Università degli studi
di Modena e Reggio Emilia

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

Progettazione di un sistema di Business Intelligence tramite Data Warehousing

Romagnoli Mirco

Tesi di Laurea

Relatore:
Ing. Riccardo Martoglia

Anno Accademico 2017/2018

Ringraziamenti

All'Ing. Riccardo Martoglia per il suo aiuto durante la stesura della tesi e per la sua enorme disponibilità.
A Barbara, che in poco tempo è diventata il mio faro e mi ha accompagnato fin qui.
Ai miei genitori, che mi hanno spronato con pazienza per arrivare a questo traguardo.
A Luana e a tutti i ragazzi del 24/7 che mi hanno dato questa possibilità di crescita.

Parole chiave

Data warehouse
OLAP
CRM
SSAS
Ticket

Indice

Introduzione **1**

I Stato dell'arte **3**

1 Situazione iniziale **5**
1.1 Problematiche 6
1.2 Richieste 6

2 Piattaforma e strumenti utilizzati **8**
2.1 Data warehouse 8
2.1.1 Struttura 9
2.1.2 On-Line Analytical Processing (OLAP) 9
2.1.3 Strutture del modello 10
2.1.4 Gerarchie 12
2.1.5 Operazioni 12
2.1.6 Problematiche 13
2.2 SQL Server Analysis Services (SSAS) 13
2.2.1 Creazione 14
2.2.2 Gestione 15
2.2.3 Interrogazione 16
2.3 SQL Server 17
2.4 Microsoft Dynamics 17
2.4.1 Ticketing 18
2.5 Django 19
2.5.1 Model 19
2.5.2 View 20
2.5.3 Controller 20
2.6 Pywin32 21
2.7 Excel 21
2.7.1 Power Query 21
2.7.2 Visual Basic for Applications (VBA) 22
2.8 Power BI 23

II Progetto e implementazione 25

3 Requisiti 26
3.1 Import . . . 26
3.2 Warehouse . . . 27
3.3 Report . . . 27
3.3.1 Report in Power BI . . . 27
3.3.2 Report in Excel . . . 28

4 Progettazione 30
4.1 Import . . . 30
4.2 Warehouse . . . 31
4.3 Report . . . 34
4.3.1 Power BI . . . 35
4.3.2 Excel . . . 35

5 Implementazione 40
5.1 Import . . . 40
5.2 Warehouse . . . 43
5.2.1 Modello . . . 44
5.2.2 Tickets . . . 44
5.2.3 Schedules . . . 46
5.2.4 Dates . . . 47
5.2.5 Subjects . . . 49
5.2.6 Plants . . . 49
5.2.7 Geographies . . . 50
5.3 Dimensioni . . . 51
5.3.1 Owners . . . 52
5.3.2 Dates . . . 53
5.3.3 Plants . . . 53
5.3.4 Subjects . . . 54
5.4 Misure . . . 55
5.5 Ruoli . . . 57
5.5.1 Reader . . . 57
5.5.2 Administrator . . . 57
5.5.3 ProcessRole . . . 57
5.6 Distribuzione . . . 58
5.7 Job di Import ed Elaborazione . . . 58
5.8 MDX . . . 60
5.8.1 Query 1 . . . 60
5.8.2 Query 2 . . . 61
5.8.3 Query 3 . . . 62
5.9 Report Power BI . . . 63

5.10 Report Excel 64
5.10.1 Report Configurator 64
5.10.2 Report Sender 65
5.10.3 Excel template 69

Conclusione **72**

Elenco delle figure

2.1 Forma minima per un data warehouse 9
2.2 Rappresentazione 3D di un cubo multidimensionale 10
2.3 Esempi di schemi 11
2.4 Schermata per la modifica di una dimensione 14
2.5 Power Query: schermata di trasformazione dei dati 21
2.6 Power BI: Interfaccia Power BI Desktop 24

4.1 Modello del progetto 32
4.2 Schema della struttura del progetto 34
4.3 Diagramma ER del database per la gestione dei report 36
4.4 Diagramma UML della struttura di Report Config 38

5.1 Suggerimento automatico di misure alla creazione di un cubo . . . 52
5.2 Schermata di configurazione delle dimensioni 52
5.3 Esempio di gerarchia incompleta 54
5.4 Finestra di progettazione del cubo 55
5.5 Finestra di modifica dei ruoli 58
5.6 Finestra delle proprietà del progetto 59
5.7 Risultati della query 1 61
5.8 Risultati della query 2 62
5.9 Risultati della query 3 63
5.10 Report con Power BI 64
5.11 Funzionalità di Report Configurator 65
5.12 Template del foglio di lavoro Timing 71

Introduzione

L'avvento dell'industria 4.0 ha portato le aziende sempre più vicine al mondo informatico, al giorno d'oggi non è più sufficiente utilizzare solamente una suite da ufficio per dirsi informatizzati ma è fondamentale riuscire a sfruttare tutti i dati che l'azienda produce con il normale svolgimento del proprio lavoro per poter prendere decisioni rapide ed efficaci, requisiti importantissimi per governare con successo un'azienda.

In questo ambito la Business Intelligence gioca un ruolo particolarmente rilevante nel decision making di un'azienda in quanto le informazioni possono incrementare il vantaggio competitivo.

Con il passare degli anni la tecnologia ha fornito soluzioni via via sempre più potenti e capienti a prezzi minori, creando un ecosistema favorevole allo sviluppo di tecniche di business intelligence sempre più complesse.

Una di queste è il Data Warehousing, uno strumento che si occupa di conservare i dati prodotti da un'azienda in modo tale da conservare la conoscenza storica di ciò che è successo, per favorire le scelte future e monitorare il presente. Il Data Warehousing è prosperato grazie alla riduzione di costo delle memorie e alla capacità di calcolo sempre crescente dei processori, in questo modo si è arrivati a poter sviluppare piccoli-medi progetti senza la necessità di avere data-center costosi.

Questo ha permesso alle aziende di costruire sistemi di Business Intelligence personalizzati, sfruttando la miriade di software di progettazione disponibili che sono nati nel tempo, specificando come e cosa vedere.

In questa tesi si mostrerà il processo di progettazione e creazione di un data warehouse per un'azienda, progetto che è servito per monitorare e prendere decisioni riguardo ad un servizio fornito ai clienti.

La tesi è strutturata in due parti:

1. **Stato dell'arte**:

   Situazione iniziale: in questo capitolo si tratterà della situazione precedente all'inizio del progetto, dello stato dell'infrastruttura informatica a supporto dell'ufficio e delle richieste che l'azienda ha posto per la creazione del data warehouse.

   Piattaforma e strumenti utilizzati: qui è presente una panoramica degli strumenti che sono stati utilizzati per la creazione del progetto, sufficientemente dettagliata per comprendere senza sforzi i contenuti del rimanente testo.

2. **Progetto e implementazione**:

   Requisiti: capitolo che contiene una descrizione formale del comportamento richiesto al progetto, suddiviso per le sue parti di Import, Warehousing e Reportistica.

   Progettazione: nel capitolo di progettazione si descriveranno i passaggi decisionali che hanno portato alla formazione del data warehouse, un passaggio fondamentale per la buona riuscita del progetto che ha richiesto un buon numero di decisioni.

   Implementazione: questo capitolo prettamente tecnico tratterà dello sviluppo materiale del progetto, del codice utilizzato in ogni sua parte e di come si è arrivati a collegare le diverse parti per farle lavorare insieme.

Parte I

Stato dell'arte

Capitolo 1

Situazione iniziale

System Logistics S.p.A. è un'azienda nel settore dell'automazione nell'ambito dei magazzini automatici. Un magazzino automatico si occupa della movimentazione e lo stoccaggio di materiali senza l'ausilio dell'essere umano. Prende in automatico decisioni riguardanti quali sono i migliori prodotti da spedire per selezionare gli ordini e qual'è il miglior posto in cui collocare il materiale per ottimizzare le operazioni di movimentazione all'interno del magazzino. Inoltre la sua struttura consente di costruire il magazzino verticalmente piuttosto che orizzontalmente, riducendo lo spazio necessario per stoccare ciò che l'azienda costruisce o immagazzina.

Essendo il magazzino una parte vitale dell'economia di un'azienda è fondamentale che sia sempre operativo e al massimo della sua capacità, per questo System Logistics fornisce un servizio di assistenza elettro-meccanica e software ai propri clienti.

La scelta dell'azienda di concentrarsi sul mondo dell'industria beverage e della grande distribuzione ha reso ancora più importante avere un servizio di assistenza efficace in quanto si tratta di settori che per motivazioni logistiche hanno tempi molto stretti per l'approvvigionamento sia alla produzione che ai distributori, basti pensare alle industrie del latte e derivati.

L'assistenza software è effettuata da remoto, 24/7, con diverse metodologie di collegamento, al fine di garantire il minor tempo di risposta possibile mentre quella elettro-meccanica unisce l'assistenza da remoto ad interventi in loco per problematiche più tecniche dove si richiede l'intervento di un tecnico specializzato sulle macchine.

L'ufficio si occupa anche dello sviluppo e dell'applicazione di modifiche soft-

ware, riguardanti generalmente alcuni algoritmi di gestione nella movimentazione dei pallet o dell'interfaccia utente, talvolta vengono aggiunte funzioni personalizzate per il cliente.

1.1 Problematiche

Al momento non esisteva un sistema di reportistica periodico e concordato con tutti gli stakeholder, il software CRM permette la creazione di alcuni report ma si basano per lo più sul numero di ticket aperti o chiusi, filtrabili per operatore o cliente.

Per il resto delle operazioni venivano effettuate delle estrazioni ad-hoc su richiesta, i dati grezzi venivano recuperati direttamente dal database del software CRM e raffinati successivamente a mano tramite Excel.

Volendo controllare più efficacemente l'operato dell'ufficio di assistenza è diventato necessario per i dirigenti misurare le prestazioni del reparto monitorando lo stato dell'attività degli operatori, questo è stato effettuato tramite diversi KPI elencati di seguito.

1.2 Richieste

L'azienda ha richiesto di creare un data warehouse da popolare a partire dai dati sui ticket aperti a partire dal 2009, questi ticket sono stati immagazzinati all'interno del software CRM.

I KPI più importanti che sono stati richiesti sono:

- **Tempo di assegnazione**: tempo intercorso tra l'apertura del ticket e l'assegnazione ad un operatore
- **Tempo di chiusura**: tempo tra l'apertura del ticket e la sua chiusura
- **Ticket aperti in reperibilità**: sono state definite delle fasce di lavoro in reperibilità, queste fasce vanno da mezzanotte alle 8 di mattina nei giorni festivi e tutti i giorni feriali
- **Saturazione delle risorse**: tempo impiegato dagli operatori sui ticket o sulle modifiche

- **Numero di ticket aperti**
- **Numero di ticket chiusi**
- **Ticket aperti per categoria**

Questi KPI sono stati richiesti per cercare di avere delle statistiche che permettessero di capire quali fossero i flussi di lavoro in entrata sul reparto, per capire quanto e come lavorassero gli operatori e controllare che i compiti di apertura, assegnazione e chiusura dei ticket venissero svolti regolarmente.

Per gran parte di questi KPI è stata richiesto sia il dato puntuale che l'andamento mensile per monitorare la variazione nel tempo

Sui KPI del tempo di assegnazione e sulla saturazione delle risorse è stato anche indicato un valore target che si richiede di rispettare al meglio delle possibilità.

Capitolo 2

Piattaforma e strumenti utilizzati

Per la realizzazione del progetto sono state utilizzate differenti tecnologie per ottenere il risultato, di seguito verranno elencate le più importanti per comprendere il resto del progetto.

2.1 Data warehouse

Le decisioni in ambito aziendale vengono prese in base alle analisi sulle situazioni passate e presenti dell'azienda, inoltre si cerca di analizzare il mercato ed effettuare previsioni per il futuro, ritrovandosi perciò con una grande mole di dati proveniente da una moltitudine di origini differenti collezionata durante la vita dell'azienda, l'accesso a questi dati è molto utile per avere una visione d'insieme dell'impresa.

Uno dei metodi migliori per integrare tra di loro tutte queste informazioni è il **data warehouse**, un repository dove tutte le informazioni dell'azienda vengono immagazzinate, dopo essere state estratte, filtrate ed unite tra loro. A livello pratico si tratta di un database dalle **grandi dimensioni**, dell'ordine di grandezza dei Terabyte.

Un data warehouse si occupa di **raccogliere** e **consolidare** i dati periodicamente, mantenendo anni di storico.

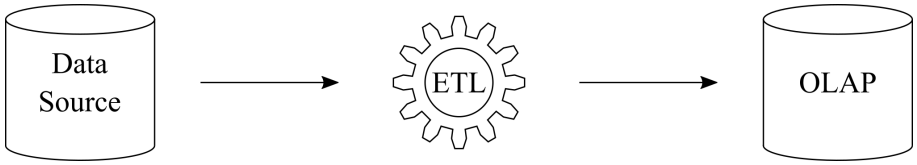


Figura 2.1: Forma minima per un data warehouse

2.1.1 Struttura

Un data warehouse consiste di più applicazioni che collaborano tra loro, una sua forma minima si può vedere in figura 2.1: se viene eliminato uno di questi elementi dalla struttura il data warehouse cessa di essere tale. Il data source è unico, la fase di *extract, transform and load* inserisce i dati all'interno della struttura multidimensionale. Nel campo della Business Intelligence questa struttura è comunque più utile rispetto all'interrogazione diretta del data source perchè i dati sono **organizzati** per dimensioni, di cui si parlerà nel prossimo paragrafo, che permettono una analisi più in linea a quelle che sono le esigenze aziendali.

2.1.2 On-Line Analytical Processing (OLAP)

Il contenuto di un data warehouse viene analizzato tramite l'**OLAP**: un insieme di tecniche software per l'**analisi interattiva** e **veloce** di grandi quantità di dati che è possibile esaminare in modalità piuttosto complesse.

Una struttura OLAP è chiamata cubo multidimensionale ed è definito da *dimensioni* e *fact*.

Le **dimensioni** sono le entità di cui si vuole tenere traccia, possono essere organizzate in maniera gerarchica, esempi di dimensioni possono essere: il tempo, il luogo geografico, una particolare filiale dell'azienda.

I **fact** rappresentano il tema centrale del modello, misure numeriche di cui si vuole analizzare la relazione tra le dimensioni, possibili fact possono essere: il numero e l'ammontare delle vendite, il numero di ticket e il loro tempo di risoluzione.

Nel caso dell'immagine 2.2 abbiamo le dimensioni che sono: oggetto, luogo e tempo mentre i fact sono i numeri all'interno dei cubi, che rappresentano il numero di oggetti venduti. In questo specifico caso abbiamo tre dimensioni ma

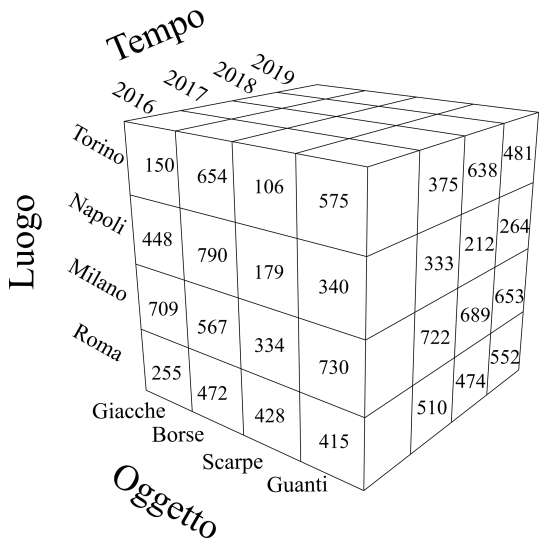


Figura 2.2: Rappresentazione 3D di un cubo multidimensionale

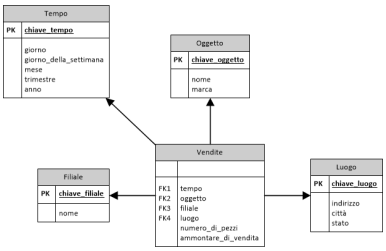
possono esisterne n, per visualizzarne quattro, ad esempio, si può immaginare di accostare questi cubi uno di fianco all'altro introducendo un ulteriore dimensione.

