Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in

STATISTICA E TECNOLOGIE INFORMATICHE

RELAZIONE FINALE

Esperienza di stage presso TeamSystem Wellness: ottimizzazione delle prestazioni di database con mole di dati prossima al vincolo strutturale di SQL Server Express Edition

Relatore Prof. Mauro Migliardi
Dipartimento di Ingegneria dell'Informazione

Laureando: Lorenzo Basso
Matricola n. 1078989

Anno Accademico 2015/2016

Ai miei genitori

INDICE

1. INTRODUZIONE

La presente relazione ha lo scopo di presentare il progetto (e la sua implementazione) svolto durante l'attività di stage presso InforYou S.r.l. nel periodo compreso tra il 06/06/2016 e il 12/08/2016.

Il progetto racchiude interamente l'analisi prestazionale del database di un cliente e l'implementazione di uno script atto a migliorarne le funzionalità in termini di prestazioni e spazio di archiviazione.
La prima parte della relazione descrive l'azienda che mi ha ospitato, di cosa si occupa e soprattutto, in riferimento ai prodotti che offrono ai loro clienti, presenta le caratteristiche della mia attività di stage, le problematiche rilevate dalla situazione iniziale e gli obiettivi finali imposti. Nella parte centrale prima vengono presentati gli strumenti (programmi, linguaggi, funzionalità) utilizzati per l'implementazione e le loro caratteristiche, successivamente delinea, fase per fase, come lo script agisce e che modifiche significative comporta. Infine vengono descritti gli effetti che lo script ha apportato, quindi quali sono i vantaggi ricavati.

Dal punto di vista personale, sono molto soddisfatto di aver svolto questo intership, perché mi ha dato un primo vero contatto con il mondo del lavoro e l'ho fatto in un ambito molto gradito da parte mia, che coinvolge il settore informatico in un ambito sportivo. Inoltre mi ha dato la possibilità di avere nuove conoscenze, sia teoriche sia pratiche, che porterò avanti nella carriera lavorativa.

2. L'AZIENDA

2.1 Descrizione

Fondata nel 1997, Inforyou S.r.l. è una software house che offre un'ampia scelta di software gestionali e servizi per aziende e professionisti.

Nel 2013 si incorpora al gruppo TeamSystem diventando TeamSystem WELLNESS e, con circa una trentina di dipendenti tra una sede a Roma e una a Castello di Godego (TV), si occupa di soluzioni gestionali e controllo accessi per il settore dello sport, del benessere e del tempo libero.

I clienti a cui si dedica sono infatti palestre, piscine, SPA, centri benessere, impianti sportivi, terme, circoli, ecc., offrendo loro i livelli di gestione in tutti i processi commerciali, di marketing e di customer service, integrando all'interno di un unico sistema il patrimonio informativo aziendale.

L'azienda è organizzata in tre divisioni:

• Ricerca e sviluppo: si occupa di innovazione del prodotto e sviluppo di nuove soluzioni;

• Divisione Tecnica: offre assistenza con call center, servizio di reperibilità, corsi di formazione e consulenza ai clienti;

• Divisione Commerciale: analisi e formulazione di offerte personalizzate, anche con sopralluogo e supporto presso il cliente.

2.2 Prodotti offerti

2.2.1 Inforyou Manager

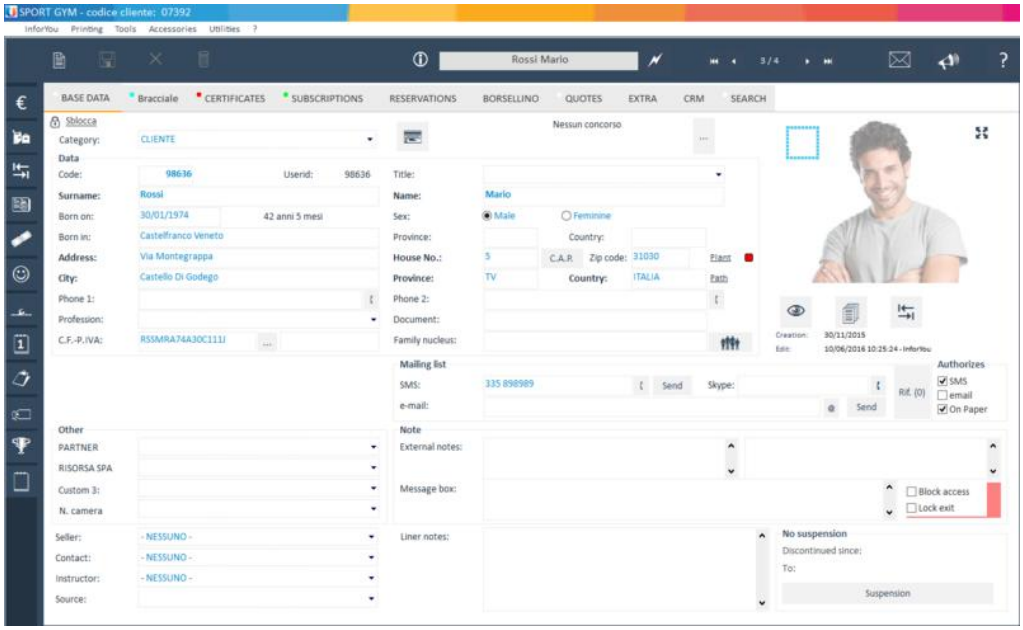

Figura 1 – Gestione anagrafica clienti

Il software si compone di un programma gestionale principale **InforYou Manager,** è il gestionale a disposizione degli imprenditori dello sport, del benessere e del tempo libero per gestire le attività della propria azienda.
Ad esso vengono integrate una serie di moduli aggiuntivi che consentono di realizzare una soluzione personalizzata per tutti i tipi di azienda.

