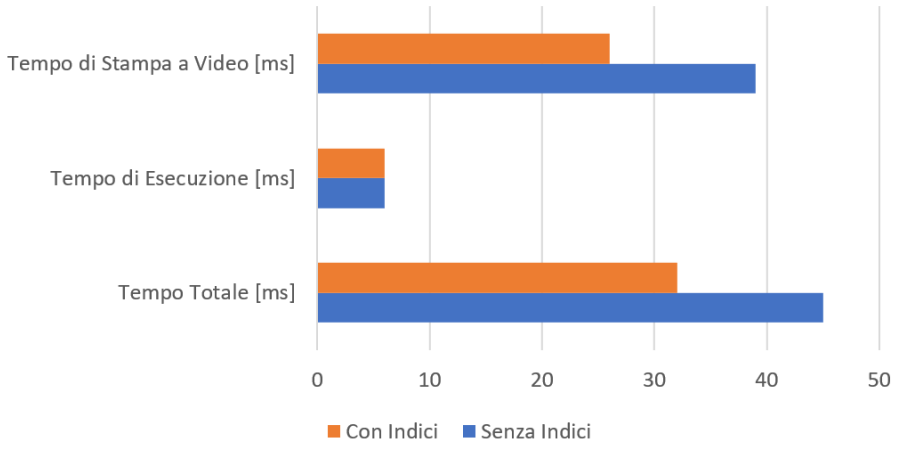


Figura 5.9 – Grafico Query 4

L'aggiunta di un indice sul campo *a_id_produttore_fk* (che rappresenta la chiave esterna verso l'entità produttore) non ha portato miglioramenti nell'effettivo tempo di esecuzione della query.

5.5.5 – Query 5

L'ultima query analizzata (*Figura 5.10*) rappresenta una specifica operazione di ricerca che i produttori potranno effettuare all'interno della pagina del sito web dedicata alla consultazione dei report trimestrali.

La query consente di ricercare il trimestre nel quale è stato dichiarato il numero maggiore di record; il risultato di questa interrogazione potrà essere utilizzato da altre query per esempio per visualizzare l'elenco di tutte le dichiarazioni contenute nel trimestre trovato.

```sql
SELECT ORD2.a_trimestre, ORD2.a_anno, SUM(ORD2.a_nr_moduli)
FROM Ordine AS ORD2
WHERE ORD2.a_id_produttore_fk = 68
GROUP BY ORD2.a_trimestre, ORD2.a_anno
ORDER BY 3 DESC
LIMIT 1;
```

Figura 5.10 – Query 5

Si riportano i tempi di esecuzione della query (riportati dall'IDE DataGrip), che ha restituito un record:

- Senza indici: 34ms (esecuzione: 4ms, stampa a video: 30ms)
- Con indici: 31ms (esecuzione: 3ms, stampa a video: 28ms)

Segue il grafico dei tempi impiegati (*Figura 5.11*):

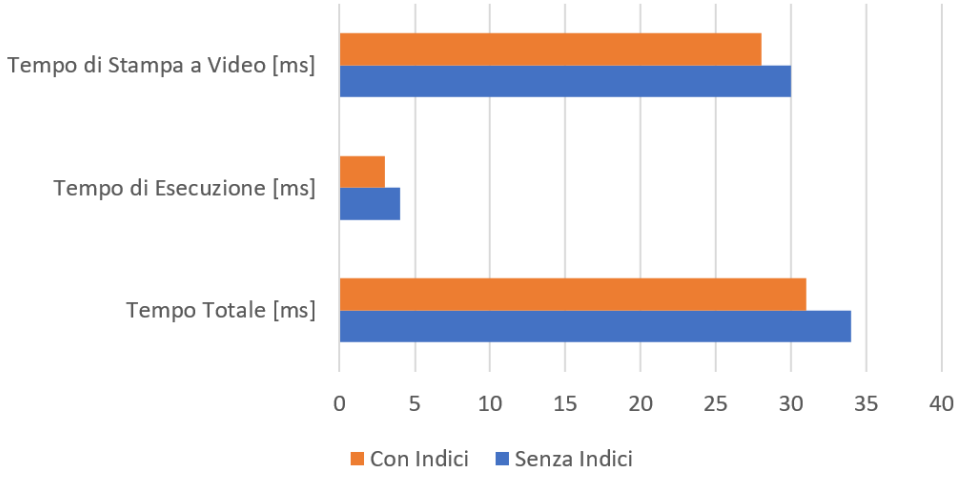


Figura 5.11 – Grafico Query 5

Sebbene nella prima esecuzione della query nessun campo era indicizzato, mentre nella seconda esecuzione tutti i campi presenti nella clausola *where* sono indicizzati, non sono stati ottenuti grandi miglioramenti prestazionali nel tempo di esecuzione della query.

È bene considerare che questo risultato rappresenta solo un piccolo campione e che la leggera differenza di tempo di esecuzione può essere stata causata da risorse di sistema occupate.

Conclusioni e Sviluppi Futuri

La realizzazione di questo progetto ha costituito una solida base per il rinnovamento ed il miglioramento del sistema software di un importante consorzio di filiera italiano.

Il sistema progettato, e realizzato, soddisfa a pieno i requisiti descritti da Multitraccia nelle fasi preliminari del progetto e va incontro alla necessità di una base di dati consistente e prestante per la gestione delle dichiarazioni dei contributi ambientali da parte di produttori di moduli fotovoltaici.

Per ragioni normative (descritte nel *Capitolo 1.5*) è fondamentale che i dati relativi alle dichiarazioni siano correttamente accessibili anche tra decine di anni, ovvero al termine del ciclo di vita dei moduli: il contributo ambientale pagato dai produttori, infatti, consentirà il corretto ritiro e smaltimento dei pannelli senza ulteriori costi (ad esempio da parte di un privato che ha un pannello montato sul tetto della propria abitazione).

Il nuovo database risolve importanti criticità dovute alla ridondanza di alcuni dati che hanno causato, nel tempo, l'incoerenza ed inconsistenza di alcuni record memorizzati. Inoltre, PostgreSQL (il DBMS scelto) offre prestazioni superiori a quelle di FileMaker e dunque crea ottime prospettive future, consentendo la gestione efficiente di una quantità di dati altamente superiore a quella attuale – pari ad oltre 4 milioni di record.

Il sistema software, tuttavia, non è stato completamente implementato per ragioni tempistiche, nonché a causa del sito del consorzio di filiera attualmente in aggiornamento che non ha permesso la realizzazione di pagine web direttamente compatibili con il futuro sito.

Gli sviluppi futuri del progetto riguardano innanzitutto la completa implementazione e messa in funzione del sistema. Inoltre, Multitraccia vorrebbe realizzare in futuro una nuova sezione del sito web che consentirà ai produttori di visualizzare una mappa interattiva dei moduli dichiarati ed attivi; la mappa riguarderà innanzitutto il territorio italiano, ma sarà possibile effettuare anche una gestione a livello internazionale.

Multitraccia mi ha già accennato della possibilità di proseguire la nostra collaborazione, al fine di completare la realizzazione del sistema in ogni sua parte nonché ampliarlo mediante l'aggiunta di nuove funzionalità. Tale proposta risulta essere un'ottima opportunità per ampliare le mie conoscenze e per maturare una maggiore esperienza nel campo dei big data, e sarà presa sicuramente in considerazione successivamente al conseguimento del titolo di studio offerto dal mio corso di laurea.