2.1.3 Strutture del modello

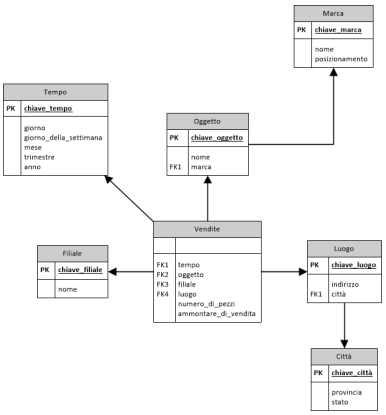
Al contrario dei database transazionali, dove un diagramma ER associa entità e relazioni, imponendo dei vincoli ben precisi per il corretto funzionamento del database, un modello multidimensionale, per poter funzionare correttamente, ha bisogno di uno **schema** conciso per permettere un'analisi veloce di una grande mole di dati. Gli schemi più popolari sono i seguenti:

- **Stella** è il più comune, possiede una tabella centrale molto grande che rappresenta i fact con una serie di tabelle collegate a quest'ultima che rappresentano le dimensioni, una per ogni dimensione.

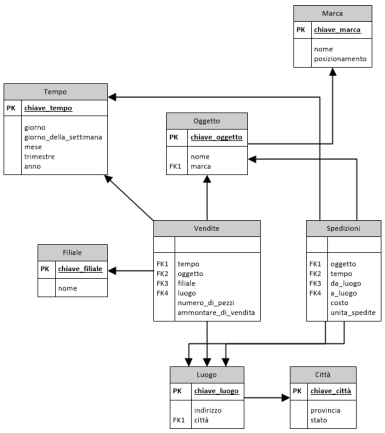
- **Fiocco di neve** è una variante dello schema a stella, permette di ridurre la ridondanza (di conseguenza lo spazio occupato) a discapito delle prestazioni che risultano ridotte a causa del maggior numero di join a parità di query.



(a) Stella



(b) Fiocco di neve



(c) Facts constellation

Figura 2.3: Esempi di schemi

- **Facts constellation** in questo schema le tabelle dei fact potrebbero dover condividere le tabelle delle dimensioni, possono essere viste come un insieme di schemi a stella, creando quindi una "costellazione".

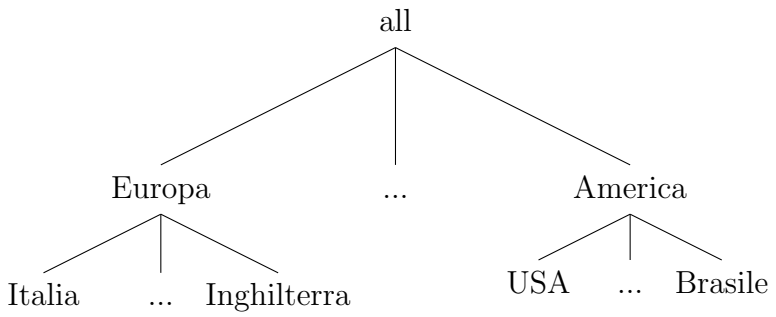
2.1.4 Gerarchie

Una gerarchia definisce una **associazione** tra un insieme di concetti di basso livello ad altri di livello più alto. Le dimensioni possono essere raggruppate in gerarchie, si pensi alla dimensione *Tempo* che naturalmente può essere ordinata come:

Anno
|
Semestre
|
Trimestre
|
Mese
|
Settimana
|
Giorno

Queste gerarchie possono essere "navigate" in modo tale da ottenere livelli di **dettaglio** più o meno precisi.

Le gerarchie si possono vedere come degli alberi alla cui radice si trova la gerarchia *all* da cui si diramano tutti i rami della gerarchia.



Esiste un tipo di gerarchie, detta *incompleta*, dove le foglie dell'albero non si trovano tutte sullo stesso livello, alcuni rami possono terminare prima, altri dopo, ad esempio nella gerarchia indicata qui sopra potrebbe essere interessante introdurre anche le province italiane, questo rende l'albero sbilanciato.

2.1.5 Operazioni

Su un cubo multidimensionale è possibile effettuare alcuni tipi di operazioni:

- **Roll-up**: esegue un'aggregazione sul cubo esistente, scalando la gerarchia di una dimensione oppure eliminandola.

- **Drill-down**: è il contrario di *roll-up*, si aumenta il dettaglio dei dati, scalando verso il basso una gerarchia oppure introducendone una nuova.

- **Slice and dice**: si tratta della selezione di un sotto-cubo selezionando un particolare valore di una (*slice*) o più (*dice*) dimensioni.

- **Pivot**: effettua una rotazione dei dati su un asse per fornire una rappresentazione dei dati alternativa.

2.1.6 Problematiche

Alcune problematiche sorte negli anni su queste tecniche di immagazzinamento dati riguardano la **mancanza di standard**, per cui ogni software spinge la propria filosofia di design delle architetture di data warehousing/OLAP, mancano i **design pattern** per aiutare gli sviluppatori nel modellare in maniera standard alcuni problemi e infine la **sicurezza dei modelli**, che assume una particolare importanza in quanto i data warehouse contengono dati cruciali sullo stato dell'azienda, esistono alcuni approcci basati sul controllo degli accessi ma è necessario sviluppare modelli nuovi, in quanto quelli utilizzati dai database transazionali non sono pensati per i data warehouse.

2.2 SQL Server Analysis Services (SSAS)

È un tool per la costruzione di sistemi OLAP e di data mining. Fa parte della famiglia dei tool Microsoft per la business intelligence assieme ai *Reporting Services* e gli *Integration Services* ed è integrato con il DBMS **SQL Server**.

Questo tool consente di **creare**, **gestire** ed **interrogare** un cubo multidimensionale. Tutto ciò avviene all'interno di Visual Studio che permette di interagire con gli analysis services tramite la sua estensione *Data Tools*.

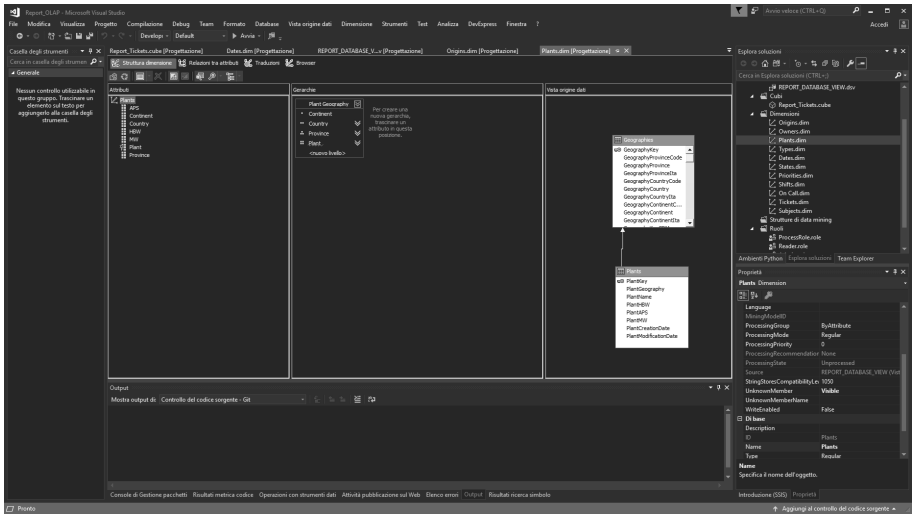


Figura 2.4: Schermata per la modifica di una dimensione

2.2.1 Creazione

La creazione di un cubo avviene innanzitutto collegando SSAS ad un database che si occupa di immagazzinare i dati per il cubo utilizzando una delle strutture indicate nel paragrafo 2.1.3, questo database verrà chiamato **data source**.

Fatto questo sarà possibile scegliere le tabelle da utilizzare all'interno del progetto, creare misure calcolate a partire dai dati grezzi contenuti nell'origine dati ed eventualmente creare tabelle automatiche a supporto delle dimensioni (es. tabella delle date), ottenendo una **data source view** che servirà da base a SSAS per costruire dimensioni e fact.

Da qui si possono creare le dimensioni e le misure, che non sono altro che i fact indicati nel paragrafo 2.1.2. SSAS cerca automaticamente di creare le dimensioni in base alle relazioni con la tabella delle misure e al loro nome, spesso riesce a comprendere correttamente quali sono le dimensioni corrette ma per un controllo più preciso e per **dimensioni complesse** che coinvolgono più tabelle si rende necessaria la **creazione manuale** di una dimensione che viene effettuata per lo più in maniera visuale.

Come si può vedere in figura 2.4 è possibile inserire attributi nelle dimensioni, creare gerarchie e costruire dimensioni a partire da più tabelle a patto che siano relazionate tra loro in qualche modo.

In seguito sarà possibile definire anche le misure del cubo che verranno

aggregate in base alle dimensioni scelte, le misure di base possono essere:

- **Somma** sommatoria sul valore dei figli, questa è la dimensione standard
- **Conteggio delle righe** numero di righe
- **Conteggio dei valori non vuoti** numero di righe il cui valore non è NULL
- **Minimo** valore minimo dei figli
- **Massimo** valore massimo dei figli
- **Conteggio dei valori univoci** conteggio dei valori unici dei figli
- **Nessuna aggregazione** vengono restituiti tutti i valori dei figli
- **Media nel tempo** media di tutti i valori dei figli non nulli
- **Primo valore** primo valore dei figli
- **Ultimo valore** ultimo valore dei figli
- **Primo valore non nullo** analogo a *primo valore* ma si prende il primo non nullo
- **Ultimo valore non nullo** analogo a *ultimo valore* ma si prende l'ultimo non nullo

È possibile creare anche **misure calcolate** a partire dalle misure di base, possono essere utilizzate per aggregare dati più complessi. Si possono anche creare degli **insiemi** che verranno utilizzati come filtri all'interno delle query. Questi calcoli vengono definiti utilizzando *MultiDimensional eXpression* (MDX) di cui si parlerà in seguito nel paragrafo 2.2.3.

2.2.2 Gestione

Come si è detto nel paragrafo 2.1.6 un cubo multidimensionale contiene dati sensibili dell'azienda il cui accesso deve essere controllato. SSAS applica questo controllo tramite gli account di **dominio** Microsoft, permettendo di decidere quali utenti possono leggere i dati effettuando un **controllo granulare**, fornendo gli accessi a determinate dimensioni e non ad altre, in questo modo i

diversi reparti possono interrogare lo stesso cubo utilizzando solo le dimensioni che possono essere interessanti per quel determinato reparto.

Questo comportamento viene modellato tramite i *ruoli*, ogni ruolo ha dei certi permessi sulle dimensioni e sulle misure del cubo. I permessi possono essere di **lettura** ma anche di **elaborazione**, quest'ultimo ruolo riguarda la possibilità di processare i dati ed è molto utile per gli sviluppatori in quanto possono manualmente aggiornare strutture e dati del cubo ma anche creare dei processi automatici che importano i dati ed in seguito li processano.

2.2.3 Interrogazione

L'interrogazione di un cubo multidimensionale è profondamente **diversa** da quella di un database transazionale, quest'ultimo si basa su tabelle composte da righe e colonne mentre un modello multidimensionale si basa su dimensioni, misure, gerarchie e livelli. Per questo scopo è stato creato un nuovo linguaggio di query, chiamato *MultiDimensional eXpression* (MDX), le cui somiglianze con SQL si limitano al ricordare la **sintassi**.

La differenza principale tra MDX e SQL è che MDX si limita ad essere un linguaggio di **interrogazione**, permette cioè esclusivamente l'analisi e la lettura dei dati, non permette la creazione di dimensioni o misure.

Una query MDX ha tipicamente questa struttura:

```
1 SELECT
2    dimensione ON ROWS,
3    dimensione ON COLUMNS
4 FROM
5    cubo
6 WHERE
7    misura
```

L'esecuzione di una query restituisce una tabella dove sulle righe e sulle colonne saranno presenti le dimensioni specificate e all'interno delle celle della tabella sarà presente la misura richiesta. Questo linguaggio è piuttosto flessibile e permette di effettuare diversi tipi di query utilizzando strutture differenti da quella appena vista, si possono specificare dei valori particolari di dimensioni, misure utilizzate sul momento, insiemi di dimensioni diverse e così via.

2.3 SQL Server

SQL Server è un DBMS Relazionale sviluppato da Microsoft. È il terzo database più utilizzato (per db-engines.com, febbraio 2019) e si trova attualmente alla versione 2017.
Esistono numerose versioni di questo DBMS, le più importanti sono:

- **Enterprise**: versione flagship del DBMS, può gestire database fino a 512 petabyte, gestire 12 terabyte di memoria e supporta 640 core.
- **Standard**: simile alla versione Enterprise ma con alcune limitazioni riguardanti numero di nodi, indici e memoria.
- **Web**: versione low-cost per il web hosting.
- **Business Intelligence**: include la versione Standard e alcuni tool di Business Intelligence
- **Express**: versione gratuita di SQL Server, limitata ad un solo processore, 1 gigabyte di memoria e 10 gigabyte per file.

Il linguaggio utilizzato è il **T-SQL**, soluzione proprietaria di Microsoft che estende il linguaggio SQL per quanto riguarda DML, DDL e gestione del database. È possibile inoltre estendere le funzioni del database tramite SQL Common Language Runtime che permette di utilizzare C# e VB.NET per creare stored procedure, trigger, funzioni e tipi di dati definiti dall'utente.

Fornisce inoltre numerosi strumenti per l'utilizzo e la manutenzione dei database come SQL Server Management Studio e Visual Studio per poter interagire graficamente con il server e **SQL Agent** per la manutenzione dei database.

2.4 Microsoft Dynamics

Dynamics è una piattaforma per la pianificazione delle risorse (ERP) e la gestione delle relazioni con i clienti (CRM). La piattaforma offre una serie di servizi per la gestione del rapporto con i clienti, le vendite, i servizi e il marketing.
Tramite questa piattaforma è possibile gestire in maniera automatizzata numerosi aspetti di una società, per quanto riguarda la parte di pianificazione delle risorse va ad incidere sulle risorse umane, sulla gestione della supply chain e la

contabilità, ad esempio; per la parte di relazioni con i clienti si possono gestire: clienti acquisiti e potenziali, campagne di marketing, ecc.

La piattaforma può essere personalizzata sotto diversi aspetti: campi aggiuntivi, workflow, invio di e-mail all'occorrenza di un evento, ecc.

2.4.1 Ticketing

Dynamics fornisce una piattaforma per la gestione di **ticket** di assistenza, l'intero sistema è **basato su database** SQL Server.

Per Dynamics un ticket viene chiamato "caso" ed è visto come un record associato ad un **cliente** e ad un **operatore** (che gestirà la richiesta di assistenza), ogni ticket include anche una serie di **impegni** e **note** in modo da tener traccia delle attività che gli operatori hanno portato avanti in seguito alla sua apertura.

All'interno di un ticket sono immagazzinate ulteriori informazioni, come:

- **Stato**: se il ticket è aperto, chiuso o annullato.
- **Descrizione**: breve testo contenente le informazioni iniziali sul problema.
- **Priorità**: il livello di gravità del problema.
- **Tipo**: se il ticket si riferisce ad un problema o ad una richiesta di modifica del software.
- ***Origine***: il problema riguarda il software o l'hardware dell'impianto.
- **Apertura**: il momento in cui la richiesta viene ricevuta e il ticket è stato aperto.
- **Assegnazione**: il momento in cui la richiesta viene presa in carico da un operatore.
- ***Riavvio***: l'istante in cui l'operatore ha risolto il problema ed il cliente è nuovamente in grado di utilizzare il proprio impianto.
- **Chiusura**: il momento in cui il ticket viene chiuso.
- ***Reperibilità***: se il ticket è stato aperto durante l'orario serale/notturno o un giorno festivo.

I punti in corsivo sono **attributi personalizzati** dall'azienda per tenere traccia di alcune informazioni utili per la natura dell'attività.

I clienti vengono chiamati anche "Impianti", sempre a seguito di una personalizzazione, hanno anch'essi attributi di cui si tiene memoria, prima tra tutti la collocazione geografica dell'impianto e i contatti dell'azienda cliente con cui un operatore si può interfacciare.

A seguito di personalizzazioni si tiene anche traccia della tipologia di impianto:

- **HBW**: High Bay Warehouse, magazzino automatico
- **APS**: Automatic Picking System, sistema per la composizione di pallet tramite sistemi robotizzati
- **MW**: Manual Warehouse

Gli impegni all'interno di un ticket hanno un titolo, una descrizione ed una durata, in modo da poter scalare i tempi dai contratti dei clienti o fatturare ore di assistenza nel caso un cliente non abbia un contratto di servizio attivo con un monte ore.