L'interfaccia consente la completa gestione dell'impianto: processi commerciali, di marketing e di customer service, integrando all'interno di un unico sistema tutto il patrimonio informativo aziendale.

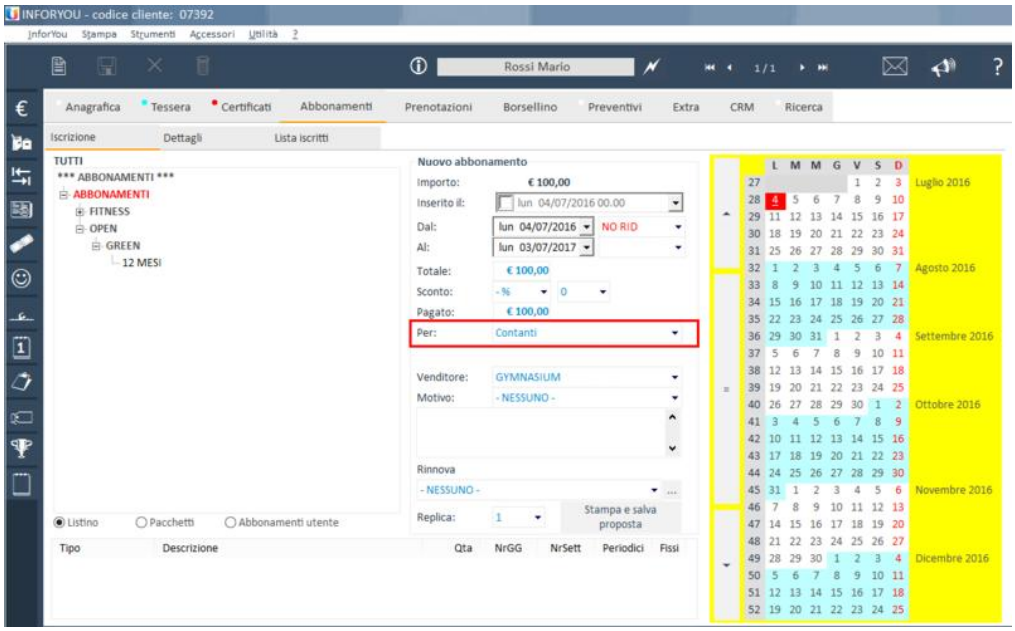

Figura 2 – Gestione abbonamenti

Il software è strutturato in tre parti:

1) Operativa per reception e forza vendita

   Permette di gestire clienti e contatti, e vendere servizi (abbonamenti, ticket, prodotti) in maniera semplice e veloce.

2) Programmazione per direttori e club-manager

   Un pannello di controllo che consente di programmare tutte le attività del centro.

3) Direzione per imprenditori e gestori

   Tutte le informazioni raccolte durante l'utilizzo di InforYou vengono aggregate e visualizzate tramite cruscotti di monitoraggio, consuntivi economici, statistiche ed indicatori di performance.

2.2.2 Controllo Accessi

Abbinati al gestionale, TeamSystem WELLNESS offre anche articoli hardware per l'accesso alle infrastrutture da parte degli utenti.
In supplemento, l'azienda fornisce ai propri clienti varchi, tornelli e dispositivi di identificazione degli utenti (tessere e bracciali), prodotti per il controllo degli accessi agli impianti o a specifiche aree, come ad esempio saune e piscine.

Il flusso degli utenti è gestito in base alle regole impostate in InforYou Manager: ogni transito viene registrato dal gestionale.

Figura 3 – Lettori tessere e bracciali

Figura 4 – Tornelli e varchi

3. ATTIVITA' DI STAGE

3.1 Descrizione del problema

La piattaforma Inforyou Manager raccoglie le informazioni direttamente dal database del cliente e tramite istruzioni (le query) inserisce, legge, aggiorna e cancella dati da esso.

Caratteristiche dei database dei clienti

I database in oggetto vengono supportati da SQL Server, il sistema di gestione dati rilasciato da Microsoft, di cui esistono diverse edizioni.
Le edizioni base gratuite e liberamente utilizzabili sono di **SQL Server Express**: ma, essendo a costo zero, ne derivano diversi limiti, tra i quali il più significativo è lo spazio di archiviazione per database vincolato a dieci gigabyte.

Necessità del cliente

Capita però a molte imprese, soprattutto nel caso di multi-azienda, di dover gestire grandi moli di dati che si avvicinano al limite prefissato e che questo provochi l'involontario rallentamento e malfunzionamento prestazionale della loro piattaforma gestionale.
Sarebbero quindi costretti ad utilizzare altre versioni commerciali di SQL Server, che richiedono però costi aggiuntivi che possono arrivare fino a migliaia di euro.