Bibliografia

[1] “Multitraccia”: http://www.multitraccia.it/

[2] “Direttiva 2002/95/CE del Parlamento Europeo e del Consiglio”: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2003:037:0019:0023:it:PDF

[3] “Direttiva 2002/96/CE del Parlamento Europeo e del Consiglio”: https://eur-lex.europa.eu/eli/dir/2002/96/oj

[4] “Direttiva 2003/108/CE del Parlamento Europeo e del Consiglio”: https://eur-lex.europa.eu/legal-content/IT/TXT/PDF/?uri=CELEX:32003L0108&from=EN

[5] “Direttiva 2012/19/UE del Parlamento Europeo e del Consiglio”: https://eur-lex.europa.eu/legal-content/IT/TXT/?uri=CELEX%3A32012L0019

[6] “Decreto Legislativo 25 luglio 2005, n. 151”: https://www.camera.it/parlam/leggi/deleghe/Testi/05151dl.htm

[7] “Decreto Legislativo 3 Aprile 2006, n. 152”: https://www.camera.it/parlam/leggi/deleghe/06152dl.htm

[8] “Decreto Legislativo 30 Dicembre 2008, n. 208”: https://www.gazzettaufficiale.it/eli/id/2009/02/28/09A02058/sg

[9] “Decreto Ministeriale 8 Marzo 2010, n. 65”: https://www.gazzettaufficiale.it/eli/id/2010/05/04/010G0087/sg

[10] “Modello Relazionale”: https://it.wikipedia.org/wiki/Modello_relazionale

[11] “Data Definition Language”: https://it.wikipedia.org/wiki/Data_Definition_Language

[12] “Data Manipulation Language”: https://it.wikipedia.org/wiki/Data_Manipulation_Language

[13] “Data Query Language”: https://it.wikipedia.org/wiki/Linguaggio_di_interrogazione

[14] “Structured Query Language”: https://it.wikipedia.org/wiki/Structured_Query_Language

[15] “NoSQL”: https://it.wikipedia.org/wiki/NoSQL

[16] “Cloud Computing”: https://it.wikipedia.org/wiki/Cloud_computing

[17] “Classifica dei DBMS – DB-Engines”: https://db-engines.com/en/ranking

[18] “Applicazione Web”: https://it.wikipedia.org/wiki/Applicazione_web

[19] “Sistema Client Server”: https://it.wikipedia.org/wiki/Sistema_client/server

[20] “Architettura Multi-Tier”: https://it.wikipedia.org/wiki/Architettura_multi-tier

[21] “Macchina Virtuale”: https://it.wikipedia.org/wiki/Macchina_virtuale

[22] “HyperVisor”: https://it.wikipedia.org/wiki/Hypervisor

[23] “Metodologia Agile”: https://it.wikipedia.org/wiki/Metodologia_agile

[24] “Extreme Programming”: https://it.wikipedia.org/wiki/Extreme_programming

[25] “Software Requirement Specification”: https://it.wikipedia.org/wiki/Specifica_dei_requisiti

[26] “UTF-8”: https://it.wikipedia.org/wiki/UTF-8

[27] “Indirizzo IP”: https://it.wikipedia.org/wiki/Indirizzo_IP

[28] “Log”: https://it.wikipedia.org/wiki/Log

[29] “Javascript Object Notation”: https://it.wikipedia.org/wiki/JavaScript_Object_Notation

[30] “Modello E/R”: https://it.wikipedia.org/wiki/Modello_E-R

[31] “Schema Logico”: https://it.wikipedia.org/wiki/Schema_di_database

[32] “Libreria DataTables”: https://datatables.net/examples/api/form.html

Appendice

Questa sezione del documento riporta il codice relativo all'implementazione del progetto di cui questa tesi è oggetto.

Vengono riportati il codice relativo alla creazione della base di dati e delle sue tabelle, le procedure di controllo ed i comandi per l'indicizzazione del database.

Non verrà mostrato il codice relativo all'applicazione web poiché ricalca fedelmente grafica e colori dell'attuale sito del consorzio, e contiene riferimenti ad esso. Inoltre, non è ritenuto utile l'inserimento del codice relativo agli script di importazione, in quanto contiene solo operazioni di lettura e conversione dei dati.

Premessa: il codice è stato scritto ed impaginato mediante l'utilizzo di appositi IDE (descritti nel *Capitolo 2.6*). Non si garantisce, tuttavia, la preservazione della corretta impaginazione del codice a seguito delle operazioni di copia-incolla nel documento *Word*.

A1 – Creazione del Database

```
/* ************************************************************************** */
/* ************************** Creazione DB e utente ************************** */
/* ************************************************************************** */

/* Creazione di un utente amministratore per il database */
DROP USER IF EXISTS multitracciaadmin;
CREATE USER multitracciaadmin SUPERUSER PASSWORD 'root';

/* Creazione del database ed assegnamento del proprietario */
DROP DATABASE IF EXISTS ftv_db;
CREATE DATABASE ftv_db OWNER multitracciaadmin;
```