È presente infine una dicitura all'interno dell'azienda, detta **Indirettismo** e abbreviata con TL, che viene identificata da una commessa "TL0000007", questo tipo di commessa indica le ore spese da un operatore all'esterno di un problema, una richiesta o una modifica, l'indirettismo può nascere da una mancanza di lavoro o da attività svolte internamente all'azienda.

2.5 Django

È un framework basato su **Python** che permette la creazione di applicazioni **web** seguendo il modello Model-View-Controller (MVC).

2.5.1 Model

I **modelli** in Django forniscono allo sviluppatore una visione astratta del database, la classe principale *Model* permette di definire i campi e i metodi di un oggetto.

Esistono anche campi particolari che implementano **relazioni** uno-a-uno e molti-a-molti, senza dover prevedere tabelle specializzate, esse verranno create

automaticamente dal framework tramite le migrazioni che si occupano di scrivere sul database i cambiamenti del model, grazie alle migrazioni è possibile ricostruire tutte le trasformazioni all'interno del model.

Il model si occupa anche delle operazioni **transazionali** che possono essere decise dal framework o controllate dal programmatore.

I DBMS supportati ufficialmente sono: PostgreSQL, MySQL, Oracle e SQLite; sono supportati tramite l'utilizzo di moduli di terze parti anche SQL Server, Db2 ed anche database NoSQL come MongoDB e Google App Engine.

2.5.2 View

Le **view** gestiscono la **logica** delle richieste fatte dagli utenti, al loro interno si vanno ad effettuare le operazioni che l'utente ha richiesto inviando la richiesta ad un certo indirizzo. All'interno delle view si possono definire i comportamenti delle richieste GET e POST.

Al loro interno è buona norma gestire solamente i dati relativi alla richiesta effettuata, per la composizione della risposta si possono utilizzare i **template**, oggetti specializzati che prendono i dati che la view espone e li utilizzano per creare una pagina HTML, la view si impegnerà a **renderizzare** il template ed inviare la risposta all'utente.

Oltre a pagine HTML si possono inviare come risposta all'utente delle risposte in JSON, rendendo Django un framework utile anche per lo sviluppo di API per servizi web.

2.5.3 Controller

Il **controller** gestisce le richieste inviate dall'utente, indirizzando una richiesta ad un particolare **URL** alla **view** designata.

Lo sviluppatore dovrà definire le **espressioni regolari** che si associano alle varie richieste e contestualmente la view da eseguire, il controller in fase di esecuzione del server si occuperà di smistare le richieste.

All'interno dell'URL si possono anche **definire parametri** che la view utilizzerà per personalizzare la risposta.

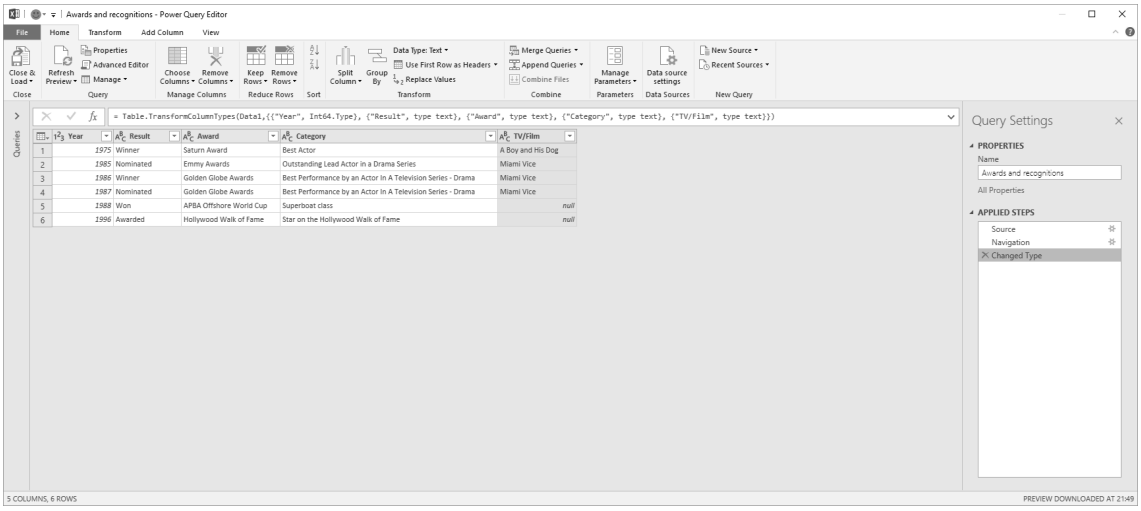


Figura 2.5: Power Query: schermata di trasformazione dei dati

2.6 Pywin32

È una **libreria** che fornisce una soluzione per utilizzare l'**interfaccia COM** tramite Python.

Questa interfaccia permette la **creazione** dinamica di oggetti e la **comunicazione** tra processi all'interno dell'ambiente Windows.

In questo modo sarà possibile, ad esempio, all'interno di un programma, la creazione ed il controllo di un nuovo processo o di uno esistente.

2.7 Excel

Excel è uno dei principali strumenti della **suite da ufficio** Microsoft Office, utilizzata in maniera estensiva soprattutto in ambienti aziendali grazie alla sua versatilità e la possibilità di produrre facilmente **tabelle** e **grafici** piuttosto complessi.

2.7.1 Power Query

Power Query è un tool che si occupa dell'**analisi** a tutto tondo dei dati tramite Excel, permette la **connessione**, la **trasformazione**, la **combinazione** e la **condivisione** dei dati.

Il tool è presente sia in Excel che in PowerBI, un'altra piattaforma per la Business Intelligence sviluppata da Microsoft che si focalizza sulla visualizzazione e la distribuzione di report, di cui si parlerà in seguito.

Power Query permette di collegarsi a diverse fonti di dati: database, data warehouse, pagine web e file di testo sono alcune tra le numerose tipologie di origini con cui è possibile lavorare, al suo interno si possono **combinare** queste origini eterogenee tra loro per poter lavorare con un unica fonte strutturata.

La **trasformazione** dei dati è uno dei punti forti di questo strumento, per raggiungere lo scopo implementa un **linguaggio specifico**, chiamato M, con cui si possono normalizzare e trasformare i dati grezzi, le operazioni vanno dalle più semplici di inserimento/cancellazione di righe o colonne a quelle più complesse come l'hashing, il join e il merge di tabelle differenti.

È possibile infine **caricare** i risultati su una tabella Excel, in questo modo si potranno visualizzare grafici o effettuare ulteriori operazioni di raffinazione, processo molto utile per la comunicazione con il settore manageriale, che necessita di informazioni di rapida fruizione per il decision making.

2.7.2 Visual Basic for Applications (VBA)

Il "Visual Basic for Applications" è un **sottoinsieme** del linguaggio Visual Basic integrato in alcuni programmi della suite office, non possiede tutte le funzionalità di Visual Basic ma permette di **interagire** con ogni aspetto del programma che lo ospita.

È un linguaggio di alto livello che si divide in **subroutine** e **funzioni**, le subroutine **permettono** di effettuare azioni sull'applicazione che le ospita ma non possono ritornare dei valori, le funzioni possono essere utilizzate all'interno dell'applicazione che le ospita ma non modificarla, possono ritornare dei valori.

Questo linguaggio di scripting risulta molto utile all'interno di Excel in quanto permette di **automatizzare** delle operazioni senza la necessità di interazione con l'utente e dà la possibilità di creare delle **interfacce** grafiche molto semplici per favorire l'esplorazione dei dati all'interno del foglio.

2.8 Power BI

Power BI è uno strumento mirato alla Business Intelligence e sviluppato da Microsoft, nasce a fianco di SQL Server Reporting Services (SSRS) con l'obiettivo di rendere la **creazione** e la **distribuzione** dei **report** più user friendly rispetto a SSRS che tende ad essere leggermente più macchinoso rispetto alla sua controparte.

Si appoggia in gran parte sul **cloud** per la distribuzione e la raccolta dei dati, il design del report avviene invece in **locale**, tramite un'interfaccia drag and drop sull'applicazione Power BI Desktop.

Power BI include Power Query, tool descritto nel paragrafo 2.7.1, può quindi collegarsi alle medesime fonti di dati e applicare le stesse trasformazioni fornendo quindi uno strumento molto **versatile** e **pratico** per la costruzione di report. Per i collegamenti verso SSAS è anche possibile avere una *live connection* dove i dati vengono aggiornati in tempo reale, così come sono presenti nel data warehouse, potendo anche filtrare i dati o effettuare operazioni di drill-through in maniera visuale.

Questi report poi possono essere pubblicati nel cloud in modo che gli interessati possano vedere i report a cui sono interessati in un unico luogo, con i dati e le visualizzazioni più recenti, senza necessariamente trovarsi nella rete aziendale dove spesso i dati sensibili vengono confinati.

L'invio al cloud dei dati dei report viene effettuato tramite un'applicazione chiamata *on-premise data gateway* il cui scopo è quello di trasmettere i dati alla piattaforma cloud di Power BI, che poi si occuperà a sua volta di gestire gli accessi e mappare gli account di accesso a quelli locali dell'azienda.

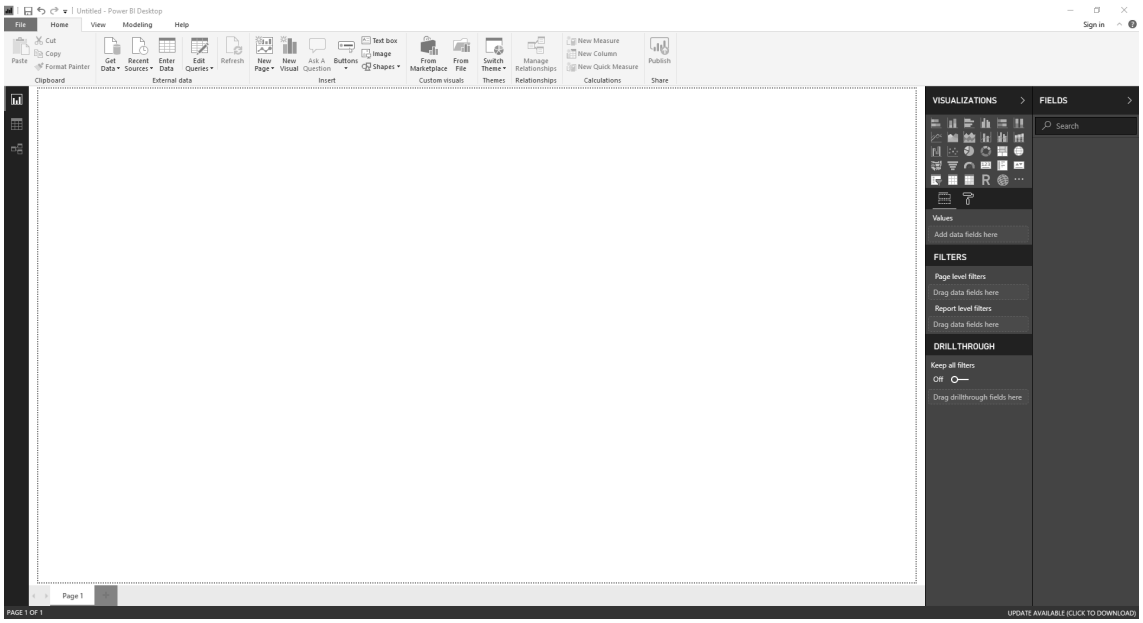


Figura 2.6: Power BI: Interfaccia Power BI Desktop

Parte II

Progetto e implementazione

Capitolo 3

Requisiti

I requisiti non sono particolarmente stringenti, è stato dato un buon grado di **libertà** se non che nelle necessità di modellare alcune entità e nei report, che sono stati concordati con gli stakeholder dell'azienda.

Le 3 macrofasi del progetto sono state circoscritte a: **import**, **warehouse** e **report**, queste fasi sono **indipendenti** tra di loro se non per lo scambio dei dati.

Non è stata eseguita una definizione rigorosa e preventiva dei requisiti, in quanto come spesso accade nelle aziende si possono incontrare necessità in corso d'opera, alcuni sono stati definiti inizialmente, primi tra tutti i **tempi** di assegnazione e chisura dei ticket, altri invece sono nati in seguito, quando il progetto ha iniziato ad acquisire importanza all'interno dell'azienda. Non è stato infrequente, pertanto, che alcuni campi venissero aggiunti a fase di import già operativa.

Per semplicità di lettura, tuttavia, i requisiti verranno descritti in maniera unitaria, come se fossero stati definiti tutti preventivamente.

3.1 Import

Il sistema deve importare i dati dal database esterno di Microsoft Dynamics, i dati da importare sono i seguenti:

- **Ticket**: chiave, cliente, operatore, origine, priorità, stato, tipo, categoria, data di creazione, modifica, apertura, riavvio impianto
- **Impegni**: ticket a cui è associato, proprietario, durata

- **Operatori**: chiave, nome dell'operatore
- **Clienti**: chiave, luogo, nome, HBW, APS, MW, data di creazione e modifica
- **Categorie**: chiave, nome, descrizione, categoria padre, nome padre, data di creazione e modifica

I campi di un ticket possono assumere i seguenti valori:

Campo	Valori
Origine	Hardware, Software
Priorità	Altissima, Alta, Normale, Bassa, Programmabile, Programmata
Stato	Attivo, Risolto, Annullato
Tipo	TL, Non disponibile, Riunione, Problema, Richiesta, Modifica, Manutenzione

I dati devono essere importati periodicamente ogni ora dal database del CRM.

3.2 Warehouse

Per quanto riguarda la struttura del data warehouse non sono stati dati vincoli particolari se non che le dimensioni fossero sufficientemente dettagliate per supportare la creazione dei report.

3.3 Report

Le metodiche per la reportistica sono varie e sono cambiate nel tempo. A fianco delle estrazioni di dati a richiesta sono stati creati due tipologie di report, una tramite Power BI e l'altra con Excel.

3.3.1 Report in Power BI

Il report in Power BI deve contenere le seguenti visualizzazioni:

- **Ticket aperti mensilmente**: numero di ticket, distinti per Origine aperti ogni mese, la visualizzazione deve includere gli ultimi 24 mesi.

- **Ticket in stato "Aperto" ad oggi**: numero di ticket, distinti per Priorità e Origine, aperti in questo mese che al momento si trovano in stato "Aperto".
- **Lista dei ticket in stato "Aperto" con priorità "Altissima"**.
- **Tempo medio di risoluzione in minuti**: tempo trascorso tra l'apertura e la chiusura di ticket espresso in minuti, la visualizzazione deve includere gli ultimi 7 mesi.
- **Tempo medio di assegnazione in minuti**: tempo trascorso tra l'apertura e l'assegnazione ad un operatore del ticket espresso in minuti, la visualizzazione deve includere gli ultimi 7 mesi.
- **Ticket aperti oggi**
- **Tempo medio di risoluzione in minuti**: tempo trascorso tra l'apertura e la chiusura di un ticket dei casi con priorità "Altissima" e "Alta"

I dati devono partire dal mese precedente, in quanto quelli del mese corrente sono troppo soggetti a fluttuazioni.

3.3.2 Report in Excel

Il report in Excel deve essere inviato via mail il primo giorno di ogni mese alle 8:30 ad un certo numero di destinatari variabile nel tempo, in quanto possono interessarsi in seguito ai KPI dell'assistenza oppure smettere di occupare un ruolo in azienda per cui la ricezione del report può avere significato. L'oggetto e il nome del file dovranno avere il seguente formato: "Helpdesk 24/7 performance report - Mese Anno".

Il report deve contenere le seguenti visualizzazioni:

- **Ticket aperti mensilmente**: numero di ticket, distinti per Origine aperti ogni mese, la visualizzazione deve includere gli ultimi 24 mesi.
- **Ticket in stato "Aperto" ad oggi**: numero di ticket, distinti per Priorità e Origine, aperti in questo mese che al momento si trovano in stato "Aperto". È necessario indicare anche il numero totale per questo mese e la differenza con il mese scorso.