3.2 Obiettivi dello stage

L'obiettivo prefissato all'inizio dello stage è stato quello di trovare una soluzione al problema citato, cioè migliorare le prestazioni del database e garantire ai clienti l'utilizzo di un'edizione Express di

SQL Server, evitando loro l'acquisto di altre edizioni che comportano costi aggiuntivi.

Nelle prime settimane della mia esperienza sono entrato a contatto con il prodotto dell'azienda che mi ha ospitato: ho studiato il loro personale modello di database e ho appreso l'utilizzo del programma Microsoft SQL Server Management Studio. Nel resto del tempo ho realizzato uno script che opera direttamente sui database, lasciando invece del tutto invariate le funzionalità di Inforyou Manager.

Con la presente relazione vengono presentati gli strumenti utilizzati e le implementazioni attuate per la soluzione del problema.

4. STRUMENTI UTILIZZATI

Essendo il progetto caratterizzato sulla gestione di basi di dati, sono risultate utili allo scopo le funzionalità del linguaggio SQL.

Vengono presentati a seguire i programmi, linguaggi e strumenti utilizzati per l'implementazione del progetto.

4.1 Linguaggio T-SQL

L'SQL (Structured Query Language) è il linguaggio standardizzato che permette di operare su modelli relazionali (creare tabelle e di modificarne la struttura, eseguire ricerche generiche o mirate tra i dati, inserire, modificare e cancellare i dati, oppure di lavorare su più tabelle contemporaneamente).

T-SQL, abbreviazione di Transact-SQL, è un linguaggio proprietario della Microsoft ed è un'estensione del linguaggio SQL base. Comprende quindi tutte le istruzioni di SQL (CREATE, ALTER, SELECT, INSERT, UPDATE, DROP, ecc.) e ad esse vengono integrate nuove funzionalità, che comunemente si trovano negli altri linguaggi di programmazione.
Non si tratta più di un linguaggio esclusivamente dichiarativo come l'SQL di base, ma le nuove implementazioni rendono T-SQL un linguaggio imperativo.

4.1.1 Dichiarazione di variabili locali

L'uso delle variabili risulta molto utile quando nell'implementazione e permette di lavorare con funzioni e procedure.

Attraverso l'istruzione **DECLARE** è possibile creare variabili locali nominandole con il carattere **@** e dichiarandone il tipo di dato (int, varchar, date, ecc.).

```
DECLARE @ndb NVARCHAR(MAX);
DECLARE @ndbsupp NVARCHAR(MAX);
```

In seguito si assegna ad ogni variabile un valore attraverso le istruzioni **SET** o **SELECT**, che può essere sia un valore fisso sia derivato da un'istruzione SELECT di SQL.

```
SET @ndb = DB_NAME();

SELECT @ndbsupp =
( SELECT Valore FROM impostazioni WHERE Descrizione ='NomeDBSupporto');
```

In questo modo il valore di @ndb sarà il nome del database corrente e in @ndbsupp sarà contenuto il valore del campo 'Valore' del record della tabella Impostazioni nel quale il valore di Descrizione è "NomeDBSupporto".

4.1.2 SQL dinamico

Dynamic SQL è una potente tecnologia di programmazione di database che consente di concatenare ed eseguire istruzioni T-SQL in modo dinamico in una variabile stringa in fase di esecuzione.

L'esecuzione di codice T-SQL dinamico è una pratica che spesso risulta molto utile durante la programmazione. Per poterlo fare si usa l'istruzione **EXEC**.

Esempio tratto dallo script:

```
EXEC ('select top(1) nome from #tab_tomove where nome not in (select
name from ['+@ndbsupp+'].sys.tables) order by nome');
```

Il vantaggio che si ha usando questa procedura è quello di eseguire delle query ponendo al loro interno dei parametri, definiti da variabili locali, che vengono forniti al momento dell'esecuzione.

4.1.3 Controllo dei flussi:

Come in tutti i linguaggi di programmazione anche il Transact-SQL richiede la possibilità di controllare il flusso di esecuzione, in base a determinate condizioni dettate dal funzionamento interno del programma.
E' quindi necessario controllare il verificarsi o meno di una condizione booleana ed eseguire in base ad essa una serie di istruzioni oppure un'altra.

Le istruzioni **IF** e il loop **WHILE** caratterizzano questo aspetto**.** A entrambe sono integrate le istruzioni **BEGIN END**, che vengono utilizzate per raggruppare più istruzioni SQL o T-SQL in un unico blocco.

IF è il costrutto di selezione più semplice e consente di eseguire o meno un blocco di codice al verificarsi o meno di una condizione.

Esempio tratto dallo script:

```
IF NOT EXISTS (SELECT * FROM impostazioni WHERE Descrizione =
'NomeDBSupporto')
    BEGIN
      INSERT INTO impostazioni (Descrizione, Valore) VALUES
      ('NomeDBSupporto', @ndb+'_supporto');
    END;
```

L'istruzione di INSERT avviene solo se la condizione specificata risulta vera.

Il ciclo WHILE è un costrutto iterativo in quanto permette di eseguire ripetutamente un determinato blocco di istruzioni. Il numero di cicli da eseguire è inizialmente indeterminato, infatti dipende da una condizione booleana ripetutamente testata.