A2 – Creazione delle Tabelle

```
/* ************************************************************************** */
/* ********************** DEBUG: drop dell'intero schema DB ********************** */
/* ************************************************************************** */

/* Drop dello schema del database (se esiste) e creazione (utile per ricreare il db da
zero in fase di DEBUG) */
DROP SCHEMA IF EXISTS public CASCADE;
CREATE SCHEMA public;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO public;
COMMENT ON SCHEMA public IS 'standard public schema';
SET search_path TO public;


/* ************************************************************************** */
/* ****************************** CREAZIONE DOMINI ****************************** */
/* ************************************************************************** */

/* La maggioranza dei seguenti domini è di tipo varchar e non ha una grande utilità, se
non quella di fare capire
   immediatamente che tipo di dato si ha davanti */

CREATE DOMAIN DomNome
    AS varchar(128);

CREATE DOMAIN DomDescrizione
    AS varchar(256);
```

```sql
CREATE DOMAIN DomPassword
    AS varchar(32);

/* Nel seguente dominio non viene controllata la lunghezza massima della singola mail */
CREATE DOMAIN DomEmail
    AS varchar(1024) /* possibilità di specificare più email separate da ';' */
    CHECK (VALUE ~*
           '^([A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+)((( )*;( )*[A-Za-z0-9._%-]+@[A-
Za-z0-9.-]+[.][A-Za-z]+)*)$');

CREATE DOMAIN DomRaggruppamento
    AS varchar(2)
    CHECK (VALUE IN ('R1', 'R2', 'R3', 'R4', 'R5'));

CREATE DOMAIN DomPerc
    AS numeric(5, 2)
    CHECK (VALUE BETWEEN 0 AND 100);

CREATE DOMAIN DomIP
    AS varchar(15)
    CHECK (VALUE ~* '^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$');

CREATE DOMAIN DomTipoLog
    AS varchar(6)
    CHECK (VALUE IN ('Script', 'Admin'));

CREATE DOMAIN DomTipoBatteria
    AS varchar(32);

CREATE DOMAIN DomOperazione
    AS varchar(32);

CREATE DOMAIN DomYear
    AS smallint
    CHECK (VALUE BETWEEN 0 AND extract(year from current_date));

CREATE DOMAIN DomMonth
    AS smallint
    CHECK (VALUE BETWEEN 1 AND 12);

CREATE DOMAIN DomCodiceListino
    AS varchar(8)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}$'); /* 1_2_13 */

CREATE DOMAIN DomComposto
    AS varchar(8)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}$'); /* 1_2_13 */

CREATE DOMAIN DomCompostoAnno
    AS varchar(15)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}_[0-9]{4}_[0-9]$'); /*
1_2_13_2020_2 (l'ultima cifra è il trimestre) */

CREATE DOMAIN DomCompostoPersonalizzato
    AS varchar(29)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}_.{5,20}$'); /* 1_2_13_12345678911
--> piva internazionale*/

CREATE DOMAIN DomCompostoAnnoPersonalizzato
    AS varchar(36)
    CHECK (VALUE ~* '^[0-9]{1,2}_[0-9]{1,2}_[0-9]{1,2}_[0-9]{4}_[0-9]_.{5,20}$'); /*
1_2_13_2020_2_12345678911 --> piva internazionale */

CREATE DOMAIN DomStato
    AS varchar(8)
    CHECK (VALUE IN ('Aperto', 'Chiuso', 'Pagato', 'Inviato', 'Inattivo', 'Attivo'));

CREATE DOMAIN DomTrimestre
```

```
    AS smallint
    CHECK (VALUE BETWEEN 1 AND 4);

CREATE DOMAIN DomUnitaMisura
    AS varchar(16)
    CHECK (VALUE IN ('Unità', 'Peso', 'Mq', 'Nr Celle', 'Qt', 'Qt Lamine'));

CREATE DOMAIN DomPrezzo
    AS numeric(8, 2)
    CHECK (VALUE >= 0);

CREATE DOMAIN DomPIva /* supporto a piva in vari formati con eventuale identificazione
internazionale */
    AS varchar(20);

CREATE DOMAIN DomVia
    AS VARCHAR(64);

CREATE DOMAIN DomCivico
    AS VARCHAR(5);
/*per civici lunghi come 1350a*/

/* Non c'è un controllo di formato perché possono essere molto differenti tra le nazioni
*/
CREATE DOMAIN DomCAP /* supporto a codici internazionali più lunghi */
    AS VARCHAR(10);

CREATE DOMAIN DomCodiceOrdine
    AS varchar(14)
    CHECK (VALUE ~* '^FTV[0-9]{11}$');

CREATE DOMAIN DomCodiceCobat
    AS varchar(32);

CREATE DOMAIN DomPeso
    AS numeric(8, 2) NOT NULL
    CHECK (VALUE >= 0);

CREATE DOMAIN DomComparto
    AS varchar(1);

CREATE DOMAIN DomCodiceIstatNazione
    AS varchar(3)
    CHECK (length(VALUE) = 3);

CREATE DOMAIN DomCodiceIstatComune
    AS varchar(6)
    CHECK (length(VALUE) = 6);

CREATE DOMAIN DomCodiceFiscale /* supporto a CF in vari formati con eventuale
identificazione internazionale */
    AS varchar(20);

CREATE DOMAIN DomTelefono /* supporto a numeri telefonici internazionali con eventuali
trattini di separazione */
    AS varchar(25);

/* Il seguente dominio serve per verificare la correttezza del nome del tracciato.
Potrebbe anche non essere usato */
CREATE DOMAIN DomNomeFile AS VARCHAR(17)
CHECK (VALUE ~* '^[A-Za-z0-9]{2}[0-9]{8}[A-Za-z0-9]{3}\.txt$');
/* 2219072020001.txt -  la regex corretta è '^[A-Za-z0-9]{2}[0-9]{11}\.txt$', ma nel db
originale ci sono casi in
   cui il progressivo giornaliero è nel formato XXX (con X carattere alfabetico...) */

CREATE DOMAIN DomNomeCartella
    AS varchar(2)
    CHECK (VALUE ~* '^[A-Za-z0-9]{2}$');
```

```sql
/* ******************************************************************************** */
/* ********************** CREAZIONE TABELLE: configurazioni ********************** */
/* ******************************************************************************** */
/* Tabella da usare per valori di configurazione globali (es. iva % attuale) */
CREATE TABLE DBConfigurazioni /* ok */
(
    a_id_cconfigurazione DomNome PRIMARY KEY,
    CHECK (length(a_id_cconfigurazione) > 0),
    a_valore             numeric(8, 2) NOT NULL
);
INSERT INTO DBConfigurazioni
VALUES ('IVA', 22);


/* ******************************************************************************** */
/* ********************** CREAZIONE TABELLE: parte centrale ********************** */
/* ******************************************************************************** */
CREATE TABLE Categoria /* ok */
(
    a_id_categoria smallserial PRIMARY KEY,
    CHECK (a_id_categoria > 0),
    a_nome         DomNome UNIQUE NOT NULL,

    a_descrizione  DomDescrizione
);

CREATE TABLE SottoCategoria /* ok */
(
    a_id_sottocategoria     smallint,
    CHECK (a_id_sottocategoria > 0),
    a_nome                  DomNome UNIQUE NOT NULL,

    a_descrizione           DomDescrizione,
    a_id_categoria_padre_fk smallint,

    PRIMARY KEY (a_id_sottocategoria, a_id_categoria_padre_fk),
    FOREIGN KEY (a_id_categoria_padre_fk) REFERENCES Categoria (a_id_categoria)
        ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE Nazione /* ok */
(
    a_id_nazione           smallserial PRIMARY KEY,
    CHECK (a_id_nazione > 0),
    a_nome                 DomNome UNIQUE                NOT NULL,
    a_codice_istat_nazione DomCodiceIstatNazione UNIQUE NOT NULL
);

CREATE TABLE Comune /* ok */
(
    a_id_comune           serial,
    CHECK (a_id_comune > 0),
    a_id_nazione_fk       smallint,
    CHECK (a_id_nazione_fk > 0),
    a_cap                 DomCAP UNIQUE                NOT NULL,
    a_nome                DomNome                      NOT NULL,
    a_codice_istat_comune DomCodiceIstatComune UNIQUE NOT NULL,

    PRIMARY KEY (a_id_nazione_fk, a_id_comune),
    FOREIGN KEY (a_id_nazione_fk) REFERENCES Nazione (a_id_nazione)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE Produttore /* ok */
(
    a_id_produttore serial PRIMARY KEY,
    CHECK (a_id_produttore > 0),
    a_cf            DomCodiceFiscale NOT NULL,
    a_p_iva         DomPIva UNIQUE   NOT NULL,
```

```
    a_email         DomEmail, /*a giudicare da Ordine1, non risulta unique */
    a_rag_sociale   DomNome, /* Può non essere presente in alcuni record */
    a_telefono      DomTelefono,
    a_note          DomDescrizione
);

CREATE TABLE Installatore /* ok */
(
    a_id_installatore serial PRIMARY KEY,
    CHECK (a_id_installatore > 0),
    a_cf              DomCodiceFiscale NOT NULL,
    a_p_iva           DomPIva UNIQUE   NOT NULL,
    a_email           DomEmail, /*a giudicare da Ordine1, non risulta unique */
    a_rag_sociale     DomNome,
    a_telefono        DomTelefono,
    a_note            DomDescrizione
);

CREATE TABLE SedeInstallazione /* ok */
(
    a_id_sede_installazione serial PRIMARY KEY,
    CHECK (a_id_sede_installazione > 0),
    a_nome                  DomNome UNIQUE NOT NULL,