- **Ticket chiusi questo mese**: numero di ticket, distinti per Priorità e Origine, chiusi in questo mese. È necessario indicare anche il numero totale per questo mese e la differenza con il mese scorso.
- **Ticket aperti in reperibilità**: numero di ticket, distinti per Origine aperti ogni mese, la visualizzazione deve includere gli ultimi 24 mesi.
- **Ticket per categoria**: tempo speso dagli operatori per ogni categoria di ticket.
- **Saturazione delle risorse**: tempo impiegato dagli operatori su ticket, modifiche o TL (attesa di ticket o attività trasversali) questo mese.
- **Saturazione mensile delle risorse**: tempo impiegato dagli operatori su ticket, modifiche o TL per gli ultimi 12 mesi, per questo grafico è necessario visualizzare un target del 45%.
- **Problemi con priorità molto alta**: ticket con Tipo "Problema" ordinati per ore spese dall'operatore in senso decrescente, la visualizzazione deve includere: Numero Ticket, Origine, Cliente, Priorità, Data di apertura, Ore Spese e Titolo del ticket.
- **Top 20 delle richieste**: prime 20 richieste ordinate per ore spese dall'operatore in senso decrescente, i dati richiesti sono gli stessi della visualizzazione precedente escluso il titolo.
- **Top 20 Modifiche Software**: prime 20 richieste ordinate per ore spese dall'operatore in senso decrescente, i dati richiesti sono gli stessi della visualizzazione precedente.
- **Tempo medio di risoluzione**: in ore, suddiviso per Priorità, devono essere presi in considerazione gli 8 mesi precedenti. È necessario indicare anche il numero totale per questo mese e la differenza con il mese scorso.
- **Tempo medio di assegnazione**: in ore, suddiviso per Priorità, devono essere presi in considerazione gli 8 mesi precedenti. È necessario indicare anche il numero totale per questo mese e la differenza con il mese scorso.
- **Tempo medio di riavvio**: in ore, suddiviso per Priorità, devono essere presi in considerazione gli 8 mesi precedenti. È necessario indicare anche il numero totale per questo mese e la differenza con il mese scorso.

Capitolo 4

Progettazione

La scelta degli strumenti è stata determinata soprattutto da ciò che l'azienda utilizzava già per lo sviluppo e come suite da ufficio, per evitare di introdurre un cambiamento significativo negli strumenti utilizzati e per sfruttare le licenze già disponibili. Per questi motivi si è deciso di adottare l'ecosistema Microsoft, comprendente di SQL Server, SQL Server Analysis Services, Excel e Power BI. Si è preferito **Python** a C# essendo il primo uno strumento già trattato durante il percorso accademico e che permette di produrre risultati in maniera più rapida rispetto a C#, inoltre è stato utilizzato per fornire un semplice servizio di **configurazione** e **invio di e-mail**, che non verrà modificato da chi non conosce il sistema.

Una volta definiti gli strumenti necessari si è passati alla progettazione vera e propria della struttura.

4.1 Import

Per creare un data warehouse bisogna avere una **origine dei dati**, le scelte possibili sono molteplici, si è preferito evitare di dipendere direttamente dal database di Dynamics per evitare di aumentare il carico di lavoro del server che viene utilizzato da più persone contemporaneamente e si è deciso di creare una base di dati **separata**, in modo da essere più **flessibili** nei cambiamenti, mantenere un certo grado di **modularità** tra i sistemi ed una **gestione centralizzata** del cubo multidimensionale.

A seguito dello studio del datasource si è rivelata necessaria una fase di **normalizzazione** dei dati per quanto riguarda alcune dimensioni come quelle geo-

grafiche e temporali, questa fase poteva essere implementata solamente tramite l'utilizzo di tabelle esterne al datasource per motivi legati al fatto che il database di Dynamics non fosse gestito internamente ma tramite un'azienda esterna.
La scelta di mantenere un database **esterno** al datasource ha richiesto la creazione di un sistema che importasse periodicamente i dati per poi normalizzarli. Data la richiesta di aggiornamenti periodici ogni ora si è dovuto trovare un tool che permettesse l'estrazione dei dati periodicamente, anche in questo caso erano possibili diverse soluzioni, la scelta è ricaduta su **SQLAgent**, servizio di schedulazione di processi compreso in SQL Server, il suo scopo è principalmente quello di eseguire job di manutenzione del database ma può essere utilizzato per l'esecuzione di qualsiasi tipo di script SQL (ma anche XMLA, ad esempio). Questo ha portato alla creazione di **stored procedure** che vengono eseguite in sequenza, importando i dati dal datasource configurato come **linked server**, per poter estrarre i dati in modo trasparente e con una sola configurazione iniziale. Le stored procedure si dovranno inoltre occupare della normalizzazione dei dati contestualmente alla loro estrazione.

Le stored procedure che si occupano di importare i dati sono:

Stored procedure	**Importa**	**Da**
sp_import_tickets	Ticket	IncidentBase
sp_import_plants	Clienti	AccountBase $\bowtie$ CustomerAddressBase
sp_import_owners	Operatori	OwnerBase
sp_import_subjects	Categorie	SubjectBase
sp_import_schedules	Impegni	apx_segnaore

Come si può notare dalla differenza nella nomenclatura la tabella degli impegni è stata **personalizzata** dall'azienda.

Una volta definita la fase di import non è ancora possibile però definire la **struttura** del database, in quanto questa sarà decisa durante la progettazione del cubo multidimensionale

4.2 Warehouse

La fase di progettazione del warehouse richiede di effettuare decisioni sul **modello** da utilizzare, sulle **dimensioni** e sulle **misure** da creare.

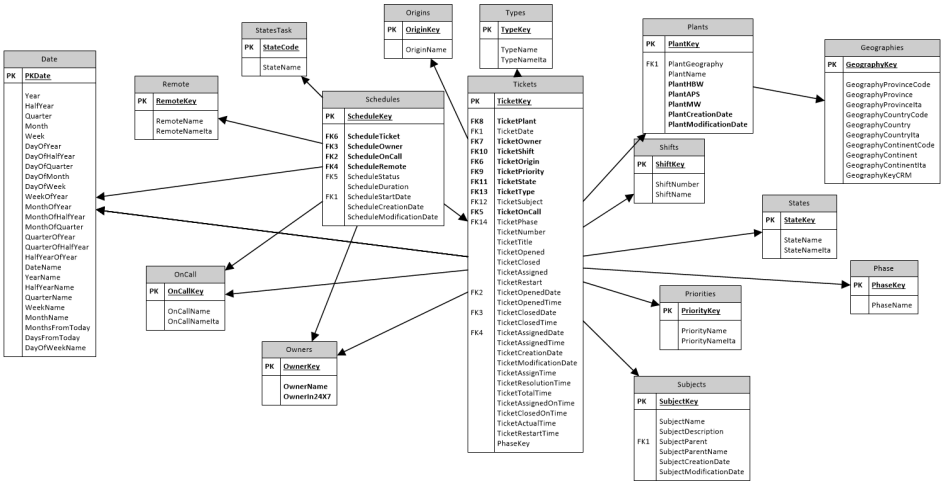


Figura 4.1: Modello del progetto

Le **misure** che si dovranno creare si baseranno su **Ticket** e **Impegni**, queste saranno al centro del modello, i requisiti richiedono di associare dimensioni temporali sia ai Ticket e agli Impegni, per questo motivo la scelta del modello è ricaduta sulla **fact constellation** sfruttando un'**unica** tabella temporale a cui si assoceranno, il ticket di un proprietario inoltre può contenere al suo interno impegni di **differenti operatori**.

Scelta la fact constellation come modello si può procedere con la progettazione dello schema delle tabelle, il risultato è visibile in figura 4.1. Possiamo quindi vedere come si è deciso di mantenere le tabelle *Tickets* e *Schedules* al **centro** dello schema e le tabelle delle dimensioni sono relazionate alle due centrali, si notino le tabelle: *Owners*, *OnCall* e *Date* che hanno una relazione sia con Tickets che con Schedules e le tabelle *Plants* e *Geographies* che sono legate tra loro relazionandosi a Tickets tramite Plants. Le altre tabelle sono connesse solamente ad una delle due tabelle.

Alcune tra le dimensioni non sono state inserite partendo dai requisiti (come Geographies) ma sono state create in quanto potrebbero risultare **utili** per future estrazioni, perciò si è deciso di includerle nelle fasi iniziali del progetto. Altre dimensioni sono state create per andare incontro ad alcune richieste di estrazione **ad-hoc**, la dimensione *Phase*, ad esempio, non verrà utilizzata all'interno dei report ma è stata utilizzata per soddisfare la richiesta di un'estrazione **occasionale** richiesta dal management.

Una volta creato il database su cui si appoggerà il cubo multidimensionale si

può passare alla progettazione del cubo vero e proprio modellando dimensioni e misure a partire dalle tabelle appena create.

Come si è già detto, per ogni tabella non centrale esisterà una dimensione (eccezione fatta per Plant e Geographies), per quanto riguarda le misure, invece ci sarà bisogno di dichiararne alcune native ed altre **calcolate** per poter andare incontro ai requisiti dei report.

L'utilizzo delle misure calcolate nasce dal fatto che l'operazione di media nativa di SSAS è utilizzabile solamente con dimensioni **temporali**, i requisiti chiedono medie anche su altri tipi di dimensioni bisognerà perciò calcolare la media sfruttando le misure di somma e conteggio.

Per quanto riguarda le dimensioni c'è un'ultima precisazione da fare: dimensioni come Origin, Subjects, Types ed altre non hanno una relazione diretta con la tabella delle misure Schedules, è molto importante quindi tracciare all'interno del database la relazione tra Schedules e Tickets, in modo da permettere l'utilizzo di queste dimensioni all'interno del database.

Come specificato nel paragrafo 2.1.6 è molto importante costruire un sistema di **permessi** per evitare che informazioni sensibili siano alla portata sia di chi si trova all'esterno dell'azienda ma anche per chi si trova al suo interno, per questo tipo di sicurezza SSAS implementa i **ruoli**, a cui vengono associati degli utenti del dominio Windows che godono di privilegi in **lettura** o in **elaborazione** delle proprietà del cubo come dimensioni e misure, si è deciso di creare tre ruoli differenti: **Admin**, **Reader** e **ProcessRole**.

Il ruolo **Admin** ha accesso a qualsiasi aspetto del cubo multidimensionale senza alcun limite in quanto persona competente nel merito dello sviluppo del progetto, il ruolo Reader ha, come si può facilmente intendere, permessi di lettura, si è deciso di non differenziare questo tipo di permessi sulle varie dimensioni in quanto non necessario dal momento che il cubo si limita già di un preciso aspetto delle attività dell'azienda e non viene utilizzato per dipartimenti differenti (come potrebbe essere quello finanziario o delle vendite), infine è stato creato il ruolo **ProcessRole** per dare la possibilità a SQLAgent di gestire anche l'**aggiornamento** dei dati in maniera automatica dopo averli importati dal datasource.

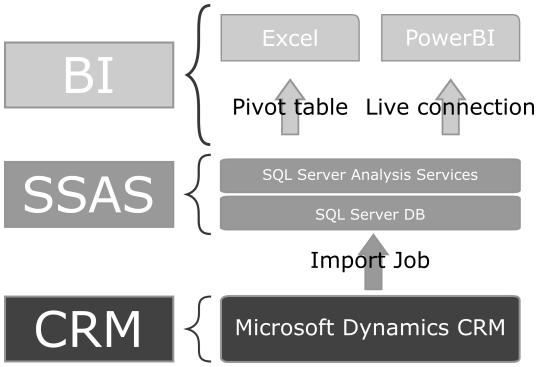


Figura 4.2: Schema della struttura del progetto

Il server su cui si trova il cubo multidimensionale non deve essere necessariamente quello su cui si trova l'origine dei dati, ovviamente deve avere la possibilità di comunicare con quest'ultimo, altrimenti non sarebbe possibile processare le dimensioni e le misure. In questo caso si è deciso di rilasciare il cubo sullo **stesso** server, questo perchè il numero di interrogazioni e le dimensioni del cubo non sono tali da giustificare l'utilizzo di un'altra macchina dedicata a questo tipo di operazioni.

La progettazione del warehouse può dirsi conclusa, si passa quindi alla progettazione del sistema di distribuzione dei report creati a partire dal cubo multidimensionale.

4.3 Report

La produzione dei report avviene tramite **due** differenti sistemi: il primo con **PowerBI** che gestisce in maniera autonoma la creazione e la distribuzione fornendo un'intefaccia drag and drop e un sistema di distribuzione dei report cloud o tramite file, il secondo invece utilizzando **Excel** richiede una progettazione di diversa natura, dalla creazione alla distribuzione.

Una volta conclusa la progettazione lo schema finale sarà quello in figura 4.2

4.3.1 Power BI

Il report di Power BI deve implementare semplicemente i requisiti richiesti, utilizzando la **connessione live** al cubo, per facilitare la creazione del report e mantenere i dati sempre aggiornati, inoltre le viste vengono collegate tra di loro per fornire dei filtri grafici di facile comprensione.

Dal momento in cui non è stata acquistata una licenza che permette la pubblicazione in cloud del report si è deciso di distribuire il **file** all'interno dell'azienda, in questo modo chi ha l'applicazione Power BI Desktop potrà visualizzare i report.

4.3.2 Excel

Il report Excel utilizza **Power Query** per poter estrarre i dati dal warehouse tramite le query MDX, portando il tutto in formato **tabulare** a cui si appoggeranno i grafici per la visualizzazione dei dati.

I grafici dei tempi di risoluzione, assegnazione e riavvio richiedono la creazione di una **combo box** per selezionare la priorità, in modo da avere una visione migliore dei dati ed evitare grafici troppo confusionari, per la selezione dei grafici sarà necessario creare una **macro** che, a seconda della priorità selezionata farà visualizzare il grafico corretto.

Inoltre è necessario pensare al fatto che questo report potrebbe essere visualizzato all'**esterno** della rete aziendale, una volta inserite all'interno del foglio di lavoro, le tabelle collegate alle query di Power Query possono essere **scollegate**, in modo da mantenere solamente i dati off-line, si è quindi deciso di creare una macro da chiamare in fase di **invio** dell'allegato aggiornando i dati e slegandoli dalle proprie query, per evitare anche messaggi di errore si dovranno anche eliminare query e connessioni al cubo.

Il report con la connessione attiva verrà quindi utilizzato come template per la creazione dell'effettivo allegato da inviare, che sarà uno snapshot di quel determinato momento.

Per l'invio della mail si utilizza un'applicazione Python chiamata *ReportSender* che si occuperà di **inviare** la mail. Tutte le proprietà del report verranno configurate sul database per permettere la **personalizzazione** tramite un'altra applicazione web chiamata *ReportConfigurator*.

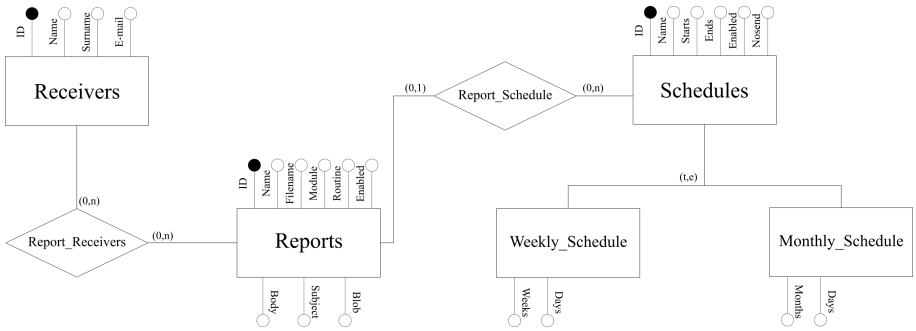


Figura 4.3: Diagramma ER del database per la gestione dei report

Una volta eseguito, ReportSender procederà a collegarsi al database tramite un file .ini, scaricherà il template, eseguirà una macro (se presente) definita all'interno del database e procederà con l'invio dello snapshot salvato, utilizzando i parametri di personalizzazione che ha preso sempre dal database.

Per comprendere meglio la progettazione di ReportSender e ReportConfigurator si tratterà del database su cui si basano.

Il database è stato costruito tramite il Model di **Django**, il cui schema è visibile in figura 4.3, è presente una tabella centrale con i report a cui sono associati i destinatari tramite una relazione **molti a molti**. I template dei report sono salvati direttamente all'interno del database tramite un **blob** e per le schedulazioni si è impostato un inizio di gerarchia, in modo da poter implementare tipi di schedulazione differenti da quello mensile.

Il database è sulla stessa istanza in cui è presente il data warehouse, in modo da evitare l'utilizzo di un ulteriore server per alcune semplici interrogazioni periodiche.

Per poter inviare le mail è stato fornito dal reparto IT l'accesso al **mail server aziendale**, tramite questo accesso si potranno inviare le mail a tutti i destinatari automaticamente.