Esempio tratto dallo script:

```
DECLARE @tab_nofk NVARCHAR(MAX)=
( SELECT TOP 1 * FROM #tab_tomove WHERE nome IN ( SELECT name FROM
sys.tables));

WHILE @tab_nofk IS NOT NULL
    BEGIN
        EXEC ('drop table '+ @tab_nofk +';');

      SELECT @tab_nofk = ( SELECT TOP (1) * FROM #tab_tomove WHERE nome
      IN ( SELECT name FROM sys.tables ));
    END;
```

Con queste istruzioni viene letto uno alla volta il nome di una tabella e sempre una alla volta questa tabella viene eliminata dal database.

4.2 MICROSOFT SQL SERVER EXPRESS

SQL Server è il sistema di gestione di basi di dati (DBMS) di Microsoft.
Utilizza **Management Studio** (SSMS) come applicazione software per la configurazione, la gestione e l'amministrazione di tutti i componenti all'interno di Microsoft SQL Server. Lo strumento include sia gli editor di script sia strumenti grafici che lavorano con gli oggetti e le caratteristiche del server.

Il DBMS di casa Microsoft è stato lanciato nel 1988 e ha subito diversi miglioramenti e restyling nel corso degli anni fino a giungere alla versione 2008, oggetto della presente guida. Le modifiche più significative in realtà sono state introdotte con SQL Server 2005, in cui varie aree sono state riprogettate e migliorate rispetto alle versioni precedenti e in cui è stata realizzata l'integrazione con le funzionalità del .NET Framework.

Il linguaggio di programmazione utilizzato è appunto **Transact-SQL.**

Come già accennato, ne esistono diverse edizioni a seconda delle esigenze: dalle edizioni standard e disponibili gratuitamente alle

edizioni per grandi aziende, offerte per soddisfare requisiti più avanzati (Enterprise, Business Intelligence).

L'edizione Express è un'edizione gratuita e consente lo sviluppo e l'utilizzo di applicazioni per desktop, web e server di piccole dimensioni e include dieci GB di spazio di archiviazione per database. Inoltre fornisce strumenti per la gestione delle istanze SQL e il Management Studio.

SQL Server Management Studio (SSMS)

Il Management Studio è lo strumento che permette di comunicare comandi e funzioni da un utente al servizio di SQL Server, il quale poi eseguirà le richieste sul sottostante database.

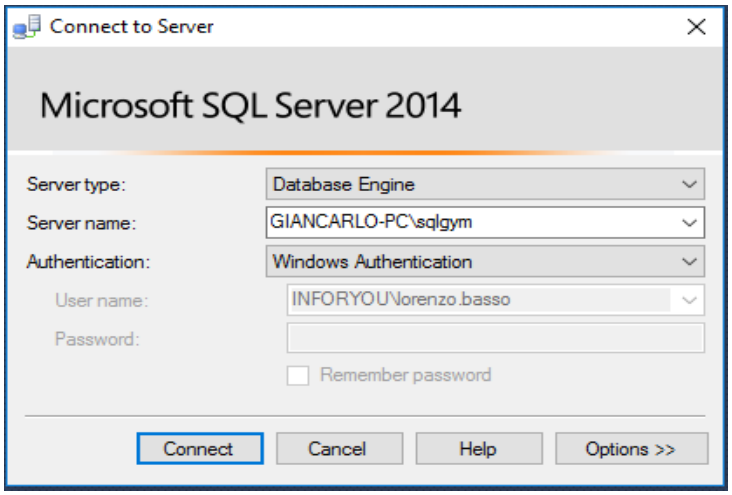

Figura 5 – Connessione al Server

All'avvio di SSMS viene specificato il tipo e il nome del server e l'autenticazione, che dipende dall'opzione scelta in fase di installazione (se si è scelto di installare con l'opzione Windows Authentication questa sarà l'unica opzione disponibile, altrimenti sarà possibile selezionare anche il tipo SQL Server Authentication inserendo le opportune informazioni di login).

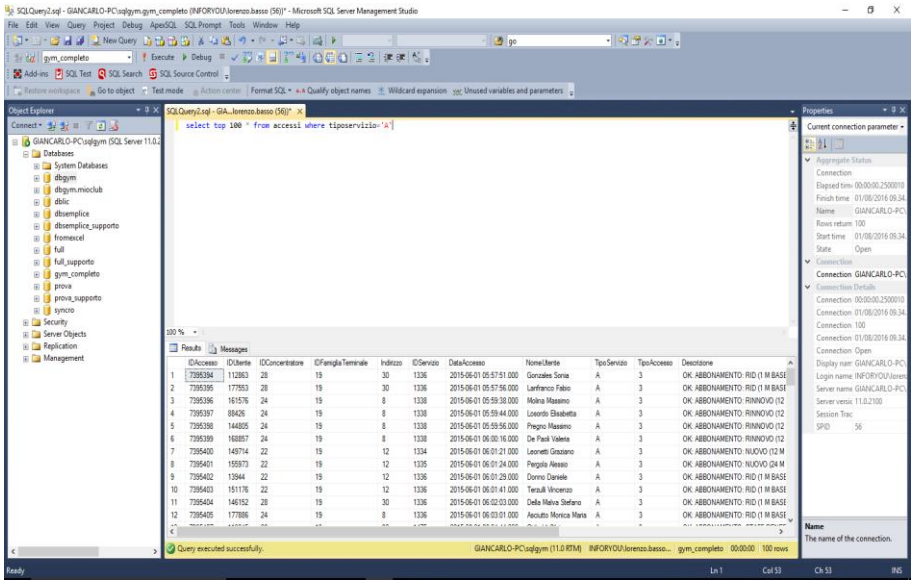

Figura 6 – SQL Server Management Studio

L'interfaccia si presenta come da figura 6.
La finestra di input permette un semplice utilizzo per l'inserimento delle istruzioni T-SQL, che vengono poi comunicate tramite il Management Studio e i risultati (output e/o numero di record presi in considerazione) visualizzati a video.