    a_id_nazione_fk         smallint       NOT NULL,
    a_id_comune_fk          int            NOT NULL,
    a_via                   DomVia,
    a_civico                DomCivico,
    a_note                  DomDescrizione,

    FOREIGN KEY (a_id_nazione_fk, a_id_comune_fk) REFERENCES Comune (a_id_nazione_fk,
a_id_comune)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE Tipologia /* ok */
(
    a_id_tipologia smallserial PRIMARY KEY,
    CHECK (a_id_tipologia > 0),
    a_nome         DomNome UNIQUE NOT NULL,

    a_descrizione  DomDescrizione
);

CREATE TABLE ListinoBase /* ok */
(
    a_id_listino_base        smallserial PRIMARY KEY,
    CHECK (a_id_listino_base > 0),
    a_codice_listino         DomCodiceListino UNIQUE NOT NULL,
    a_composto               DomComposto UNIQUE      NOT NULL, /* da tenere per
retrocompatibilità */
    a_composto_anno          DomCompostoAnno UNIQUE  NOT NULL, /* da tenere per
retrocompatibilità */

    z_vecchio_codice_listino DomCodiceListino UNIQUE,
    z_vecchio_composto       DomNome UNIQUE, /* tipo dato particolare - a volte non
soddisfa il formato
                                                 del composto e quindi ho scelto un campo
senza vincoli di formato */
    z_vecchio_composto_anno  DomCompostoAnno UNIQUE,

    a_anno                   DomYear                 NOT NULL,
    a_trimestre              DomTrimestre            NOT NULL,
    a_attivo                 bool                    NOT NULL default true,
    a_id_categoria_fk        smallint                NOT NULL,
    a_id_sottocategoria_fk   smallint,
    a_id_tipologia_fk        smallint,
    a_stato                  boolean                 NOT NULL,
    a_nome                   DomNome,
    a_descrizione            DomDescrizione,
```

```
    a_unita_misura           DomUnitaMisura          NOT NULL,
    a_valore                 DomPrezzo               NOT NULL,
    a_sconto                 DomPerc                 NOT NULL default 0,
    a_note                   DomDescrizione,

    a_raggruppamento         DomRaggruppamento,
    a_tipo_batteria          DomTipoBatteria,

    z_timestamp_creazione    timestamp               NOT NULL DEFAULT now(),
    z_timestamp_modifica     timestamp               NOT NULL DEFAULT now(),

    FOREIGN KEY (a_id_categoria_fk) REFERENCES Categoria (a_id_categoria)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_sottocategoria_fk, a_id_categoria_fk) REFERENCES SottoCategoria
(a_id_sottocategoria,

a_id_categoria_padre_fk)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_tipologia_fk) REFERENCES Tipologia (a_id_tipologia)
        ON DELETE NO ACTION ON UPDATE CASCADE,

    CHECK (z_timestamp_modifica >= z_timestamp_creazione)
);

CREATE TABLE ListinoPersonalizzato /* solo per le info aggiuntive rispetto al listino
base */ /* ok */
(
    a_id_listino_personalizzato serial PRIMARY KEY,
    a_id_listino_base_fk        smallint                            NOT NULL,
    a_id_produttore_fk          int                                 NOT NULL,
    a_composto                  DomCompostoPersonalizzato UNIQUE    NOT NULL, /* da
tenere per retrocompatibilità */
    a_composto_anno             DomCompostoAnnoPersonalizzato UNIQUE NOT NULL, /* da
tenere per retrocompatibilità */

    z_vecchio_codice_listino    DomCodiceListino, /* da tenere per retrocompatibilità */
    z_vecchio_composto          DomCompostoPersonalizzato UNIQUE,
    z_vecchio_composto_anno     DomCompostoAnnoPersonalizzato UNIQUE,

    a_unita_misura              DomUnitaMisura                      NOT NULL,
    a_valore                    DomPrezzo                           NOT NULL,
    a_sconto                    DomPerc                             NOT NULL default 0,
    a_note                      DomDescrizione,
    a_descrizione               DomDescrizione,

    z_timestamp_creazione       timestamp                           NOT NULL DEFAULT
now(),
    z_timestamp_modifica        timestamp                           NOT NULL DEFAULT
now(),

    FOREIGN KEY (a_id_listino_base_fk) REFERENCES ListinoBase (a_id_listino_base)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_produttore_fk) REFERENCES Produttore (a_id_produttore)
        ON DELETE NO ACTION ON UPDATE CASCADE,

    UNIQUE (a_id_listino_base_fk, a_id_produttore_fk),
    UNIQUE (z_vecchio_codice_listino, a_id_produttore_fk),

    CHECK (z_timestamp_modifica >= z_timestamp_creazione)
);

CREATE TABLE Ordine /* ok */
(
    a_id_ordine                 serial PRIMARY KEY,
    CHECK (a_id_ordine > 0),
    a_codice_ordine             DomCodiceOrdine UNIQUE NOT NULL,

    a_titolo                    DomNome,
    a_id_produttore_fk          int                     NOT NULL,
```

```
    a_email                     DomEmail,

    a_imponibile                DomPrezzo                       DEFAULT 0,
    a_nr_moduli                 int                     NOT NULL DEFAULT 0, /* Conta il
numero di moduli */
    CHECK (a_nr_moduli >= 0),
    a_iva                       DomPerc                 NOT NULL, /* Aliquota iva nel
momento in cui è stato
                                                                    effettuato l'ordine
*/
    a_pagato                    boolean                 NOT NULL DEFAULT false,
    a_fatturato                 boolean                 NOT NULL DEFAULT false,

    a_stato                     DomStato                NOT NULL DEFAULT 'Aperto',
    a_attivo                    boolean                 NOT NULL DEFAULT true,
    a_inviate                   smallint                NOT NULL DEFAULT 0,
    a_data_invio                date,
    a_data_conferma             date,
    a_anno                      DomYear                 NOT NULL,
    a_trimestre                 DomTrimestre,
    a_note                      DomDescrizione,

    a_sollecito_pagamento_to    DomEmail,
    a_sollecito_pagamento_cc    DomEmail,
    a_sollecito_pagamento_ccn   DomEmail,
    a_sollecito_pagamento_invio date,

    z_timestamp_creazione       timestamp               NOT NULL DEFAULT now(),
    z_timestamp_modifica        timestamp               NOT NULL DEFAULT now(),

    FOREIGN KEY (a_id_produttore_fk) REFERENCES Produttore (a_id_produttore)
        ON DELETE NO ACTION ON UPDATE CASCADE,

    CHECK (a_data_conferma >= a_data_invio),
    CHECK (z_timestamp_modifica >= z_timestamp_creazione)
);

CREATE TABLE Tracciato
(
    a_id_tracciato  serial PRIMARY KEY,
    CHECK (a_id_tracciato > 0),
    a_id_ordine_fk  int         NOT NULL,
    a_nome_file     DomNomeFile NOT NULL, /* ATTENZIONE: dovrebbe essere unique, ma nel
db originale non sempre lo è*/
    a_nome_cartella DomNomeCartella, /* campo tenuto per compatibilità con il vecchio db
*/
    z_file          bytea,

    FOREIGN KEY (a_id_ordine_fk) REFERENCES Ordine (a_id_ordine)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE Modulo /* ok */
(
    a_id_modulo                     bigserial PRIMARY KEY,
    CHECK (a_id_modulo > 0),
    a_id_ordine_fk                  int             NOT NULL,
    a_id_listino_personalizzato_fk  int             NOT NULL,
    a_id_tracciato_fk               int, /* ATTENZIONE:  dovrebbe essere not null, ma in
alcuni record è assente il
                                        riferimento al tracciato */
    a_codice_modulo                 DomNome         NOT NULL, /* può essere usato da più
produttori e per questo non è
                                                                  unique */
    a_codice_sistema                DomNome UNIQUE NOT NULL, /* Nel vecchio DB era un
campo generato univocamente, ora
                                                                  non è molto utile */
    a_codice_modulo_piva            DomNome UNIQUE NOT NULL,
```

```
    a_codice_comparto              DomComparto    NOT NULL,

    a_valore                       DomPrezzo      NOT NULL,
    /*a_quantita                     smallint                default 1,
    CHECK (a_quantita > 0),*/ /* La quantità è sempre pari ad 1 */
    a_unita_misura                 DomUnitaMisura NOT NULL default 'Peso',
    a_peso                         DomPeso,
    a_numero_celle                 smallint                default 1,
    a_potenza                      numeric(10, 2),

    a_pagato                       boolean        NOT NULL default false,