La struttura dell'applicazione permette di avere un buon grado di configurabilità, sia per quanto riguarda la personalizzazione del messaggio che la gestione del report Excel vero e proprio. Può essere utilizzato inoltre per una pluralità di report allo stesso momento. s L'applicazione è costituita da diversi componenti:

- **Receiver**: astrazione di un destinatario, rappresentato da: id, nome, cognome, e-mail

- **Report**: si occupa di immagazzinare tutti i dati rilevanti di un report, compresa una lista dei destinatari

- **DatabaseModule**: modulo che si occupa di interfacciarsi con il database per conto degli altri moduli, fornisce metodi per la raccolta dei report e dei destinatari

- **ExcelSnapshot**: modulo che crea uno snapshot a partire da un file Excel, esegue una macro se necessario

- **ReportSender**: incapsula un report all'interno di una mail, con relativo oggetto, corpo, allegato e destinatari.

- **Main**: si occupa di controllare le funzionalità come l'invio del report o servizi di debug come la lista dei report o la creazione dello snapshot senza invio.

Il tutto è stato incluso in un file eseguibile di Windows utilizzando il tool *pyinstaller* per facilitare l'avvio dell'applicazione. In questo modo invocando ReportSender.exe si potrà inviare il report specificato da riga di comando.

Per implementare l'**invio automatico** delle mail si utilizza il tool Task Scheduler di Windows che permette l'esecuzione di task periodici, si potrà quindi configurare l'esecuzione automatica di ReportSender con il **report id** impostando come *trigger* dell'esecuzione il primo giorno di ogni mese ad un determinato orario. La motivazione principale per cui non si utilizza SQLAgent anche per l'invio dei report via mail è dato dal fatto che SQLAgent, viene eseguito da un utente dedicato, l'invio dello snapshot richiede però l'esecuzione tramite Pywin32 di un processo Excel, l'utilizzo di questo processo non può essere effettuata da questo tipo di utente senza una configurazione particolare dell'utenza, si è preferito quindi l'utilizzo di Task Scheduler che è in grado di avviare un processo impersonando un utente specificato, bypassando così le limitazioni.

Per quanto riguarda la configurazione il tutto viene effettuato tramite un servizio che utilizza Django, chiamato ReportConfigurator, che fornisce un'interfaccia web che permette di gestire i report in ogni suo aspetto, struttura della mail, destinatari e schedulazione.

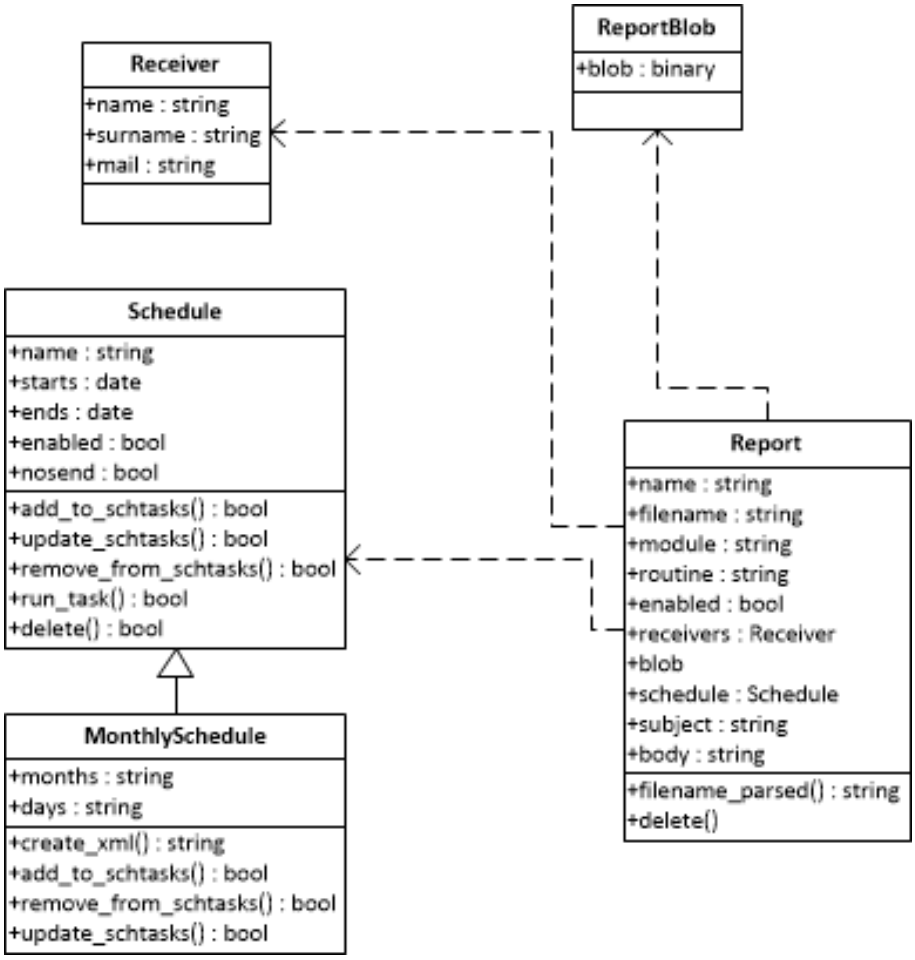


Figura 4.4: Diagramma UML della struttura di Report Config

L'applicazione si basa, come già detto sul database utilizzato da ReportSender, in questo modo sarà possibile applicare le modifiche sfruttando il model di Django.

Tramite questa interfaccia sarà possibile creare nuovi report e nuovi destinatari ma anche configurare: il **nome** del report, la **macro** da eseguire per creare lo snapshot, il nome del **file** da allegare, **oggetto** e **corpo** della mail, l'elenco dei **destinatari**, abilitare e disabilitare i report e infine la **schedulazione** dell'invio.

L'interfacciamento con Task Scheduler di Windows avviene creando un **file XML** formattato in modo da poter essere importato dal tool, questo XML conterrà tutte le informazioni necessarie per creare un **nuovo** task con la schedulazione richiesta. L'import verrà effettuato eseguendo tramite riga di comando

Task Scheduler (schtasks.exe) fornendogli il file XML appena prodotto, in questo modo sarà possibile, tenendo traccia anche del nome dato al task schedulato **modificare** ed **eliminare** il task, gestendone le sue proprietà.

Il **webserver** utilizzato sarà quello messo a disposizione da Django e verrà eseguito sulla stessa macchina utilizzata per l'origine dei dati ed il cubo sempre per motivi legati al non utilizzo costante del server. Questa macchina inoltre viene riavviata periodicamente ogni weekend, quindi per mantenere sempre online il webserver verrà creato un **job** in Task Scheduler che lo eseguirà una volta avviata la macchina. L'utilizzo del webserver di Django è stato preferito ad un webserver IIS per la sua **semplicità** di creazione in quanto i tempi per lo studio della pubblicazione del servizio su IIS non erano compatibili con quelli dello sviluppo del progetto.

Capitolo 5

Implementazione

In questo capitolo si tratterà dell'effettiva creazione del progetto, inserendo e commentando alcuni frammenti importanti di codice, in modo da comprendere nel dettaglio la costruzione del progetto.

5.1 Import

L'import, costruito utilizzando SQLAgent e alcune stored procedure, si occuperà della **selezione** dei nuovi dati da memorizzare e della successiva **normalizzazione**.

Per scegliere solamente i nuovi record e quelli modificati successivamente all'ultima fase di import si è deciso di utilizzare i campi di **creazione** e ultima **modifica**, si aggiunge la possibilità di selezionare tutti i dati tramite un **flag**, questa opzione viene utilizzata se necessario effettuare operazioni di debug, normalmente è disabilitata.

Una volta selezionati, i dati vengono quindi immagazzinati in una **tabella temporanea** in modo da poterli unire alla tabella esistente tramite il comando *merge*. Merge permette di sincronizzare due tabelle specificando le operazioni di insert, update e delete su una in base alle **differenze** trovate sull'altra. Questo tipo di approccio rende possibile specificare chiaramente le relazioni tra le due tabelle con una sintassi più semplice e concisa rispetto ad un insieme di update e insert, soluzione inizialmente adottata ma più bug-prone e di difficile comprensione nella logica di funzionamento.

Le stored procedure condividono la stessa struttura, prendiamo la sp_import_tickets come esempio, essendo la più importante e completa.

```
1  create procedure [dbo].[sp_import_tickets]
2    @forceUpdate as bit = 0
3  as
4  begin
5    set nocount on;
6
7    declare @lastcreated datetime = coalesce((select max(TicketCreationDate) from
         Tickets), '19000101')
8    declare @lastmodified datetime
9    if @forceUpdate = 1
10     set @lastmodified = '19000101'
11   else
12     set @lastmodified = coalesce((select max(TicketModificationDate) from
           Tickets), '19000101')
13
14   if object_id(N'tempdb..#ticketstemp', N'U') is not null
15     drop table #ticketstemp
```

In questa prima sezione della stored procedure si va a scrivere l'intestazione e ad impostare i parametri @lastcreated e @lastmodified che diranno quali sono gli ultimi record da escludere in modo tale da poter prendere solamente i più recenti, inoltre alla fine si controlla che la tabella temporanea #ticketstemp sia stata effettivamente eliminata per evitare errori durante l'esecuzione della stored procedure.

```
16 select
17   IncidentId,
18   CustomerId,
19   dbo.fn_get_date_key(dateadd(hour, 2, new_dataaperturacaso)) as DateKey,
20   OwnerId,
21   timefromparts(datepart(hour,
22         dateadd(hour, 2, new_dataaperturacaso)),
23   0, 0, 0, 0) as ShiftKey,
24   case when new_sw = 1 then 'SW' else 'HW' end as OriginKey,
25   PriorityCode,
26   StateCode,
27   case when isnull(New_NumeroCommessa, '') like 'TL0%07' or isnull(
         New_NumeroCommessa, '') like 'TL0%04' then -2
28   when CaseTypeCode in (100000000, 100000001) then -1
29   else coalesce(CaseTypeCode, -1) end as TypeKey,
30   isnull(SubjectId, 'FE93C118-E926-47C5-9F10-EEC57C7A0307') as SubjectId,
31   case when datepart(weekday, dateadd(hour, 2, new_dataaperturacaso)) in (1, 7)
32     or datepart(hour, dateadd(hour, 2, new_dataaperturacaso)) >= 17
33     or datepart(hour, dateadd(hour, 2, new_dataaperturacaso) < 8
34   then 1
35   else 0 end as OnCall,
36   case when new_datachiusuracaso is not null or StateCode = 1 then 3
37      when apx_DataRiavvioImpianto is not null then 2
38      when new_datapresaincarico is not null then 1
39      when new_dataaperturacaso is not null then 0
```

```
40    end as Phase,
41    TicketNumber,
42    Title,
43    dateadd(hour, 2, new_dataaperturacaso) as new_dataaperturacaso,
44    dateadd(hour, 2, new_datachiusuracaso) as new_datachisuuracaso,
45    dateadd(hour, 2, new_datapresaincarico) as new_datapresaincarico,
46    dateadd(hour, 2, apx_DataRiavvioImpianto) as apx_DataRiavvioImpianto,
47    dbo.fn_get_date_key(dateadd(hour, 2, new_dataaperturacaso)) as OpenedDate,
48    cast(dateadd(hour, 2, new_dataaperturacaso) as time) as OpenedTime,
49    dbo.fn_get_date_key(dateadd(hour, 2, new_datachiusuracaso)) as ClosedDate,
50    cast(dateadd(hour, 2, new_datachiusuracaso) as time) as ClosedTime,
51    dbo.fn_get_date_key(dateadd(hour, 2, new_datapresaincarico)) as AssignedDate,
52    cast(dateadd(hour, 2, new_datapresaincarico) as time) as AssignedTime,
53    createdon,
54    modifiedon
55  into #ticketstemp
56  from [CRMDB-P-FRN001.SLOG.LOCAL].[CrmCustomerCare_MSCRM].[dbo].[IncidentBase]
57  where createdon > @lastcreated or modifiedon > @lastmodified
```

In questa parte si vanno a selezionare i dati la cui data di creazione è successiva a @lastcreated o quella di modifica successiva a @lastmodified dalla tabella IncidentBase che contiene i Ticket, nel database di Dynamics. Qui avviene anche la **normalizzazione** dei dati, ad esempio alle date vengono aggiunte due ore per conformarle con il fuso orario italiano, il turno del ticket viene estratto prendendo solo l'orario di apertura del ticket e così via.

Dopo aver inserito i dati nella tabella #ticketstemp arriva il momento di scriverli sulla tabella Tickets del database.

```
53  merge Tickets as Target
54  using #ticketstemp as Source
55  on Target.TicketKey = Source.IncidentId
56  when matched and (Source.modifiedon > Target.TicketModificationDate
57                    or @forceUpdate = 1) then
58    update set
59      Target.TicketPlant = Source.CustomerId,
60      Target.TicketDate = Source.DateKey,
61      Target.TicketOwner = Source.OwnerId,
62      Target.TicketShift = Source.ShiftKey,
63      Target.TicketOrigin = Source.OriginKey,
64      Target.TicketPriority = Source.PriorityCode,
65      Target.TicketState = Source.StateCode,
66      Target.TicketType = Source.TypeKey,
67      Target.TicketSubject = Source.SubjectId,
68      Target.TicketOnCall = Source.OnCall,
69      Target.TicketPhase = Source.Phase,
70      Target.TicketNumber = Source.TicketNumber,
71      Target.TicketTitle = Source.Title,
72      Target.TicketOpened = Source.new_dataaperturacaso,
73      Target.TicketClosed = Source.new_datachiusuracaso,
```

```
74      Target.TicketAssigned = Source.new_datapresaincarico,
75      Target.TicketRestart = Source.apx_DataRiavvioImpianto,
76      Target.TicketOpenedDate = Source.OpenedDate,
77      Target.TicketOpenedTime = Source.OpenedTime,
78      Target.TicketClosedDate = Source.ClosedDate,
79      Target.TicketClosedTime = Source.ClosedTime,
80      Target.TicketAssignedDate = Source.AssignedDate,
81      Target.TicketAssignedTime = Source.AssignedTime,
82      Target.TicketCreationDate = Source.createdon,
83      Target.TicketModificationDate = Source.modifiedon
84  when not matched by Target then
85    insert
86      (TicketKey, TicketPlant, TicketDate, TicketOwner,
87      TicketShift, TicketOrigin, TicketPriority, TicketState,
88      TicketType, TicketSubject, TicketOnCall, TicketPhase,
89      TicketNumber, TicketTitle, TicketOpened,
90      TicketClosed, TicketAssigned, TicketRestart,
91      TicketOpenedDate, TicketOpenedTime, TicketClosedDate,
92      TicketClosedTime, TicketAssignedDate, TicketAssignedTime,
93      TicketCreationDate, TicketModificationDate)
94    values
95      (Source.IncidentId, Source.CustomerId, Source.DateKey, Source.OwnerId,
96      Source.ShiftKey, Source.OriginKey, Source.PriorityCode, Source.StateCode,
97      Source.TypeKey, Source.SubjectId, Source.OnCall, Source.Phase,
98      Source.TicketNumber, Source.Title, Source.new_dataaperturacaso,
99      Source.new_datachiusuracaso, Source.new_datapresaincarico,
100     Source.apx_DataRiavvioImpianto, Source.OpenedDate, Source.OpenedTime,
101     Source.ClosedDate, Source.ClosedTime, Source.AssignedDate,
102     Source.AssignedTime, Source.createdon, Source.modifiedon);
103
104 drop table #ticketstemp
```

Tramite questo comando si vanno ad aggiornare i campi per cui si possiede già un record e ad inserire i record che invece devono ancora essere importati, una volta fatto questo ci si assicura di eliminare la tabella temporanea in quanto non sarà più utile.

Questo è lo schema che è stato utilizzato per tutte le altre stored procedure di import.

5.2 Warehouse

Nella figura 4.2 il warehouse è il livello intermedio del progetto, nonchè il più importante, dove si va a costruire dapprima la struttura relazionale che sono le fondamenta per il cubo multidimensionale.

5.2.1 Modello

Di seguito verrà fornito un elenco dei campi delle tabelle mostrate nella figura 4.1.