La componente più importante del SSMS è l'**Object Explorer**:

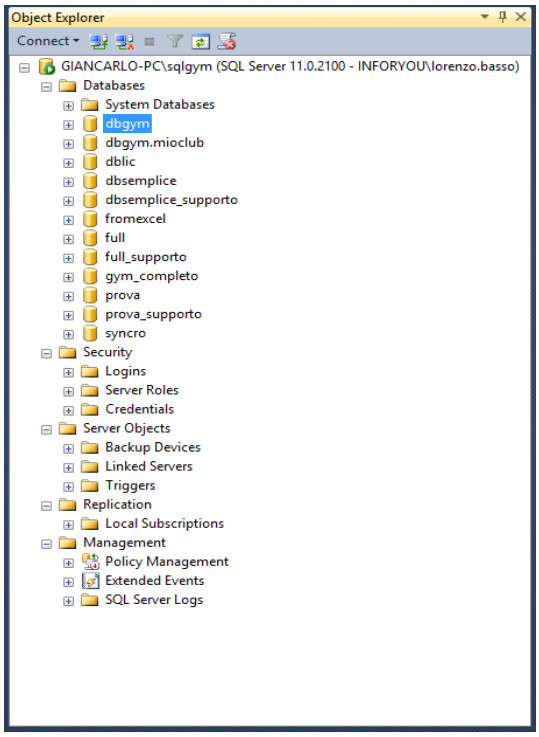

Figura 7 – Object Explorer

Le caratteristiche e i contenuti del server a cui ci si connette durante l'avvio sono presentati in questa sezione.
Come mostrato da figura 7, sono presenti diversi nodi:

- Databases – Contiene i database sia di sistema che di utente ai quale si è connessi
- Security – Contiene la lista dei login degli utenti che possono connettersi a SQL Server
- Server Objects – Contiene oggetti come periferiche di backup e fornisce la lista dei linked server (server remoti connessi)
- Replication – Mostra i dettagli relativi alla riproduzione dei dati dal database sul server ad un altro database sullo stesso server o su un altro
- Management – Permette di gestire i piani di manutenzione (maintenance plans), di gestire le policy, di gestire i log e i messaggi di errore

4.3 OGGETTI DEL DATABASE

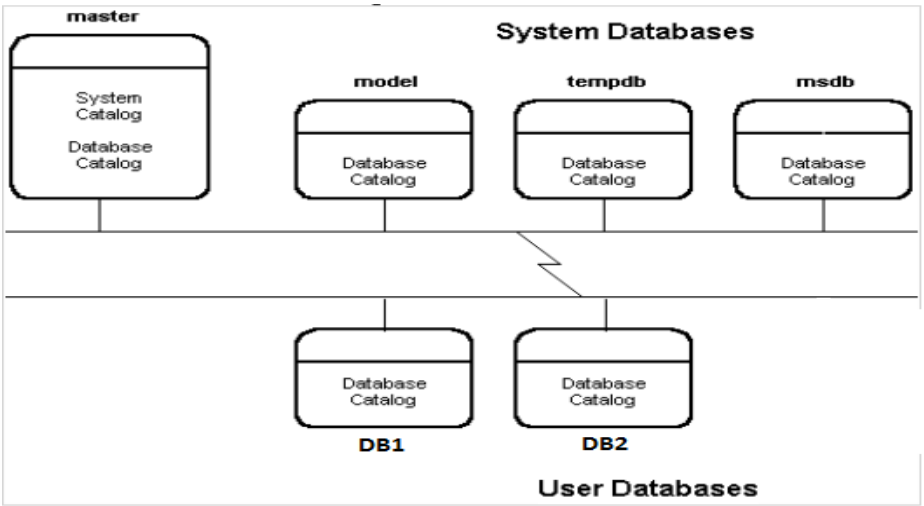

Figura 8 – Istanza di SQL Server

I database di sistema SQL Server contengono le informazioni necessarie al suo funzionamento, sono presenti dal momento dell'installazione e sono quattro:

Master contiene informazioni sul server (i login ed i ruoli relativi agli utenti, impostazioni di sistema, informazioni relative a tutti i database all'interno del server, i messaggi d'errore di sistema, ecc.

Model contiene il modello per la creazione di un database tipo.

Tempdb è un database temporaneo la cui durata corrisponde alla durata di una sessione di SQL Server. Contiene infatti tabelle e oggetti temporanei, contraddistinti dal carattere # nel nome (es. #tab_tomove).

Msdb fornisce le informazioni necessarie all'esecuzione automatica di attività (job) definite tramite il SQL Server Agent, un servizio di Windows che esegue tutti i job schedulati. Altri processi molto importanti che msdb permette sono quelli di backup e restore dei database presenti nel server.