    a_stato                        DomStato       NOT NULL default 'Aperto', /*
teoricamente corrisponde a quello

dell'ordine, ma possono esservi casi
                                                                                in cui
non accade */
    a_attivo                       boolean        NOT NULL default false, /* installato
si no - per memorizzare i casi
                                                                              in cui il
modulo sia installato ma non
                                                                              teniamo
conto di chi lo ha fatto */
    a_note                         DomDescrizione,

    a_modulo_attivo                boolean        NOT NULL default false,
    a_id_sede_installazione_fk     int,
    a_id_installatore_fk           int,
    a_modulo_modello               DomNome,
    z_data_installazione           date,
    z_data_attivazione             date,

    z_timestamp_creazione          timestamp      NOT NULL DEFAULT now(),
    z_timestamp_modifica           timestamp      NOT NULL DEFAULT now(),

    FOREIGN KEY (a_id_listino_personalizzato_fk) REFERENCES ListinoPersonalizzato
(a_id_listino_personalizzato)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_ordine_fk) REFERENCES Ordine (a_id_ordine)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_sede_installazione_fk) REFERENCES SedeInstallazione
(a_id_sede_installazione)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_installatore_fk) REFERENCES Installatore (a_id_installatore)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (a_id_tracciato_fk) REFERENCES Tracciato (a_id_tracciato)
        ON DELETE NO ACTION ON UPDATE CASCADE,

    CHECK (z_timestamp_modifica >= z_timestamp_creazione)/*,
    CHECK (z_data_attivazione >= z_data_installazione)*/ /*Non sempre viene rispettato
questo ordine*/

);


/* ******************************************************************************** */
/* ************************** CREAZIONE TABELLE: utenti ************************** */
/* ******************************************************************************** */

/* Le seguenti tabelle sono state predisposte per una gestione degli utenti, che per
ragioni di tempo non è stata
   effettuata.

   I timestamp e l'ip dell'ultimo login hanno senso di essere tenuti anche qui (oltre che
nei log) per motivi di
   performance

   Le tabelle utente produttore ed installatore non aggiungono molte informazioni
rispetto alle tabelle produttore
```

```
   ed installazione, tuttavia è meglio mantenere una suddivisione logica tra i dati
dell'utente del sito e tutte le
   altre informazioni anagrafiche */

CREATE TABLE UtenteAdmin
(
    a_id_utente_admin        smallserial PRIMARY KEY,
    CHECK (a_id_utente_admin > 0),
    a_username               DomNome UNIQUE  NOT NULL,
    a_email                  DomEmail UNIQUE NOT NULL,

    a_password               DomPassword     NOT NULL, /* da hashare */

    z_ultimo_login_timeStamp timestamp,
    z_ultimo_login_ip        DomIP
);

CREATE TABLE UtenteLettura
(
    a_id_utente_lettura      smallserial PRIMARY KEY,
    CHECK (a_id_utente_lettura > 0),
    a_username               DomNome UNIQUE  NOT NULL,
    a_email                  DomEmail UNIQUE NOT NULL,

    a_password               DomPassword, /* da hashare */

    z_ultimo_login_timeStamp timestamp,
    z_ultimo_login_ip        DomIP
);

CREATE TABLE UtenteProduttore
(
    a_id_utente_produttore   smallserial PRIMARY KEY,
    CHECK (a_id_utente_produttore > 0),
    a_username               DomNome UNIQUE,
    a_email                  DomEmail UNIQUE NOT NULL,
    a_id_produttore_fk       int UNIQUE      NOT NULL,

    a_password               DomPassword     NOT NULL, /* da hashare */

    z_ultimo_login_timeStamp timestamp,
    z_ultimo_login_ip        DomIP,

    FOREIGN KEY (a_id_produttore_fk) REFERENCES Produttore (a_id_produttore)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE UtenteInstallatore
(
    a_id_utente_installatore smallserial PRIMARY KEY,
    CHECK (a_id_utente_installatore > 0),
    a_username               DomNome UNIQUE,
    a_email                  DomEmail UNIQUE NOT NULL,
    a_id_installatore_fk     int UNIQUE      NOT NULL,

    a_password               DomPassword     NOT NULL, /* da hashare */

    z_ultimo_login_timeStamp timestamp,
    z_ultimo_login_ip        DomIP,

    FOREIGN KEY (a_id_installatore_fk) REFERENCES Installatore
        ON DELETE NO ACTION ON UPDATE CASCADE
);


/* ******************************************************************************** */
/* **************************** CREAZIONE TABELLE: Log ************************** */
/* ******************************************************************************** */
CREATE TABLE LogModifiche /* memorizza le modifiche ai record delle tabelle */ /* ok */
```

```
(
    a_id_log_modifiche bigserial PRIMARY KEY,
    CHECK (a_id_log_modifiche > 0),

    a_operazione       DomOperazione NOT NULL,
    a_nome_tabella     varchar(32)   NOT NULL,
    a_new_val          json,
    a_old_val          json,

    z_timestamp        timestamp     NOT NULL DEFAULT now(),
    z_utente           varchar(32)            DEFAULT current_user /* questo è quello di
postgre*/
);

CREATE TABLE LogInterno /* script ed amministratore*/ /* ok */
(
    a_id_log_interno     bigserial PRIMARY KEY,
    CHECK (a_id_log_interno > 0),
    a_id_utente_admin_fk smallint,

    a_operazione         DomOperazione NOT NULL,
    a_tipo_log           DomTipoLog    NOT NULL default 'Script',

    z_timestamp_1        timestamp     NOT NULL default now(), /* inizio */
    z_timestamp_2        timestamp     NOT NULL default now(), /* fine */
    z_file_utilizzati    varchar(256),
    z_modifica           varchar(256), /* Descrizione dell'operazione */
    z_progetto           DomNome,
    z_script_utilizzati  varchar(256),

    FOREIGN KEY (a_id_utente_admin_fk) REFERENCES UtenteAdmin (a_id_utente_admin)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE LogProduttore /* ok */
(
    a_id_log_produttore       bigserial PRIMARY KEY,
    CHECK (a_id_log_produttore > 0),
    a_id_utente_produttore_fk smallint      NOT NULL,

    a_operazione              DomOperazione NOT NULL,

    z_timestamp_1             timestamp     NOT NULL default now(), /* inizio */
    z_timestamp_2             timestamp     NOT NULL default now(), /* fine */
    z_file_utilizzati         varchar(256),
    z_modifica                varchar(256), /* Descrizione dell'operazione */
    z_progetto                DomNome,
    z_script_utilizzati       varchar(256),

    FOREIGN KEY (a_id_utente_produttore_fk) REFERENCES UtenteProduttore
(a_id_utente_produttore)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE LogInstallatore /* ok */
(
    a_id_log_installatore       bigserial PRIMARY KEY,
    CHECK (a_id_log_installatore > 0),
    a_id_utente_installatore_fk smallint      NOT NULL,

    a_operazione                DomOperazione NOT NULL,

    z_timestamp_1               timestamp     NOT NULL default now(), /* inizio */
    z_timestamp_2               timestamp     NOT NULL default now(), /* fine */
    z_file_utilizzati           varchar(256),
    z_modifica                  varchar(256), /* Descrizione dell'operazione */
    z_progetto                  DomNome,
    z_script_utilizzati         varchar(256),
```

```
    FOREIGN KEY (a_id_utente_installatore_fk) REFERENCES UtenteInstallatore
(a_id_utente_installatore)
        ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE LogLettura /* ok */
(
    a_id_log_lettura       bigserial PRIMARY KEY,
    CHECK (a_id_log_lettura > 0),
    a_id_utente_lettura_fk smallint      NOT NULL,

    a_operazione           DomOperazione NOT NULL,

    z_timestamp_1          timestamp     NOT NULL default now(), /* inizio */
    z_timestamp_2          timestamp     NOT NULL default now(), /* fine */
    z_file_utilizzati      varchar(256),
    z_modifica             varchar(256), /* Descrizione dell'operazione */
    z_progetto             DomNome,
    z_script_utilizzati    varchar(256),

    FOREIGN KEY (a_id_utente_lettura_fk) REFERENCES UtenteLettura (a_id_utente_lettura)
        ON DELETE NO ACTION ON UPDATE CASCADE
);
```