5.2.2 Tickets

Colonna	**Tipo**	**Note**
TicketKey	uniqueidentifier	Chiave primaria, importata
TicketPlant	uniqueidentifier	Foreign key dell'impianto, importata
TicketDate	datetime	Foreign key della data, calcolata dalla data di apertura
TicketOwner	uniqueidentifier	Foreign key dell'operatore, importata
TicketShift	time(7)	Foreign key del turno, calcolata dalla data di apertura
TicketOrigin	char(2)	Foreign key dell'origine, importata
TicketPriority	int	Foreign key della priorità, importata
TicketState	tinyint	Foreign key dello stato del ticket, importata
TicketType	int	ForeignKey del tipo del ticket, importata
TicketSubject	uniqueidentifier	Foreign key della categoria, importata
TicketOnCall	bit	Foreign key della reperibilità, calcolata in base alle fasce al paragrafo 1.2

TicketPhase	int	Fase del ticket, calcolata in base allo stato del ticket
TicketNumber	nvarchar(100)	Chiave parlante del ticket, importata
TicketTitle	nvarchar(200)	Titolo del ticket
TicketOpened	datetime	Data di apertura del ticket, importata
TicketClosed	datetime	Data di chiusura del ticket, importata
TicketAssigned	datetime	Data di assegnazione del ticket, importata
TicketRestart	datetime	Data di riavvio impianto, importata
TicketOpenedDate	datetime	Foreign key, calcolata da TicketOpened
TicketOpenedTime	time(7)	Al momento inutilizzato
TicketClosedDate	datetime	Foreign key, calcolata da TicketClosed
TicketClosedTime	time(7)	Al momento inutilizzato
TicketAssignedDate	datetime	Foreign key, calcolata da TicketAssigned
TicketAssignedTime	time(7)	Al momento inutilizzato
TicketCreationDate	datetime	Data di creazione del record sul database
TicketModificationDate	datetime	Data dell'ultima modifica del record sul database

TicketAssignTime		Calcolato, tempo dall'apertura all'assegnazione del ticket
TicketResolutionTime		Calcolato, tempo dall'assegnazione alla chisura del ticket
TicketTotalTime		Calcolato, tempo dall'apertura alla chiusura del ticket
TicketAssignedOnTime		Calcolato, 0 se il ticket non è stato assegnato in tempo, 1 altrimenti
TicketClosedOnTime		Calcolato, 0 se il ticket non è stato chiuso in tempo, 1 altrimenti
TicketActualTime		Calcolato, tempo dall'apertura del ticket al momento attuale
TicketRestartTime		Calcolato, tempo dall'apertura del ticket al riavvio dell'impianto

5.2.3 Schedules

Colonna	**Tipo**	**Note**
ScheduleKey	uniqueidentifier	Chiave primaria, importata
ScheduleTicket	uniqueidentifier	Foreign key, ticket a cui è associato l'impegno
ScheduleOwner	uniqueidentifier	Foreign key, proprietario del ticket
ScheduleOnCall	bit	Foreign key, indica se il ticket è stato aperto in reperibilità
ScheduleRemote	bit	Al momento inutilizzato

ScheduleStatus	int	Foreign key, stato dell'impegno
ScheduleDuration	int	Durata in minuti dell'impegno, importato
ScheduleStartDate	datetime	Data di inizio dell'impegno, importato
ScheduleCreationDate	datetime	Data di creazione del record sul database, importata
ScheduleModificationDate	datetime	Data di ultima modifica del record sul databsae, importata

5.2.4 Dates

Questa tabella è stata generata inizialmente dal tool di Visual Studio, in seguito sono stati aggiunti dei campi calcolati, la necessità di effettuare query che fornissero un certo numero di dati nel passato ha reso necessario inserire delle misure che indicassero la profondità temporale a partire dal giorno attuale.

Colonna	**Tipo**	**Note**
PKDate	datetime	Chiave primaria, impostato a mezzanotte del giorno che rappresenta
Year	datetime	Anno
HalfYear	datetime	Semestre
Quarter	datetime	Trimestre
Month	datetime	Mese
Week	datetime	Settimana
DayOfYear	int	Giorno dell'anno
DayOfHalfYear	int	Giorno del semestre

DayOfQuarter	int	Giorno del trimestre
DayOfMonth	int	Giorno del mese
DayOfWeek	int	Giorno della settimana
WeekOfYear	int	Settimana dell'anno
MonthOfYear	int	Mese dell'anno
MonthOfHalfYear	int	Mese del semestre
MonthOfQuarter	int	Mese del trimestre
QuarterOfYear	int	Trimestre dell'anno
QuarterOfHalfYear	int	Trimestre del semestre
HalfYearOfYear	int	Semestre dell'anno
DateName		Stringa che rappresenta la data nel formato "Mese Giorno, Anno"
YearName		Stringa che rappresenta l'anno
HalfYearName		Stringa che rappresenta il semestre nel formato "S< Semestre dell'anno >, Anno"
QuarterName		Stringa che rappresenta il trimestre nel formato "T< Trimestre dell'anno >, Anno"
WeekName		Stringa che rappresenta la settimana nel formato "Settimana< Settimana dell'anno >, Anno"
MonthName		Stringa che rappresenta il mese nel formato "Mese, Anno"
MonthsFromToday		Mesi a partire dalla data di oggi, il mese attuale ha valore 0, quello passato 1 e il successivo -1

DaysFromToday		Giorni a partire dalla data di oggi, il giorno attuale ha valore 0, quello passato 1 e il successivo -1
DayOfWeekName		Stringa che rappresenta il giorno della settimana

5.2.5 Subjects

La particolarità di questa tabella è che rappresenta una gerarchia incompleta, come descritto nel paragrafo 2.1.4

Colonna	**Tipo**	**Note**
SubjectKey	uniqueidentifier	Chiave primaria, importata
SubjectName	nvarchar(500)	Nome della categoria
SubjectDescription	nvarchar(MAX)	Descrizione della categoria
SubjectParent	uniqueidentifier	Categoria padre, NULL se la categoria è la prima della gerarchia
SubjectParentName	nvarchar(500)	Nome della categoria padre
SubjectCreationDate	datetime	Data di creazione del record sul database
SubjectModificationDate	datetime	Data di ultima modifica del record sul database

5.2.6 Plants

Colonna	**Tipo**	**Note**
PlantKey	uniqueidentifier	Chiave primaria, importata

PlantGeography	uniqueidentifier	Foreign key, collocazione geografica
PlantName	nvarchar(MAX)	Nome del cliente
PlantHBW	bit	1 se il cliente possiede un High Bay Warehouse, 0 altrimenti
PlantAPS	bit	1 se il cliente possiede un Automatic Picking System, 0 altrimenti
PlantMW	bit	1 se il cliente possiede un Manual Warehouse, 0 altrimenti
PlantCreationDate	datetime	Data di creazione del record sul database, importata
PlantModificationDate	datetime	Data di ultima modifica del record sul database, importata

5.2.7 Geographies

Colonna	**Tipo**	**Note**
GeographyKey	uniqueidentifier	Chiave primaria
GeographyProvinceCode	nvarchar(3)	Provincia, utilizzata nel caso di province italiane, uguale a *GeographyCountryCode* altrimenti
GeographyProvince	nvarchar(MAX)	Nome della provincia
GeographyProvinceIta	nvarchar(MAX)	Nome della provincia, in italiano
GeographyCountryCode	nvarchar(3)	Codice ISO 3166-1 alpha-3 (a 3 caratteri) della nazione

GeographyCountry	nvarchar(MAX)	Nome della nazione
GeographyCountryIta	nvarchar(MAX)	Nome della nazione, in italiano
GeographyContinentCode	nvarchar(2)	Codice del continente, a due caratteri
GeographyContinentCode	nvarchar(MAX)	Nome del continente
GeographyContinentIta	nvarchar(MAX)	Nome del continente, in italiano
GeographyKeyCRM	int	Chiave della nazione utilizzato sul CRM, importata

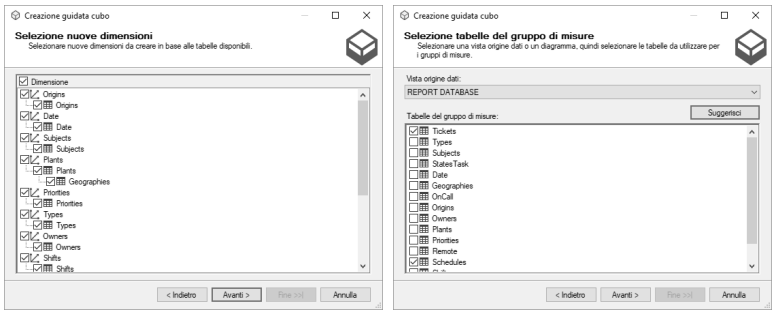
Si è deciso di importare la chiave GeographyKeyCRM senza però utilizzarla come chiave primaria in quanto il database del CRM non memorizza su una tabella dedicata i nomi delle varie nazioni ma sono contenute nella tabella *StringMap* che contiene le varie stringhe di traduzione per le lingue, per evitare di utilizzare parti di chiavi che possono essere utilizzate in altri ambiti si è deciso di creare i record con una chiave dedicata, questa tabella è stata popolata manualmente, quindi non viene aggiornata automaticamente.

5.3 Dimensioni

Una volta implementato il modello e costruito un sistema per popolarlo e mantenerlo aggiornato è possibile procedere con la costruzione effettiva del cubo multidimensionale. Questo procedimento viene effettuato utilizzando Visual Studio.

Creato un nuovo *Progetto Multidimensionale* sarà necessario collegarlo al database e, dopo aver selezionato le tabelle che da utilizzare per costruire il cubo, un **wizard** permetterà la creazione delle prime dimensioni e misure, Visual Studio ne proporrà alcune in base allo schema utilizzato ed al nome delle tabelle.

Le misure create automaticamente saranno abbastanza basilari, sarà quindi necessario completare la configurazione delle gerarchie, dei valori e delle relazioni tra campi e dei campi calcolati.



(a) Creazione dimensioni (b) Creazione misure

Figura 5.1: Suggerimento automatico di misure alla creazione di un cubo

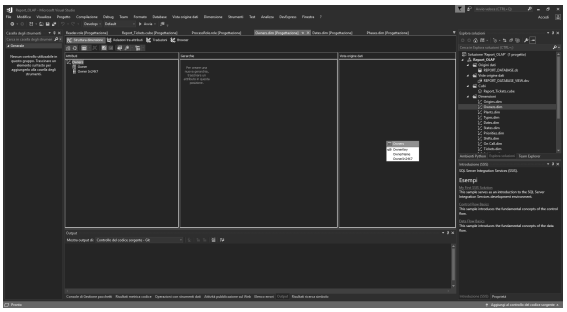


Figura 5.2: Schermata di configurazione delle dimensioni

Di seguito si vedrà più nel dettaglio la creazione di alcune dimensioni più complesse.

5.3.1 Owners

Questa è una dimensione basilare, la modifica delle dimensioni avviene nella schermata vista in figura 5.2, la chiave primaria **OwnerKey** è già stata aggiunta automaticamente ed è stato inserito solamente l'attributo OwnerIn24X7, i nomi delle dimensioni sono stati cambiati in "Owner" e "Owner In 24 X 7" per motivi di leggibilità.

Se effettuassimo una query con l'attributo *Owner* otterremmo come risposta una o più **chiavi** di cui non si riuscirebbe a capire il significato, per poter vedere il nome del proprietario del ticket al posto della chiave si deve modificare la proprietà "NameColumn" dell'attributo, facendo questo si potrà specificare la

colonna che contiene i risultati da restituire, in questo caso si utilizzerà la colonna *OwnerName* per mostrare i risultati dell'attributo *Owner*.

È utile impostare l'**ordinamento** con cui i valori degli attributi verranno restituiti, di default SSAS sceglie come ordinamento la **chiave** dell'attributo, in questo caso non sarebbe comodo perchè verrebbero visualizzati gli operatori in un ordine apparentemente sparso, tramite la proprietà "OrderBy" si potrà scegliere se effettuare l'ordinamento sui valori di chiave o di **nome**, scegliendo il nome i risultati verranno restituiti in ordine alfabetico.

5.3.2 Dates

A tutti gli attributi sono state associate le NameColumn appropriate, in modo da restituire un valore più leggibile e rappresentativo dell'attributo che si sta visualizzando.

Qui è stata introdotta anche una **gerarchia** chiamata "Calendar Date" con questa struttura: Year → Half Year → Quarter → Month → Date.

Una volta definita una gerarchia è importante definire le **relazioni** che ci sono tra gli attributi della gerarchia, quando le gerarchie sono ampie e/o complesse si corre il rischio di impattare pesantemente sulle **performance** delle query, SSAS permette di specificare le relazioni nella scheda "Relazioni tra attributi".

5.3.3 Plants

In questo caso si tratta una dimensione che sfrutta lo schema a **costellazione** che è stato scelto per il warehouse, lo stesso varrebbe anche per lo schema a fiocco di neve dal momento in cui la dimensione si estende su più tabelle.

SSAS rileva quali sono le relazioni tra le due tabelle e permette di gestire la dimensione in maniera non dissmile da quanto non si sia già fatto con le dimensioni precedenti.

Anche in questo caso si va a creare una **gerarchia** chiamata "Plant Geography" con la seguente struttura: Continent → Country → Province → Plant.

Fatto questo si aggiustano anche le relazioni tra attributi e le colonne dei nomi, per i motivi visti nel paragrafo precedente.

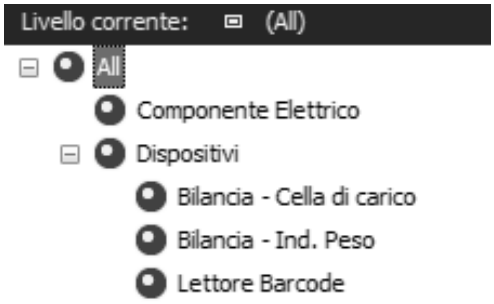


Figura 5.3: Esempio di gerarchia incompleta

5.3.4 Subjects

Su questa dimensione, come accennato precedentemente, si ha una **gerarchia incompleta**, questo impedisce di dichiarare la gerarchia come fatto in precedenza ma bisognerà eseguire alcuni passaggi differenti.

Si aggiunge prima di tutto l'attributo *SubjectParent* alla dimensione. Fatto questo si impostano le seguenti proprietà di SubjectParent ai valori:

- **Usage** = Parent
- **RootMemberIf** = ParentIsSelf
- **MembersWithData** = NonLeafDataHidden
- **NamingTemplate** = Nomenclatura per ogni livello della gerarchia

Usage permette di specificare quale **tipo di utilizzo** si farà dell'attributo, se impostato a "Key" specifica che ogni valore dell'attributo è un nodo foglia, se invece viene impostata a "Parent" SSAS utilizzerà i valori come se fossero padri di altri nodi dell'albero.

RootMemberIf fornisce indicazioni a SSAS su come capire quali valori sono le **radici** dell'albero, nel nostro caso le radici dell'albero contengono il proprio valore di chiave nell'attributo *SubjectParent* e quindi si utilizza la proprietà "ParentIsSelf", esistono anche proprietà che permettono di specificare il valore radice utilizzando valori NULL o vuoti.

Le ultime restanti prorietà vanno a definire il **comportamento** dei livelli, se vogliamo mostrare o meno un parent nel livello inferiore come membro visibile della gerarchia come nel caso di "MembersWithData" o di quali nomi dare ai livelli della gerarchia in "NamingTemplate"

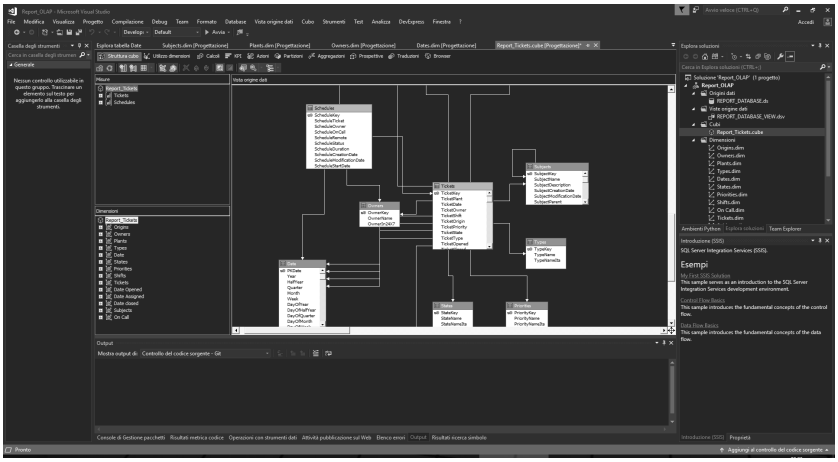


Figura 5.4: Finestra di progettazione del cubo

Il risultato di questa configurazione si può vedere in figura 5.3 dove il membro padre "Dispositivi" non è presente nel livello inferiore.