4.3.1 Tabelle

Le tabelle sono oggetti che contengono tutti i dati disponibili in un database. Nelle tabelle, i dati sono organizzati in righe e colonne in un formato simile a quello di un foglio di calcolo. Ogni riga rappresenta un record univoco e ogni colonna rappresenta un campo all'interno del record.
Essendo i dati soggetti a vincoli nella maggior parte dei casi, le tabelle possono avere vincoli di integrità.

Vincoli di chiave primaria (primary key)
Il vincolo di chiave primaria è il più importante contrassegna l'unicità del recordset, per la tabella. Può essere costituito da una o più colonne, ma ne può esistere solo uno per ogni tabella.

Vincoli di chiave esterna (foreign key)
Il vincolo di chiave esterna (foreign key) coinvolge due tabelle (che possono essere due istanze della stessa tabella). La tabella

che contiene la chiave esterna è detta secondaria, mentre la tabella a cui fa riferimento la chiave esterna è detta principale. Gli attributi riferiti nella tabella principale devono formare una chiave candidata, di solito la chiave primaria. Il vincolo specifica che ogni valore non nullo della chiave esterna nella tabella secondaria deve corripondere ad un valore nella tabella principale.
Sono le foreign keys a creare relazioni tra le tabelle (da qui database relazionale), per estrapolare le informazioni legate tra di loro.

Vincoli predefiniti
Vengono a crearsi quando è SQL Server che assegna un valore, calcolato o fisso, a un determinato campo.
Ad esempio attribuire a un campo il valore della data odierna.

Vincolo non null
Semplicemente se un campo può ammettere valori NULL oppure no.

Vincolo check (di controllo)
Controlla che l'immissione di un dato in una colonna, appartenga ad un certo range di valori.

Vincoli unique
Una colonna si definisce unique, ovvero unica, quando ogni voce che contiene è unica per tutta la tabella. Questo campo è molto utile per evitare ridondanze di dati.
Anche la Primary Key ha lo stesso funzionamento ma a differenza di questa, si possono inserire più colonne unique in una tabella.

4.3.2 Sinonimi

I sinonimi sono una nuova funzionalità introdotta dalla versione 2005 di SQL Server che consente di specificare alias per altri oggetti di database (tabelle, viste, stored procedures e funzioni), anche appartenenti a un server o database differente.

Permettono le istruzioni SELECT, INSERT, DELETE, UPDATE, EXECUTE (quindi una gestione dei dati) e le eventuali modifiche applicate al sinonimo si ripercuotono sull'oggetto a cui fa riferimento.

Al momento della creazione del sinonimo, oltre al nome dell'oggetto a cui si riferisce, lo schema e il database a cui appartiene.

L'utilizzo dei sinonimi comporta molti vantaggi, in quanto dà accesso rapido agli oggetti di altri database, che non vengono alterati in caso di eliminazione del sinonimo.

Inoltre, l'alterazione (o lo spostamento in un altro database) dell'oggetto di riferimento non provoca disagi, in quanto non obbliga la riscrittura di tutto il codice di interrogazione ma alla distruzione e ricreazione del solo sinonimo.

4.3.3 Viste

Una vista è una sorta di tabella virtuale che non contiene fisicamente dati ma soltanto una query che l'utente definisce al momento della sua creazione. Si può pensare dunque ad una vista come una query fatta su una o più tabelle che viene memorizzata sul database.

Le viste quindi non sono uno strumento che serve a processare i dati, ma vengono utilizzate molto come misura di sicurezza, restringendo la visualizzazione dei dati da parte degli utenti solo a certe colonne o righe e restituendo dati riepilogativi anziché dettagliati.

Le viste quindi si utilizzano fondamentalmente per due scopi: uno è quello di raggruppare dati appartenenti a diverse tabelle ma correlati; l'altro è quello di consentire agli utenti di visualizzare informazioni specifiche provenienti da determinate tabelle senza che essi accedano direttamente alle tabelle per motivi, come accennato, anche di sicurezza.

4.3.4 Funzioni e stored procedures

L'utilizzo di funzioni risulta molto utile in quanto è possibile utilizzarle all'interno di una query per sfruttare i dati che esse restituiscono.

Transact-SQL permette di creare piccoli programmi o funzioni all'interno del database e poi farle eseguire dal programma esterno. Queste procedure si chiamano **Stored Procedure**.

Si tratta di moduli o routine che incapsulano codice in modo da poterlo riutilizzare. Una stored procedure può accettare parametri di input, restituire al client risultati tabulari o scalari, richiamare istruzioni DDL (Data Definition Language) e DML (Data Manipulation Language) e restituire parametri di output.

Una stored procedure (SP) è dunque un insieme di comandi T-SQL compilati, direttamente accessibili da SQL Server. Tali comandi vengono eseguiti come un'unica unità di lavoro (batch) sul server. Oltre a istruzioni SELECT, UPDATE e DELETE una SP possono richiamare altre SP, utilizzare istruzioni che controllano il flusso di esecuzione e funzioni di aggregazione.

Oltre alle SP create dagli utenti esistono centinaia di SP di sistema all'interno di SQL Server e cominciano con il prefisso "sp_".