A3 – Creazione delle Procedure

```
SET search_path TO public;

/* ********************************************** SOTTOCATEGORIA
********************************************** */
/* T1) All'inserimento di una sottocategoria, viene generato il suo id seriale */
CREATE OR REPLACE FUNCTION insert_sottocategoria()
RETURNS TRIGGER AS $insert_sottocategoria$
    DECLARE
        local_max_id_sottocategoria smallint := 0;
   BEGIN
      local_max_id_sottocategoria = (
                                        SELECT MAX(S.a_id_sottocategoria)
                                        FROM SottoCategoria S
                                        WHERE S.a_id_categoria_padre_fk =
NEW.a_id_categoria_padre_fk
                                      );

      IF (local_max_id_sottocategoria IS NOT NULL)
      THEN
          NEW.a_id_sottocategoria = local_max_id_sottocategoria + 1;
      ELSE
          NEW.a_id_sottocategoria = 1;
      END IF;

      RETURN NEW;
   END;
$insert_sottocategoria$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_sottocategoria /* in caso esista già */
ON SottoCategoria;

CREATE TRIGGER insert_sottocategoria
BEFORE INSERT OR UPDATE ON SottoCategoria
FOR EACH ROW EXECUTE PROCEDURE insert_sottocategoria();

/* ********************************************** LISTINO BASE
********************************************** */
/* T2) All'aggiunta di un record nel listino base, i campi codice listino, composto e
composto anno vengono generati,
   rispettivamente come:
   - id_categoria_sottocategoria
```

```
   - id_categoria_sottocategoria
   - id_categoria_sottocategoria_anno_trimestre */
CREATE OR REPLACE FUNCTION insert_listino_base()
RETURNS TRIGGER AS $insert_listino_base$
    DECLARE
        local_id_tipologia smallint = 0;
   BEGIN

       IF NEW.a_id_tipologia_fk IS NOT NULL
       THEN
            local_id_tipologia = NEW.a_id_tipologia_fk;
        END IF;
        NEW.a_codice_listino = concat(NEW.a_id_categoria_fk, '_',
NEW.a_id_sottocategoria_fk, '_', local_id_tipologia);
        NEW.a_composto = NEW.a_codice_listino;
        NEW.a_composto_anno = concat(NEW.a_codice_listino, '_', NEW.a_anno, '_',
NEW.a_trimestre);

      RETURN NEW;
   END;
$insert_listino_base$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_listino_base /* in caso esista già */
ON listinobase;

CREATE TRIGGER insert_listino_base
BEFORE INSERT OR UPDATE ON listinobase
FOR EACH ROW EXECUTE PROCEDURE insert_listino_base();

/* ********************************************** LISTINO PERSONALIZZATO
********************************************** */
/* T3) Per ogni ListinoPersonalizzato, si verifica che il timestamp di creazione sia >= a
quello del rispettivo
   ListinoBase

   Inoltre, All'aggiunta di un record nel listino personalizzato, i campi composto e
composto anno vengono generati,
   rispettivamente come:
   - id_categoria_sottocategoria_p_iva
   - id_categoria_sottocategoria_p_iva_anno_trimestre */
CREATE OR REPLACE FUNCTION insert_listino_personalizzato()
RETURNS TRIGGER AS $insert_listino_personalizzato$
    DECLARE
        local_codice_listino varchar(8) := '';
        local_anno smallint := 0;
        local_trimestre smallint := 0;
        local_p_iva_produttore varchar(20) := '';

   BEGIN
        IF(
            (NEW.z_timestamp_creazione <
                (
                    SELECT z_timestamp_creazione
                    FROM ListinoBase
                    WHERE ListinoBase.a_id_listino_base = NEW.a_id_listino_base_fk
                )
            )
        )
        THEN
            RAISE EXCEPTION 'Timestamp di creazione del Listino Personalizzato precedente
a quello del Listino Base! ID: "%"', NEW.a_id_listino_personalizzato;
        END IF;

        /* generazione codice composto */
        local_codice_listino = (
                                    SELECT a_codice_listino
                                    FROM ListinoBase
                                    WHERE ListinoBase.a_id_listino_base =
NEW.a_id_listino_base_fk
```

```
                                );
        local_anno = (
                                        SELECT a_anno
                                        FROM ListinoBase
                                        WHERE ListinoBase.a_id_listino_base =
NEW.a_id_listino_base_fk
                                );
        local_trimestre = (
                                        SELECT a_trimestre
                                        FROM ListinoBase
                                        WHERE ListinoBase.a_id_listino_base =
NEW.a_id_listino_base_fk
                                );
        local_p_iva_produttore = (
                                        SELECT a_p_iva
                                        FROM Produttore
                                        WHERE Produttore.a_id_produttore =
NEW.a_id_produttore_fk
                                );

        NEW.a_composto = concat(local_codice_listino, '_', local_p_iva_produttore);
        NEW.a_composto_anno = concat(local_codice_listino, '_', local_anno, '_',
local_trimestre, '_',
            local_p_iva_produttore);

      RETURN NEW;
   END;
$insert_listino_personalizzato$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_listino_personalizzato /* in caso esista già */
ON ListinoPersonalizzato;

CREATE TRIGGER insert_listino_personalizzato
BEFORE INSERT OR UPDATE ON ListinoPersonalizzato
FOR EACH ROW EXECUTE PROCEDURE insert_listino_personalizzato();

/* ****************************************************** ORDINE
****************************************************** */
/* T4) All'inserimento di un Ordine, l'iva attuale viene presa dalla configurazione */
CREATE OR REPLACE FUNCTION insert_ordine()
RETURNS TRIGGER AS $insert_ordine$
   BEGIN
       IF NEW.a_iva IS NULL
        THEN
            NEW.a_iva = (
                                SELECT a_valore
                                FROM dbconfigurazioni
                                WHERE a_id_cconfigurazione = 'IVA'
                        );
        END IF;

      RETURN NULL;
   END;
$insert_ordine$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_ordine /* in caso esista già */
ON Ordine;

CREATE TRIGGER insert_ordine
AFTER INSERT ON Ordine
FOR EACH ROW EXECUTE PROCEDURE insert_ordine();


/* T5) Aggiornamento next val del campo seriale in caso di inserimento manuale e
generazione codice */
CREATE OR REPLACE FUNCTION before_insert_ordine()
RETURNS TRIGGER AS $before_insert_ordine$
    DECLARE
        local_codice_ordine varchar(14) = '';
```

```
        local_anno smallint = 2000;
        local_id smallint = 0;
        local_iva numeric = 0;
   BEGIN
       IF NEW.a_id_ordine IS NOT NULL
        THEN
            PERFORM setval('ordine_a_id_ordine_seq', (SELECT MAX(a_id_ordine) + 1 from
Ordine));
        END IF;

       /* Generazione del codice */
       local_anno = NEW.a_anno;
       local_id = NEW.a_id_ordine;
        local_codice_ordine = concat(local_id, local_anno);

       /* Aggiunta zeri per normalizzare la lunghezza */
        WHILE (length(local_codice_ordine) < (14 - 3)) LOOP
            local_codice_ordine = concat('0', local_codice_ordine);
        END LOOP;
        local_codice_ordine = concat('FTV', local_codice_ordine);
       NEW.a_codice_ordine = local_codice_ordine;

       local_iva = (SELECT a_valore FROM DBConfigurazioni WHERE a_id_cconfigurazione =
'IVA');
       NEW.a_iva = local_iva;

      RETURN NEW;
   END;
$before_insert_ordine$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS before_insert_ordine /* in caso esista già */
ON Ordine;

CREATE TRIGGER before_insert_ordine
BEFORE INSERT ON Ordine
FOR EACH ROW EXECUTE PROCEDURE before_insert_ordine();


/* T6) Generazione codice ordine in caso di update*/
CREATE OR REPLACE FUNCTION after_update_ordine()
RETURNS TRIGGER AS $after_update_ordine$
    DECLARE
        local_codice_ordine varchar(14) = '';
        local_anno smallint = 2000;
        local_id smallint = 0;
   BEGIN
       IF NEW.a_id_ordine != OLD.a_id_ordine OR NEW.a_anno != OLD.a_anno
        THEN
            local_anno = NEW.a_anno;
            local_id = NEW.a_id_ordine;
            local_codice_ordine = concat(local_id, local_anno);

            WHILE (length(local_codice_ordine) < (14 - 3)) LOOP
                local_codice_ordine = concat('0', local_codice_ordine);
            END LOOP;
            local_codice_ordine = concat('FTV', local_codice_ordine);
            NEW.a_codice_ordine = local_codice_ordine;
        END IF;

       /* aggiornamento del nextval del seriale in caso di modifiche, sembra non essere
necessario */
       /*IF NEW.a_id_ordine IS NOT NULL
        THEN
            PERFORM setval('ordine_a_id_ordine_seq', (SELECT MAX(a_id_ordine) from
Ordine));
        END IF;*/

      RETURN NEW;
   END;
```

```
$after_update_ordine$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS after_update_ordine /* in caso esista già */
ON Ordine;

CREATE TRIGGER after_update_ordine
AFTER UPDATE ON Ordine
FOR EACH ROW EXECUTE PROCEDURE after_update_ordine();


/* ***************************************************** MODULO
***************************************************** */
/* T7) Prima dell'inserimento di un modulo è necessario controllare che l'ordine ed il
listino personalizzato in FK
   facciano riferimento, a loro volta, allo stesso produttore: il produttore che fa
l'ordine e quello a cui appartiene
   il listino devono combaciare.

   Si verifica che i timestamp di creazione sia >= a quello del rispettivo Listino
Personalizzato ed Ordine.

   Si aggiorna il seriale in caso di inserimento manuale

   Generazione dei codice modulo piva */
CREATE OR REPLACE FUNCTION insert_modulo()
RETURNS TRIGGER AS $insert_modulo$
    DECLARE
        local_id_produttore_listino int := 0;
        local_id_produttore_ordine int := 0;
        local_codice_modulo DomNome = '';
        local_piva DomPIva = NULL;
        local_codice_modulo_piva DomNome = NULL;
        local_valore DomPrezzo = 0;
   BEGIN
        /* calcolo valore modulo dal listino */
        IF (NEW.a_valore = 0 OR NEW.a_valore IS NULL)
        THEN
            local_valore = (
                                SELECT (ListinoPersonalizzato.a_valore -
(ListinoPersonalizzato.a_valore *

ListinoPersonalizzato.a_sconto / 100))
                                FROM ListinoPersonalizzato
                                WHERE ListinoPersonalizzato.a_id_listino_personalizzato =
                                      NEW.a_id_listino_personalizzato_fk
                            );
            NEW.a_valore = local_valore;
        END IF;

        /* controllo corretto riferimento ai produttori */
        local_id_produttore_listino = (
                                        SELECT a_id_produttore_fk
                                        FROM ListinoPersonalizzato
                                        WHERE
ListinoPersonalizzato.a_id_listino_personalizzato =
                                              NEW.a_id_listino_personalizzato_fk
                                      );

        local_id_produttore_ordine = (
                                        SELECT a_id_produttore_fk
                                        FROM Ordine
                                        WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk
                                      );

        IF (local_id_produttore_ordine <> local_id_produttore_listino)
      THEN
   RAISE EXCEPTION 'Ordine e ListinoPersonalizzato fanno riferimento a due produttori
differenti! ID: "%"', NEW.a_id_modulo;
END IF;
```

```
      /* controllo ts modulo - listino */
      IF(
          (NEW.z_timestamp_creazione <