5.4 Misure

Le misure possono essere gestite dalla finestra di progettazione del cubo in figura 5.4, a sinistra sarà possibile visualizzare le misure già create automaticamente dal wizard e crearne di nuove.

Normalmente ci si troverà con le misure standard, cioè il conteggio di ticket, durata e numero di impegni.

Per poter visualizzare i dati all'interno dei report si dovranno creare ulteriori misure, native e calcolate.

Di seguito saranno elencate le misure native.

Gruppo	**Misura**	**Tipo**
Tickets		
	Ticket Assign Time	Sum
	Ticket Resolution Time	Sum
	Ticket Total Time	Sum
	Tickets Count	Count

	Ticket Assign Time	Count non-null
	Ticket Resolution Time Count	Count non-null
	Ticket Total Time Count	Count non-null
	Ticket On Call	Sum
	Ticket Restart Time	Sum
	Ticket Restart Time count	Count non-null
	Ticket Assigned On Time	Sum
	Ticket Closed On Time	Sum
Schedules		
	Schedule Duration	Sum
	Schedules Count	Count

Le misure in corsivo non sono **visibili**, cioè non sarà possibile utilizzarle all'interno di query ma possono essere utilizzate per creare **misure calcolate**, in quanto utili per calcoli intermedi, come si vedrà tra poco.

Di seguito verranno elencate le misure con la relativa formula.

Misura	**Formula**
Tickets to date	**SUM**(PeriodsToDate([**Date**].[(**All**)]), [Measures].[Tickets **Count**])
Average Assignation Time	[Measures].[Ticket Assign **Time**]/[Measures].[Ticket Assign **Time Count**]
Average Resolution Time	[Measures].[Ticket Resolution **Time**]/[Measures].[Ticket Resolution **Time Count**]
Average Restart Time	[Measures].[Ticket Restart **Time**]/[Measures].[Ticket Restart **Time Count**]
Average Total Time	[Measures].[Ticket Total **Time**]/[Measures].[Ticket Total **Time Count**]

Average Total Time (days)	[Measures].[Average Total **Time**]/1440
% Tickets Assigned On Time	[Measures].[Ticket Assigned **On Time**] / [Measures].[Tickets **Count**]
% Tickets Closed On Time	[Measures].[Ticket Closed **On Time**] / [Measures].[Tickets **Count**]
Hours consumed	[Measures].[Schedule Duration] / 60

5.5 Ruoli

I ruoli si trovano scorrendo la finestra *Esplora soluzioni*, all'interno della cartella *Ruoli*. In figura 5.5 è possibile vedere la schermata di modifica di un ruolo. I ruoli da creare sono tre: Reader, Admin e ProcessRole.

5.5.1 Reader

Per quanto riguarda questo ruolo bisognerà abilitare "Lettura definizione" in modo da permettere la lettura dei metadati ai vari utenti, in seguito si aggiungeranno gli utenti che avranno la possibilità di interrogare il cubo che nel nostro caso saranno i lettori del report in Power BI.

5.5.2 Administrator

Questo ruolo avrà il controllo completo su tutti gli oggetti all'interno del database, pertanto si dovrà spuntare l'opzione "Controllo completo" e anche qui aggiungere tutti gli utenti a cui si vogliono dare privilegi di amministratore.

5.5.3 ProcessRole

Qui si dovrà spuntare l'opzione "Elaborazione database" per permettere all'utente di SQLAgent di elaborare automaticamente tutti i dati all'interno del cubo. Per ottenere il nome dell'utente sotto cui viene eseguito SQLAgent è possibile effettuare una select sull'oggetto sys.dm_server_services, questa tabella contiene le informazioni sullo stato di tutti i servizi di SQL Server, una volta individuata la riga riguardante SQLAgent l'utente si troverà nella colonna *service_account*.

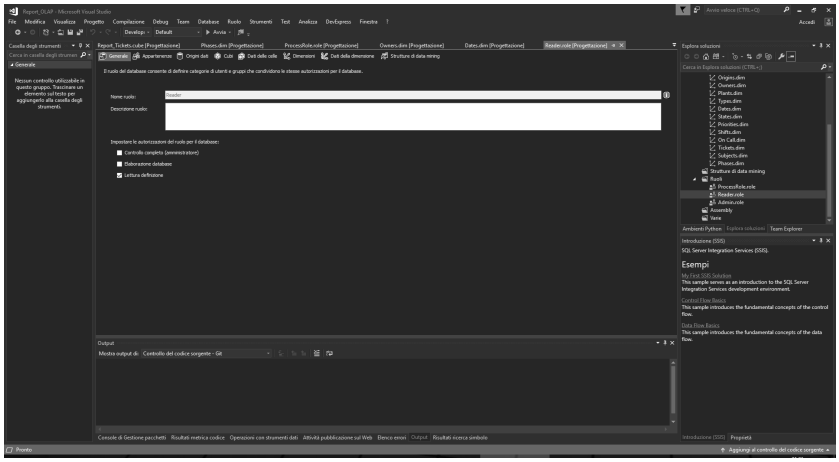


Figura 5.5: Finestra di modifica dei ruoli

5.6 Distribuzione

Una volta create tutte le componenti necessarie all'utilizzo del cubo arriva il momento di **distribuire** il progetto su un server per poterlo utilizzare.

Si può indicare su quale server distribuire il cubo all'interno delle proprietà del progetto di Visual Studio, la configurazione utilizzata è quella in figura 5.6, una volta specificata questa configurazione Visual Studio si occuperà della distribuzione del progetto ogni volta che verrà elaborato il cubo o su richiesta, le modifiche al cubo non saranno effettive finchè una di queste operazioni non verrà effettuata.

5.7 Job di Import ed Elaborazione

Di seguito parleremo dell'effettiva implementazione del job di import ed elaborazione, in modo da avere un quadro completo di come evolve il data warehouse nel tempo.

Il job è composto da 8 passaggi eseguiti nell'ordine:

1. **Import Plants**: import degli impianti

2. **Import Owners**: import degli operatori

3. **Import Subjects**: import delle categorie

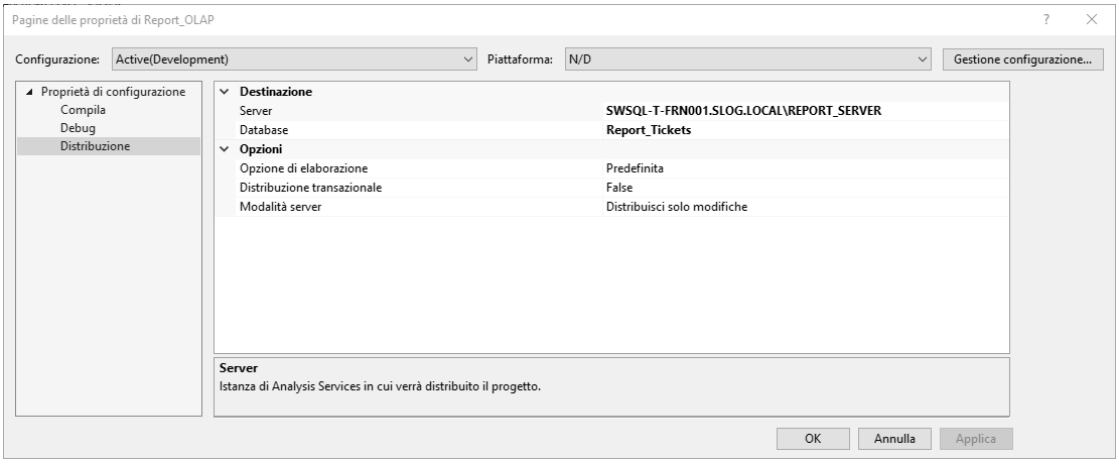


Figura 5.6: Finestra delle proprietà del progetto

4. **Import Tickets**: import dei ticket
5. **Import Schedules**: import degli impegni
6. **Process Dimensions**: elaborazione delle dimensioni
7. **Process Cube**: elaborazione delle misure

I passaggi vengono eseguiti in maniera **sequenziale**, ogni passaggio, in caso di fallimento, viene eseguito altre tre volte a distanza di 3 minuti tra un tentativo e l'altro.

La struttura dei passaggi da 1 a 5 è stata vista precedentemente, i restanti due passaggi eseguono codice **XMLA** (XML for Analysis), in questo modo possiamo eseguire le operazioni di elaborazione dei dati all'interno del cubo.

Il passo 6 ha la seguente struttura:

```
1  <Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
2    <Parallel>
3      <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
4        <Object>
5          <DatabaseID>Report_Tickets</DatabaseID>
6          <DimensionID>Date</DimensionID>
7        </Object>
8        <Type>ProcessUpdate</Type>
9        <WriteBackTableCreation>UseExisting</WriteBackTableCreation>
10     </Process>
11     <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
12       <Object>
13         <DatabaseID>Report_Tickets</DatabaseID>
```

```
14          <DimensionID>Origins</DimensionID>
15        </Object>
16        <Type>ProcessUpdate</Type>
17        <WriteBackTableCreation>UseExisting</WriteBackTableCreation>
18      </Process>
19      ...
20    </Parallel>
21  </Batch>
```

Abbiamo definito un batch di processi eseguiti in **parallelo**, ogni processo aggiornerà l'oggetto che gli viene comunicato, verrà definito un processo per ogni dimensione presente nel cubo.

Per il passo 7 la struttura è la seguente:

```
1  <Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
2    <Parallel>
3      <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
4        <Object>
5          <DatabaseID>Report_Tickets</DatabaseID>
6          <CubeID>Report_Tickets</CubeID>
7        </Object>
8        <Type>ProcessFull</Type>
9        <WriteBackTableCreation>UseExisting</WriteBackTableCreation>
10     </Process>
11   </Parallel>
12 </Batch>
```

In questo caso il processo è unico e invece di definire una *DimensionID* l'oggetto conterrà un *CubeID*.

5.8 MDX

Come già descritto nel paragrafo 2.2.3 MDX permette di interrogare il cubo con delle query la cui struttura assomiglia a quella di SQL ma con alcune differenze per quanto riguarda la selezione di righe e colonne.

Con MDX si va a definire una **proiezione** tabulare di quello che è il cubo multidimensionale, di seguito si mostrano alcune query di esempio per introdurre le potenzialità del linguaggio.

Una volta comprese le basi di questo linguaggio si proseguirà con l'implementazione dei report.

5.8.1 Query 1

La prima query è abbastanza basilare:

	Tickets Count
January, 2009	28
February, 2009	93
March, 2009	93
April, 2009	86
May, 2009	87
June, 2009	90
July, 2009	105
August, 2009	53
September, 2009	81
October, 2009	113
November, 2009	116
December, 2009	93

(a) Mesi sulle righe

	January, 2009	February, 2009	March, 2009	April, 2009
Tickets Count	28	93	93	86

(b) Mesi sulle colonne

Figura 5.7: Risultati della query 1

```
1 select
2   [Date Opened].[Calendar Date].Month on rows,
3   [Measures].[Tickets Count] on columns
4 from
5   [Report_Tickets]
```

Si chiede di vedere il numero di ticket aperti per ogni mese, specificando che i **mesi** devono essere sulle **righe** mentre il numero di **ticket** sulle **colonne**, si ottiene quindi la tabella in figura 5.7(a), sarebbe anche possibile porre i mesi sulle colonne, ottenendo il risultato in figura 5.7(b), meno leggibile però.

5.8.2 Query 2

Nel caso precedente vengono restituiti tutti i mesi che sono contenuti nella dimensione, inoltre essendo che la dimensione ha membri anche nel futuro si otterrebbero **numerose** righe con valori NULL, per filtrare i membri che non contengono misure si aggiunge la clausola *non empty*.

È possibile filtrare anche insiemi di membri che si vogliono vedere nella tabella, si possono vedere questi due filtri in azione scrivendo la query

```
1 select
2   non empty { [Date].[Calendar Date].[Month].&[2019-01-01T00:00:00],
3         [Date].[Calendar Date].[Month].&[2019-02-01T00:00:00],
4         [Date].[Calendar Date].[Month].&[2019-03-01T00:00:00],
5         [Date].[Calendar Date].[Month].&[2019-04-01T00:00:00]
6   } on rows,
7   [Measures].[Tickets Count] on columns
8 from
9   [Report_Tickets]
```

	Tickets Count
January, 2019	334
February, 2019	273
March, 2019	41

Figura 5.8: Risultati della query 2

In figura 5.8 si può vedere il risultato della query, in questo caso è stato definito un insieme di mesi tramite le parentesi graffe, si noti come il mese di Aprile 2019 non sia presente nella tabella, in quanto la clausola *non empty* ha nascosto la riga dal momento che *Tickets Count* è Null.

Tramite le parentesi graffe è possibile definire anche più misure da mostrare, l'importante è che all'interno delle parentesi graffe siano presenti solamente membri della stessa dimensione, se si vogliono mostrare più dimensioni differenti è possibile utilizzare il *crossjoin* inserendo le dimensioni all'interno di parentesi tonde.

Per ridurre il codice una query equivalente sarebbe:

```
1  select
2    non empty { [Date].[Calendar Date].[Month].&[2019-01-01T00:00:00] :
3          [Date].[Calendar Date].[Month].&[2019-04-01T00:00:00]
4    } on rows,
5    [Measures].[Tickets Count] on columns
6  from
7    [Report_Tickets]
```

Tramite il carattere ":" si va a definire un range **inclusivo** dei due estremi che contiene i membri della dimensione, l'ordinamento della dimensione qui risulta di estrema importanza.

5.8.3 Query 3

```
1  with member [Measures].[Over 300 Tickets] as
2    iif([Measures].[Tickets Count] > 300, 'Yes', 'No')
3
4  select
5    [Date Opened].[Calendar Date].[Month] on rows,
6    {[Measures].[Tickets Count], [Measures].[Over 300 Tickets]} on columns
7  from
8    [Report_Tickets]
9  where
10   [Date Opened].[Months From Today].&[1] :
11   [Date Opened].[Months From Today].&[6]
```

	Tickets Count	Over 300 Tickets
September, 2018	214	No
October, 2018	292	No
November, 2018	266	No
December, 2018	247	No
January, 2019	334	Yes
February, 2019	273	No

Figura 5.9: Risultati della query 3

MDX permette anche di creare misure con visibilità a livello di query, è possibile quindi definire delle formule che, basandosi sulle misure già esistenti, **personalizzano** i risultati.

Spesso si richiede che i dati mostrati siano dipendenti dal **momento** in cui si effettua la query, ad esempio si potrebbero voler vedere solo i 6 mesi precedenti, per questo motivo è stato creato l'attributo *Months From Today*, in questo modo possiamo implementare la funzionalità.

Se si inserisse l'attiributo *Months From Today* direttamente nella query si otterrebbero solamente dei numeri, volendo ottenere i mesi sulle righe come nei precedenti esempi si introduce la clausola *where* che funziona da slicer, in questo modo si possono filtrare i dati mantenendo i mesi.

Si può vedere il risultato nella figura 5.9, la misura definita inizialmente è "Yes" se *Tickets Count* è superiore a 300, "No" altrimenti.

5.9 Report Power BI

Una volta effettuata la connessione **live** con SSAS è possibile vedere sul lato destro i campi e le visualizzazioni disponibili, al centro è presente il foglio di lavoro.

Come è possibile vedere in figura 5.10, in basso a destra si ha la possibilità di inserire i campi del cubo all'interno dei vari grafici, come gli assi, la legenda e i filtri. Il **design** di ogni vista può essere modificato per quanto riguarda **titolo**, **colore** dei grafici, **etichette** dei dati, ecc.