Esempio di utilizzo di una store procedure:

```
exec sp_rename @tab_copy, 'temp_identity'
```

"sp_rename" è una store procedure di sistema che, passandole in input il nome di una tabella e un'altra stringa, la rinomina.

4.3.5 Triggers

I trigger sono degli oggetti di SQL Sever molto simili alle stored procedures, un blocco di codice che si attiva automaticamente dopo un determinato evento.

Gli eventi per i quali si attiva un trigger sono l'esecuzione di una istruzione INSERT, UPDATE, DELETE su una tabella di SQL Server. Il trigger viene ancorato ad una tabella e qualora si verifichi un evento tra quelli descritti prima si attiva eseguendo il codice T-SQL contenuto al suo interno.

Un utilizzo abbastanza diffuso è legato a diverse forme di validazione e controllo sui dati.
Un altro uso molto diffuso si può riscontrare quando è necessario effettuare modifiche sui dati di altre tabelle nel momento in cui si verifica una variazione dei dati sulla tabella contenente un trigger.

Esempio di trigger:

```
CREATE TRIGGER [dbo].[tr_Accessi_NotifyAccessQueue]
   ON  [dbo].[Accessi]
   AFTER INSERT
AS
BEGIN
       SET NOCOUNT ON;

    DECLARE @Message VARCHAR(MAX)
       SET @Message = '<ChangedRows>';

       SELECT @Message = @Message +
              ISNULL('<Inserted IDAccesso="' + CAST(ISNULL(IDAccesso, 0)
AS VARCHAR) +
              '" IDAzienda="' + CAST(ISNULL(IDAzienda, 0) AS VARCHAR) +'"
/>', '')
       FROM
              Inserted t0 INNER JOIN Terminali t1
              ON t0.IDConcentratore = t1.IDConcentratore AND
t0.IDFamigliaTerminale = t1.IDFamigliaTerminale AND t0.Indirizzo =
t1.Indirizzo
       SET @Message = @Message + '</ChangedRows>'
       DECLARE @DialogHandle UNIQUEIDENTIFIER
    BEGIN DIALOG @DialogHandle
    FROM SERVICE [IYAccessNotifications]
    TO SERVICE 'IYAccessNotifications'
    ON CONTRACT
[http://schemas.microsoft.com/SQL/Notifications/PostQueryNotification]
    WITH ENCRYPTION = OFF, LIFETIME = 30;
```

```
    SEND ON CONVERSATION @DialogHandle
    MESSAGE TYPE
[http://schemas.microsoft.com/SQL/Notifications/QueryNotification]
(@Message);
END
GO
```

Nell'esempio, al susseguirsi di ogni istruzione INSERT nella tabella Accessi, viene creata una stringa di testo @Message che contiene informazioni sul/sui record creati.

5. SOLUZIONE IMPLEMENTATA

Viene presentato in questo capitolo come lavora lo script T-SQL implementato e quali sono le modifiche che attua nel database.

La procedura consiste nell'aver trasferito alcune tabelle del database primario (e quindi della memoria occupata) in un altro database di supporto, con le stesse identiche caratteristiche del database contenente i dati (e quindi anch'esso con spazio di archiviazione di dieci Gigabyte).

Al completamento dell'implementazione hanno avuto molta importanza gli oggetti di tipo sinonimo, che hanno rimpiazzato le tabelle trasferite.

Inoltre, avendo modificato la struttura del database a cui il gestionale Inforyou Manager fa riferimento, sono stati inclusi vari blocchi di codice in supporto alla patch di aggiornamento del programma., risolvendo il problema.

Al termine dell'esecuzione dello script la situazione è quella di due database che compartiscono i dati delle tabelle.

5.1.1 Fase prima

Si crea, se non ancora esistente, il database di supporto. Ad ogni database, il proprio supporto sarà indicato, se esiste, nel campo "Valore" di "impostazioni" (tabella che contiene parametri e informazioni di quel specifico db) dove la descrizione è "NomeDBSupporto".

Si creano e si popolano le tabelle temporanee, che saranno stanziate, come già accennato, nel database di sistema TEMPDB.
Questi oggetti temporanei contengono varie informazioni riguardo la procedura di split, ad esempio quali sono le tabelle soggette al "trasferimento" da un database all'altro, se contengono campi identity, triggers, quanto spazio in memoria occupano.

Per la scelta delle tabelle da spostare sono state tenute le seguenti considerazioni:

- la tabella non deve avere vincoli di integrità referenziale con altre tabelle (tabelle con chiavi esterne che puntano esclusivamente a loro stesse vengono quindi prese in considerazione), in quanto sarebbe del tutto sconveniente eliminare le relazioni tra gli oggetti che popolano il database.

- la tabella deve occupare almeno 1 KB di memoria, poiché è preferibile non considerare tabelle con un uso irrilevante.
- per evitare sfavorevoli eccessi di spostamento di dati, il peso in memoria massimo delle tabelle da trasferire sarà sempre minore o uguale alla metà del peso complessivo di tutte le tabelle.

- per decisione personale e convenienza, sarà sempre possibile includere/escludere "a mano" dalla lista di trasferimento una specifica tabella.