              (
                  SELECT z_timestamp_creazione
                  FROM ListinoPersonalizzato
                  WHERE ListinoPersonalizzato.a_id_listino_personalizzato =
NEW.a_id_listino_personalizzato_fk
                      AND ListinoPersonalizzato.a_id_produttore_fk = (
                                                                      SELECT
O.a_id_produttore_fk
                                                                      FROM Ordine O
                                                                      WHERE O.a_id_ordine
= NEW.a_id_ordine_fk
                                                                  )
              )
          )
      )
      THEN
          RAISE EXCEPTION 'Timestamp di creazione del modulo precedente a quello del
listino personalizzato! ID: "%"', NEW.a_id_listino_personalizzato;
      END IF;

   /* controllo ts modulo - ordine */
      IF(
          (NEW.z_timestamp_creazione <
              (
                  SELECT z_timestamp_creazione
                  FROM Ordine
                  WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk
              )
          )
      )
      THEN
          RAISE EXCEPTION 'Timestamp di creazione del modulo precedente a quello dell
Ordine! ID: "%"', NEW.a_id_modulo;
      END IF;


       /* aggiornamento seriale */
       IF NEW.a_id_modulo IS NOT NULL
        THEN
            PERFORM setval('modulo_a_id_modulo_seq', (SELECT MAX(a_id_modulo) + 1 from
Modulo));
        END IF;

        /* composizione codici */
        local_codice_modulo = NEW.a_codice_modulo;
       local_piva = (
                        SELECT a_p_iva
                        FROM Produttore P
                        WHERE P.a_id_produttore = (
                                                    SELECT O.a_id_produttore_fk
                                                    FROM Ordine O
                                                    WHERE O.a_id_ordine =
NEW.a_id_ordine_fk
                                                )
                    );

        local_codice_modulo_piva = concat(local_codice_modulo, '_', local_piva);
       NEW.a_codice_modulo_piva = local_codice_modulo_piva;

        IF NEW.a_codice_sistema IS NULL
        THEN
            NEW.a_codice_sistema = NEW.a_id_modulo;
        END IF;

      RETURN NEW;
```

```
   END;
$insert_modulo$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_modulo /* in caso esista già */
ON Modulo;

CREATE TRIGGER insert_modulo
BEFORE INSERT OR UPDATE ON Modulo
FOR EACH ROW EXECUTE PROCEDURE insert_modulo();


/* T8) Cacolo del totale ordine all'inserimento dei moduli che lo compongono.
   Nel modulo vengono inseriti valore e sconto presi dal rispettivo listino
personalizzato */
/* il totale viene aggiornato e non ricalcolato ogni volta.
   Inoltre viene incrementato il contatore del numero di moduli dell'ordine */
CREATE OR REPLACE FUNCTION ordine_insert_modulo()
RETURNS TRIGGER AS $ordine_insert_modulo$
   BEGIN
       IF (NEW.a_stato = 'Aperto')
       THEN
            UPDATE Ordine
            SET a_imponibile = a_imponibile + NEW.a_valore
            WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk;
        END IF;

       UPDATE Ordine
        SET a_nr_moduli = a_nr_moduli + 1
        WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk;

      RETURN NEW;
   END;
$ordine_insert_modulo$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS ordine_insert_modulo /* in caso esista già */
ON Modulo;

CREATE TRIGGER ordine_insert_modulo
AFTER INSERT ON Modulo
FOR EACH ROW EXECUTE PROCEDURE ordine_insert_modulo();


/* T9) Cacolo del totale ordine in caso di rimozione dei moduli che lo compongono */
CREATE OR REPLACE FUNCTION ordine_delete_modulo()
RETURNS TRIGGER AS $ordine_delete_modulo$
   BEGIN
       IF (OLD.a_stato = 'Aperto')
       THEN
            UPDATE Ordine
            SET a_imponibile = a_imponibile - OLD.a_valore
            WHERE Ordine.a_id_ordine = OLD.a_id_ordine_fk;
        END IF;

       UPDATE Ordine
        SET a_nr_moduli = a_nr_moduli - 1
        WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk;

      RETURN NEW;
   END;
$ordine_delete_modulo$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS ordine_delete_modulo /* in caso esista già */
ON Modulo;

CREATE TRIGGER ordine_delete_modulo
AFTER DELETE ON Modulo
FOR EACH ROW EXECUTE PROCEDURE ordine_delete_modulo();
```

```
/* T10) Cacolo del totale ordine in caso di modifica dei moduli che lo compongono.
   Il prezzo teoricamente non dovrebbe mai venire modificato in quanto è pescato dal
listino */
CREATE OR REPLACE FUNCTION ordine_update_modulo()
RETURNS TRIGGER AS $ordine_update_modulo$
   BEGIN
       IF (OLD.a_stato = 'Aperto')
       THEN
            UPDATE Ordine
            SET a_imponibile = a_imponibile - OLD.a_valore
            WHERE Ordine.a_id_ordine = OLD.a_id_ordine_fk;
        END IF;

       IF (NEW.a_stato = 'Aperto') /*può essere stato modificato */
       THEN
            UPDATE Ordine
            SET a_imponibile = a_imponibile + NEW.a_valore
            WHERE Ordine.a_id_ordine = NEW.a_id_ordine_fk;
        END IF;

      RETURN NEW;
   END;
$ordine_update_modulo$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS ordine_update_modulo /* in caso esista già */
ON Modulo;

CREATE TRIGGER ordine_update_modulo
AFTER UPDATE ON Modulo
FOR EACH ROW EXECUTE PROCEDURE ordine_update_modulo();


/* ******************************************************** LOG
************************************************** */
/* T11) Log delle modifiche alle tabelle del db */
CREATE OR REPLACE FUNCTION log_generico_modifiche()
RETURNS TRIGGER AS $log_generico_modifiche$
   BEGIN
        IF  TG_OP = 'UPDATE'
        THEN
            INSERT INTO LogModifiche (a_nome_tabella, a_operazione, a_new_val, a_old_val)
            VALUES (TG_RELNAME, TG_OP, row_to_json(NEW), row_to_json(OLD));
            RETURN NEW;
        ELSIF TG_OP = 'DELETE'
        THEN
            INSERT INTO LogModifiche (a_nome_tabella, a_operazione, a_old_val)
            VALUES (TG_RELNAME, TG_OP, row_to_json(OLD));
            RETURN OLD;
        ELSIF TG_OP = 'INSERT'
        THEN
            INSERT INTO LogModifiche (a_nome_tabella, a_operazione, a_new_val)
            VALUES (TG_RELNAME, TG_OP, row_to_json(NEW));
            RETURN NEW;
        END IF;

      RETURN NULL;
   END;
$log_generico_modifiche$ LANGUAGE plpgsql;

/* Categoria */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Categoria;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Categoria*/ /* Memorizzare l'interno nel log anche
per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Categoria
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* SottoCategoria */
```

```sql
DROP TRIGGER IF EXISTS log_generico_modifiche
ON SottoCategoria;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON SottoCategoria*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON SottoCategoria
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Nazione */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Nazione;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Nazione*/ /* Memorizzare l'interno nel log anche
per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Nazione
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Comune */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Comune;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Comune*/ /* Memorizzare l'interno nel log anche per
gli insert non appare utile */
AFTER UPDATE OR DELETE ON Comune
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Produttore */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Produttore;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Produttore*/ /* Memorizzare l'interno nel log anche
per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Produttore
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Installatore */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Installatore;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Installatore*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Installatore
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Sede Installazione */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON SedeInstallazione;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON SedeInstallazione*/ /* Memorizzare l'interno nel
log anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON SedeInstallazione
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Tipologia */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Tipologia;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Tipologia*/ /* Memorizzare l'interno nel log anche
per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Tipologia
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Listino Base */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON ListinoBase;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON ListinoBase*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON ListinoBase
```

```
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Listino Personalizzato */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON ListinoPersonalizzato;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON ListinoPersonalizzato*/ /* Memorizzare l'interno
nel log anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON ListinoPersonalizzato
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Ordine */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Ordine;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Ordine*/ /* Memorizzare l'interno nel log anche per
gli insert non appare utile */
AFTER UPDATE OR DELETE ON Ordine
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Modulo */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Modulo;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Modulo*/ /*troppo costoso loggare ogni import di
milioni di record */
AFTER UPDATE OR DELETE ON Modulo
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Tracciato */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON Tracciato;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON Tracciato*/ /* Memorizzare l'interno nel log anche
per gli insert non appare utile */
AFTER UPDATE OR DELETE ON Tracciato
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Utenti, potrebbe non servire se si usano gli altri log */
/* Utente Admin */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON UtenteAdmin;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON UtenteAdmin*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON UtenteAdmin
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Utente Lettura */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON UtenteLettura;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON UtenteLettura*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON UtenteLettura
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Utente Produttore */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON UtenteProduttore;
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON UtenteProduttore*/ /* Memorizzare l'interno nel log
anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON UtenteProduttore
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();

/* Utente Installatore */
DROP TRIGGER IF EXISTS log_generico_modifiche
ON UtenteInstallatore;
```

```
CREATE TRIGGER log_generico_modifiche
/*AFTER INSERT OR UPDATE OR DELETE ON UtenteInstallatore*/ /* Memorizzare
l'interno nel log anche per gli insert non appare utile */
AFTER UPDATE OR DELETE ON UtenteInstallatore
FOR EACH ROW EXECUTE PROCEDURE log_generico_modifiche();
```