La *Live connection* si aggiorna continuamente, in questo modo ogni cambiamento sul cubo viene mostrato immediatamente. Questo tipo di connessione permette di configurare automaticamente gli **slice** in fase di consultazione, in questo modo cliccando ad esempio la colonna di un istogramma si limiteranno tutte le

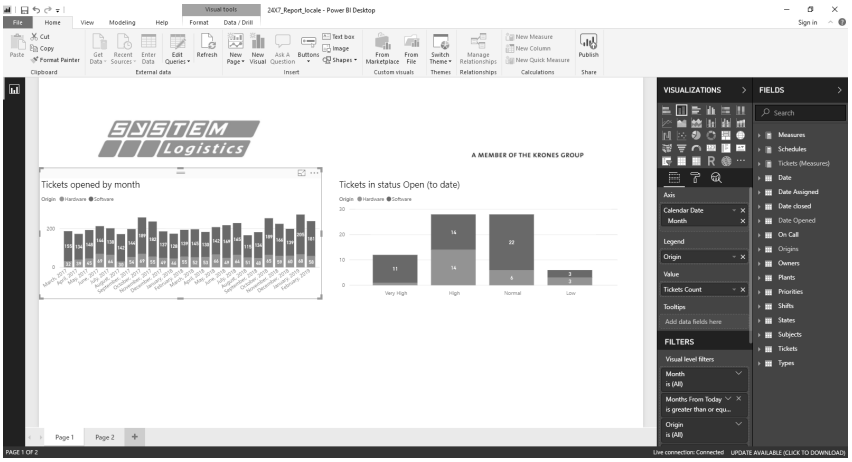


Figura 5.10: Report con Power BI

altre visualizzazioni a mostrare i dati rappresentati da quella colonna, senza la live connection si dovrebbe configurare questa funzionalità manualmente.

Per quanto riguarda la **distribuzione** si è preferito utilizzare solo la versione **desktop**, in quanto a livello aziendale l'utilizzo di Power BI al momento non è tale da giustificare l'acquisto di licenze che permettono la distribuzione a più persone, facendo così gli interessati utilizzeranno il file in formato *.pbx* per aprire e leggere il report a patto di aver installato Power BI Desktop sul proprio PC.

5.10 Report Excel

In questa sezione si vedrà come è stato costruito il sistema di **reportistica**, partendo da *Report Configurator*, vedendo come è stato costruito il database con le sue viste, per poi descrivere il funzionamento di *Report Sender* ed infine vedere come è stato costruito il template del report in Excel.

5.10.1 Report Configurator

Report Configurator è un'applicazione **web** che si basa su Django. Il database (il cui diagramma è in figura 4.3) su cui questa applicazione si appoggia è stato costruito tramite il model che Django permette di creare.

Il model quindi contiene tutte le entità di cui Report Configurator ha bisogno per modellare i **report** ed il loro invio. All'interno del model sono descritti 5

(a) Lista dei report (b) Configurazione report

(c) Configurazione schedulazione (d) Lista dei destinatari

Figura 5.11: Funzionalità di Report Configurator

oggetti diversi: **Report**, **ReportBlob**, **Receiver**, **Schedule**, **MonthlySchedule**. Le operazioni di questi oggetti sono descritte all'interno del capitolo sulla **progettazione**.

Le view si occupano di fornire i form necessari alle operazioni di **aggiunta** di un report o di un receiver, **avvio** manuale di un task, **download** di un template o di uno snapshot.

5.10.2 Report Sender

Report Sender si occupa di inviare via **mail** un report al suo avvio. Di seguito si vedrà nel dettaglio come funziona questa applicazione.

L'entry point del programma è il file **main.py**, che per prima cosa tramite **argparse** fornisce una serie di **parametri** per modificare il comportamento del programma.

```
1 def main():
2   parser = argparse.ArgumentParser()
3   group = parser.add_mutually_exclusive_group()
4   group.add_argument("-l", "--list", action="store_true", help = "list reports")
5   group.add_argument("-lr", "--list_report", type = int, help = "show report
      info")
```

```
6   group.add_argument("-a", "--all", action="store_true", help = "sends every
        reports")
7   group.add_argument("-id", type = int, help = "id of the report to send")
8   parser.add_argument("-c", "--config", help = "ini file for configuration",
        default = "config.ini")
9   parser.add_argument("-ns", "--no_send", help = "don't email the report",
        action="store_false")
10  args = parser.parse_args()
11  if args.list == False and args.all == False and args.id is None and args.
        list_report is None:
12    parser.print_help()
13  config_file = args.config
14  report_id = args.id
15  all_reports = args.all
16  send_mail = args.no_send
17  list_reports = args.list
18  report_info = args.list_report
19
20  config = configparser.ConfigParser()
21  config.read(config_file)
22  db_server = config['DB']['server']
23  database = config['DB']['database']
24  username = config['DB']['username']
25  password = config['DB']['password']
26  mail_server = config['MAILSERVER']['server']
27  sender = config['MAILSERVER']['sender']
28  snapshots_dir = config['REPORTS']['snapshots_dir']
29  cache_dir = config['REPORTS']['cache_dir']
30
31  rm = DatabaseModule(db_server, database, username, password)
32  reports_to_send = []
```

Grazie ad argparse possiamo creare un sistema affidabile di gestione degli **argomenti**, potendo dare valori di default e creando anche un **helper** in caso di errata formattazione del comando.

Successivamente si vanno a leggere i parametri di **configurazione** all'interno del file .ini ed infine ci si collega al database tramite la classe **DatabaseModule**.

Se si è selezionato uno o più report si passa alla fase di **creazione** degli snapshot e del relativo **invio**. Inizialmente si riempie la lista reports_to_send con i report da inviare.

```
52  if all_reports:
53    reports_to_send = rm.get_reports()
54  else:
55    report = rm.get_report(report_id)
56    if report is None:
57      rm.log(f"Report with id {report_id} doesn't exists")
58    else:
59      reports_to_send.append(rm.get_report(report_id))
```

```
60
61  for report in reports_to_send:
62    if not report.enabled:
63      rm.log(f'Report "{report.name}" is disabled')
64      continue
65    try:
66      with ExcelSnapshot(report.file, cache_dir) as snapshot:
67        if report.has_macro():
68          snapshot.execute_macro(report.module, report.routine)
69        filename = hashlib.sha256(os.urandom(8)).hexdigest()
70        report.snapshot_path = snapshot.save_to(snapshots_dir, filename, 'xlsm')
71      if send_mail:
72        sender = ReportSender(mail_server, sender)
73        receivers = rm.get_receivers_for_report(report)
74        sender.send_email(report)
75        rm.log(f'{report.name} sent successfully')
76      else:
77        rm.log(f'{report.name} created successfully')
78    except Exception as e:
79      rm.log(traceback.format_exc())
```

Una volta riempita la lista si itera su ogni report, si crea uno snapshot tramite **ExcelSnapshot**, lo si salva nella cartella di cache dopo aver eseguito la macro (se configurata) ed infine lo si invia ai destinatari con ReportSender.

Si va quindi rapidamente a vedere le classi **DatabaseModule**, **ExcelSnapshot** e **ReportSender**.

DatabaseModule effettua l'interrogazione del database tramite pyodbc, all'interno del costruttore si crea la connessione:

```
1  def __init__(self, server = '', database = '', username = '', password = ''):

2    self.conn = pyodbc.connect('DRIVER={SQL Server};SERVER='+server+';DATABASE='
       +database+';UID='+username+';PWD='+password)
```

Ed in seguito si effettuano le **interrogazioni** inviando la query al database e leggendone i risultati:

```
9  def get_receivers_for_report(self, report):
10   sql = """select
11           Receivers.id, name, surname, mail
12         from
13           Receivers
14         join
15           Report_Receivers on Receivers.id = receiver_id
16         where
17           report_id = ?"""
18   receivers = self.conn.cursor()
19   receivers.execute(sql, report.id)
```

```
20      return [Receiver(rec[0], rec[1], rec[2], rec[3]) for rec in receivers]
```

La wildcard *?* serve per inserire i **parametri** all'interno della query in maniera sicura.

ExcelSnapshot si occupa di creare uno snapshot a partire da un file Excel, il compito viene svolto utilizzando **pywin32** che fornisce un interfaccia per comunicare con i processi di Windows. Il costruttore della classe scrive il file all'interno della cartella di cache utilizzando un nome casuale, in seguito viene aperto creando un nuovo processo di Excel.

```
1   def __init__(self, file, cache_folder = ''):
2     self.filename = hashlib.sha256(os.urandom(8)).hexdigest() + '.xlsm'
3     self.path = os.path.join(cache_folder, self.filename)
4     with open(self.path, "wb") as new_file:
5       new_file.write(file)
6     self.app = win32com.client.Dispatch('Excel.Application')
7     self.wb = self.app.Workbooks.Open(self.path)
```

La classe mette a disposizione alcuni metodi per l'esecuzione di macro, il salvataggio e la chisura del processo.

```
1   def execute_macro(self, module, routine):
2     if module is not None:
3       self.app.Application.Run('{0}!{1}.{2}'.format(self.filename, module,
            routine))
4
5   def save_to(self, path, filename, extension):
6     new_file = os.path.join(path, f'{filename}.{extension}')
7     self.wb.SaveAs(new_file)
8     return new_file
9
10  def close(self):
11    self.wb.Close(False)
12    self.app.Quit()
13    os.remove(self.path)
```

In ultimo la classe **ReportSender** si occupa di inviare una mail con il report allegato ad una lista di destinatari.

```
1   def __init__(self, server = '', sender = ''):
2     self.server = server
3     self.sender = sender
4
5   def send_email(self, report):
6     msg = MIMEMultipart()
```

```
7       msg['From'] = self.sender
8       msg['To'] = COMMASPACE.join([f'{rec.name} {rec.surname} <{rec.mail}>' for
            rec in report.receivers])
9       msg['Date'] = formatdate(localtime=True)
10      msg['Subject'] = report.subject
11
12      msg.attach(MIMEText(report.body))
13
14      with open(report.snapshot_path, 'rb') as fil:
15        part = MIMEApplication(fil.read(), Name=basename(report.snapshot_path))
16      part['Content-Disposition'] = f'attachment; filename="{report.filename}.xlsm
            "'
17      msg.attach(part)
18
19      server_conn = smtplib.SMTP(self.server)
20      server_conn.sendmail(self.sender, [rec.mail for rec in report.receivers],
            msg.as_string())
21      server_conn.close()
```

Il metodo send_email costruisce l'header della mail inserendo il file del report come allegato.

5.10.3 Excel template

Il template è composto da un file Excel composto da cinque fogli di lavoro:

1. **Ticket Status**: contiene informazioni essenzialmente riguardanti il numero di ticket suddivisi per diversi fattori: mesi, priorità, ecc.

2. **Resources Saturation**: riguarda la saturazione delle risorse, quanto tempo viene impiegato da ogni operatore sulle diverse attività.

3. **Tickets Status Database**: mostra i ticket più impegnativi per tempo impiegato nelle varie categorie: problemi, modifiche e richieste.

4. **Timing**: riguarda i tempi di apertura, assegnazione, riavvio e chiusura dei ticket.

5. **Data**: foglio nascosto che contiene tutte le tabelle a supporto dei grafici.

Le query effettuate vengono tutte **caricate** sul foglio "Data", in questo modo sarà possibile salvare il foglio **scollegato** dalla rete aziendale una volta cancellate le query, per ottenere questo viene utilizzata una **macro** VBA che viene chiamata da Report Sender al momento della creazione dello snapshot:

```
Sub unlink()
    For Each wksheet In ThisWorkbook.Worksheets
        For Each listObj In wksheet.ListObjects
            listObj.unlink
        Next listObj
    Next wksheet

    For Each q In ThisWorkbook.Queries
        q.Delete
    Next q

    For Each C In ThisWorkbook.Connections
        C.Delete
    Next C
End Sub

Sub update()
    For Each q In ThisWorkbook.Connections
        q.OLEDBConnection.BackgroundQuery = False
    Next q
    ThisWorkbook.RefreshAll
    DoEvents
End Sub

Sub update_unlink()
    Call update
    Call unlink
End Sub
```

La subroutine **update** si occupa di aggiornare tutte le query create all'interno del template, occupandosi di **disabilitare** l'aggiornamento in background in modo da non rilasciare il controllo al chiamante prima di aver terminato tutti gli aggiornamenti. La subroutine **unlink** elimina dal workbook tutte le query e le connessioni. L'ultima subroutine sarà quella chiamata da Report Sender, in modo da creare un workbook **statico**, utilizzabile senza essere collegati alla rete aziendale.

Per quanto riguarda i **grafici** che possiamo vedere in figura 5.12 possiamo notare una combo box al di sopra di ogni visualizzazione, grazie a questa gli utenti potranno **selezionare** il grafico relativo ad una priorità, in modo da non far risultare il grafico troppo pesante includendo tutte le priorità allo stesso momento, il cambio dei dati del grafico viene effettuato utilizzando questa subroutine:

```
Sub SelectTime(ByVal chrt As String, ByVal priority As String, ByVal chart_type
    As Integer)
    Dim tbl As String
    Select Case chart_type
        Case CHART_RESOLUTION
```

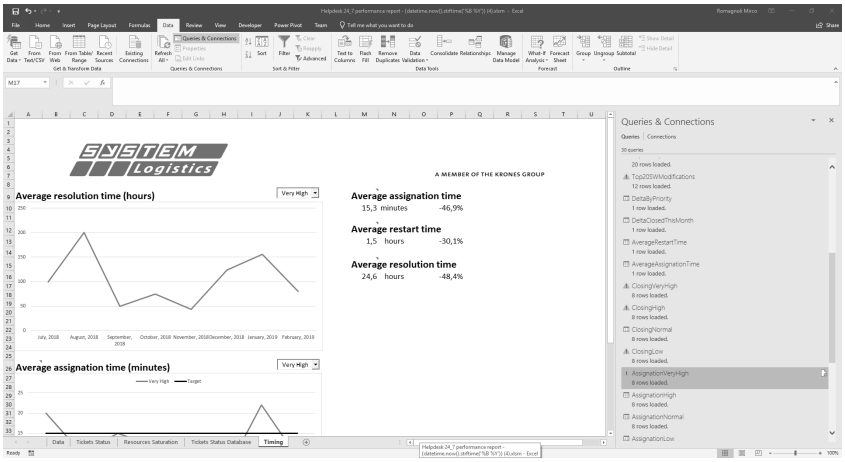


Figura 5.12: Template del foglio di lavoro Timing

```
            tbl = getTableNameResolution(priority)
        Case CHART_ASSIGNATION
            tbl = getTableNameAssignation(priority)
        Case CHART_RESTART
            tbl = getTableNameRestart(priority)
        Case Else
            Exit Sub
    End Select
    Set ran = ThisWorkbook.Sheets(DATA_TABLE).Range(tbl & "[#All]")
    Set wk = Worksheets("Timing")
    Set chrObj = wk.ChartObjects(chrt)
    chrObj.Chart.SetSourceData Source:=ran
    chrObj.Chart.SeriesCollection(1).Format.Line.ForeColor.RGB = getTableColor(
        priority)
    If chart_type = CHART_ASSIGNATION Then
        chrObj.Chart.FullSeriesCollection(2).Format.Line.ForeColor.RGB = 1
    End If
End Sub
```

Questa, collegata all'evento *Change* di ogni combobox si occupa di caricare i dati corretti dal foglio Data, mostrando di volta in volta il grafico richiesto.

Una volta conclusa la costruzione del template è sufficiente inserirlo come template all'interno di Report Configurator, in questo modo verrà inviato con la schedulazione mensile richiesta.

Conclusione

L'obiettivo prefissato all'inizio del tirocinio è stato raggiunto con successo, tuttavia il progetto è ancora in fase di trasformazione, i membri dell'azienda si stanno avvicinando a questo tipo di tecnologie e l'interesse per questo tipo di report richiede un lavoro periodico di manutenzione ed aggiunta di visualizzazioni per soddisfare ogni requisito.

Questo progetto ha tantissimi margini di crescita, la struttura permette di essere utilizzato su un grande numero di report differenti ed il data warehouse ha la possibilità di fornire tantissime visualizzazioni utili al decision-making, senza dimenticare il data mining, non usato in questo caso per mancanza di un numero di dati sufficientemente grande.

Questo tirocinio è stato molto utile per il mio sviluppo come programmatore e come persona, ho trovato dei colleghi estremamente preparati e motivati da cui ho avuto modo di attingere conoscenze sui metodi di lavoro e le tecnologie di cui ero sprovvisto.

Ho avuto la possibilità di affinare le mie conoscenze di SQL lavorando su un progetto maturo e scritto da una comunità di persona, di imparare linguaggi nuovi come C# e VBA che hanno ancora una buona importanza. La cosa più importante che ho imparato da questo tirocinio è stato il poter imparare a lavorare con altre persone e gestire la pressione delle richieste che mi sono state poste.

Bibliografia

[1] Jiawei Han, Micheline Kamber, Jian Pei.
Data Mining: Concepts and Techniques. Morgan Kaufmann, 3 edizione, 2011.

[2] Robert Wrembel, Christian Koncilia.
Data Warehouses and OLAP: Concepts, Architectures and Solutions. IRM Press, 2007

[3] Vincent Rainardi.
Building a Data Warehouse With Examples in SQL Server. Apress, 2008