A4 – Creazione degli Indici

```
SET search_path TO public;

/* ************************************************ CATEGORIA
******************************************************* */
/* Gli indici su id e campo nome (unique) sono creati automaticamente */

/* ************************************************ SOTTOCATEGORIA
************************************************** */
/* Gli indici su id e campo nome (unique) sono creati automaticamente */
CREATE INDEX sottocategoria_index_a_id_cat_padre_fk ON Sottocategoria
(a_id_categoria_padre_fk);

/* ************************************************ NAZIONE
******************************************************* */
/* Gli indici su id, nome (unique) e codice istat (unique) sono creati automaticamente */

/* ************************************************ COMUNE
******************************************************* */
/* Gli indici su id, nome (unique), cap (unique) e codice istat (unique) sono creati
automaticamente */
CREATE INDEX comune_index_a_id_nazione_fk ON Comune (a_id_nazione_fk);
CREATE INDEX comune_index_a_nome ON Comune (a_nome);

/* ************************************************ PRODUTTORE
******************************************************* */
/* Gli indici su id, piva (unique), sono creati automaticamente */
CREATE INDEX produttore_index_a_cf ON Produttore (a_cf);
CREATE INDEX produttore_index_a_rag_sociale ON Produttore (a_rag_sociale);

/* ************************************************ INSTALLATORE
******************************************************* */
/* Gli indici su id, piva (unique), sono creati automaticamente */
CREATE INDEX installatore_index_a_cf ON Installatore (a_cf);
CREATE INDEX installatore_index_a_rag_sociale ON Installatore (a_rag_sociale);

/* ************************************************ SEDE INSTALLAZIONE
******************************************************* */
/* Gli indici su id, nome (unique), sono creati automaticamente */
CREATE INDEX sedeinstallazione_index_a_id_nazione_fk ON SedeInstallazione
(a_id_nazione_fk);
CREATE INDEX sedeinstallazione_index_a_nome ON SedeInstallazione (a_nome);

/* ************************************************ TIPOLOGIA
******************************************************* */
/* Gli indici su id e campo nome (unique) sono creati automaticamente */

/* ************************************************ LISTINO BASE
******************************************************* */
/* Gli indici su id e ed i seguenti campi unique sono creati automaticamente: codice
listino, composto, composto anno,
   vecchio codice listino, vecchio composto, vecchio composto anno */
CREATE INDEX listinobase_index_a_anno ON ListinoBase (a_anno);
CREATE INDEX listinobase_index_a_trimestre ON ListinoBase (a_trimestre);
CREATE INDEX listinobase_index_a_categoria_fk ON ListinoBase (a_id_categoria_fk);
CREATE INDEX listinobase_index_a_sottocategoria_fk ON ListinoBase
(a_id_sottocategoria_fk);
CREATE INDEX listinobase_index_a_tipologia_fk ON ListinoBase (a_id_tipologia_fk);
CREATE INDEX listinobase_index_a_nome ON ListinoBase (a_nome);
```

```
/* ********************************************** LISTINO PERSONALIZZATO
****************************************** */
/* Gli indici su id e ed i seguenti campi unique sono creati automaticamente: codice
listino, composto, composto anno,
   vecchio composto, vecchio composto anno */
CREATE INDEX listinopersonalizzato_index_z_vecchio_cod_listino ON ListinoPersonalizzato
(z_vecchio_codice_listino);
CREATE INDEX listinopersonalizzato_index_a_listino_base_fk ON ListinoPersonalizzato
(a_id_listino_base_fk);
CREATE INDEX listinobase_index_a_produttore_fk ON ListinoPersonalizzato
(a_id_produttore_fk);

/* ********************************************** ORDINE
********************************************************** */
/* Gli indici su id e codice ordine (unique) sono creati automaticamente */
CREATE INDEX ordine_index_a_produttore_fk ON Ordine (a_id_produttore_fk);
CREATE INDEX ordine_index_a_anno ON Ordine (a_anno);
CREATE INDEX ordine_index_a_trimestre ON Ordine (a_trimestre);
CREATE INDEX ordine_index_a_data_invio ON Ordine (a_data_invio);
CREATE INDEX ordine_index_a_pagato ON Ordine (a_pagato);
CREATE INDEX ordine_index_a_stato ON Ordine (a_stato);
CREATE INDEX ordine_index_a_attivo ON Ordine (a_attivo);

/* ********************************************** MODULO
********************************************************** */
/* Gli indici su id, codice sistema, codice modulo iva (unique) sono creati
automaticamente */
CREATE INDEX modulo_index_a_ordine_fk ON Modulo (a_id_ordine_fk);
CREATE INDEX modulo_index_a_listino_fk ON Modulo (a_id_listino_personalizzato_fk);
CREATE INDEX modulo_index_a_sede_installazione_fk ON Modulo (a_id_sede_installazione_fk);
CREATE INDEX modulo_index_a_installatore_fk ON Modulo (a_id_installatore_fk);
CREATE INDEX modulo_index_a_tracciato_fk ON Modulo (a_id_tracciato_fk);
CREATE INDEX modulo_index_a_codice_modulo ON Modulo (a_codice_modulo);
CREATE INDEX modulo_index_a_modello ON Modulo (a_modulo_modello);
CREATE INDEX modulo_index_a_pagato ON Modulo (a_pagato);
CREATE INDEX modulo_index_a_stato ON Modulo (a_stato);
CREATE INDEX modulo_index_a_attivo ON Modulo (a_attivo);
CREATE INDEX modulo_index_a_modulo_attivo ON Modulo (a_modulo_attivo);
CREATE INDEX modulo_index_z_data_installazione ON Modulo (z_data_installazione);
CREATE INDEX modulo_index_z_data_attivazione ON Modulo (z_data_attivazione);

/* ********************************************** TRACCIATO
********************************************************** */
/* Indice sull'id creato automaticamente */
CREATE INDEX tracciato_index_a_ordine_fk ON Tracciato (a_id_ordine_fk);

/* ********************************************** LOG
********************************************************** */
/* I log non sono indicizzati (se non per la pkey) in quanto sarebbe troppo
oneroso, oltre che superfluo */
```