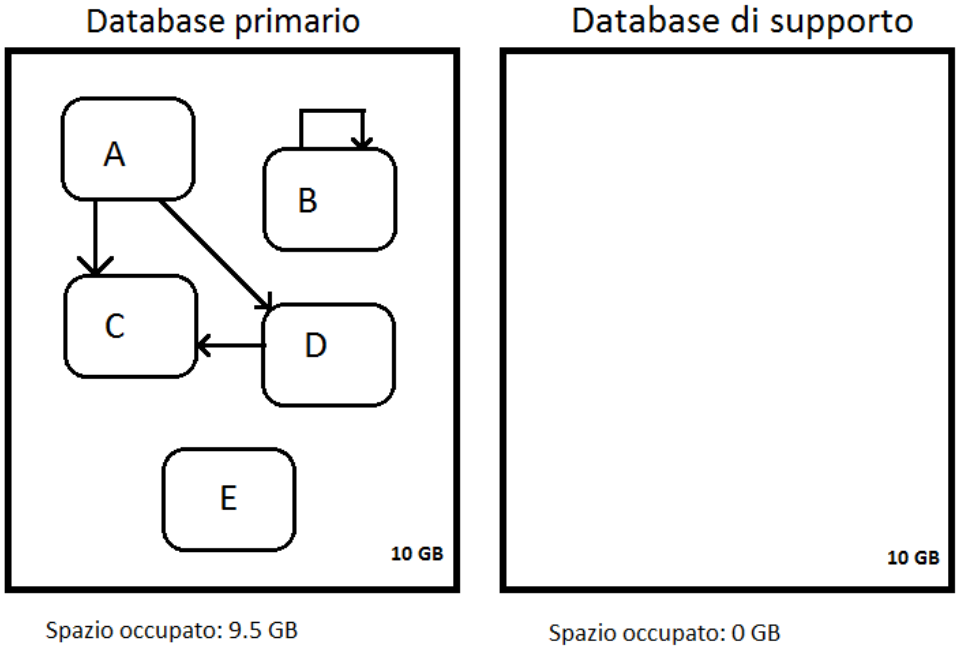

Figura 9 – Vengono selezionate come tabelle trasferibili le tabelle come B e E.

5.1.2 Fase seconda

Questa fase è il cuore dell'implementazione: è qui infatti che avviene lo split dal database primario a quello di supporto.

A questo punto sono state selezionate quante e quali tabelle verranno trasferite al database di supporto.

Si procede una tabella selezionata alla volta mediante un costrutto WHILE.

Con uno script di creazione, viene creata la tabella nel database di supporto. Viene introdotta nell'istruzione l'intera struttura della tabella, compresa di indici, vincoli e i triggers relativi.
Una volta creata, la tabella viene popolata con i record (copiati da un sinonimo temporaneo) tramite l'istruzione INSERT INTO.
Infine viene eliminata la tabella dal database primario (DROP TABLE).
L'eliminazione diretta non comporta errori, in quanto le tabelle non soggette non possiedono relazioni con altre tabelle presenti nel database.

La situazione corrente si presenta come in figura:

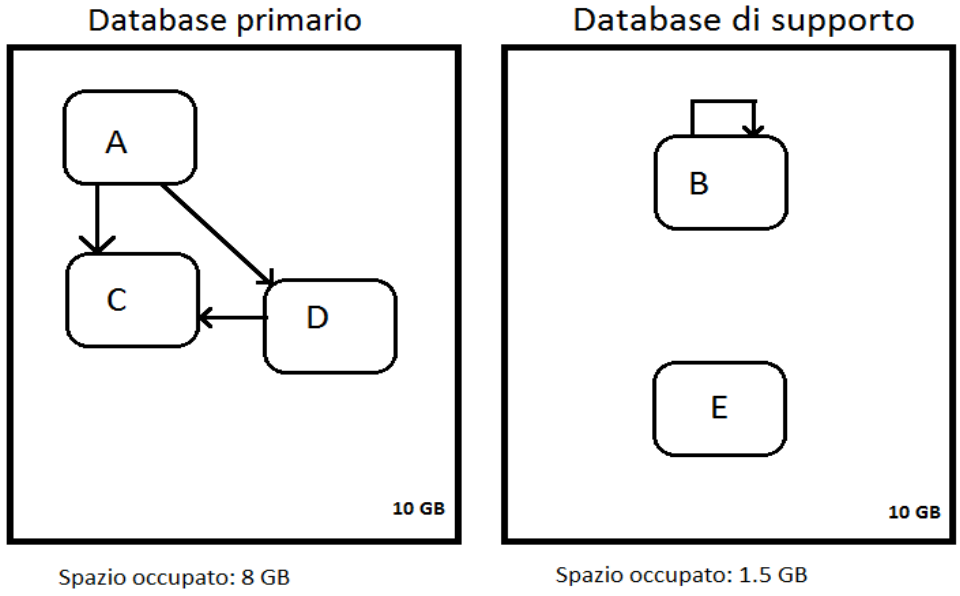

Figura 10

5.1.3 Fase terza

Infine, viene creato un sinonimo per ogni tabella presente nel database di supporto, ripristinando le funzionalità e il numero iniziale di oggetti presenti nel database di riferimento.

È molto importante che i sinonimi creati abbiamo lo stesso schema (dbo) e lo stesso nome della tabella a cui fanno riferimento, in modo che risultati e output di funzioni, stored procedures e query che li richiamano siano esattamente gli stessi.

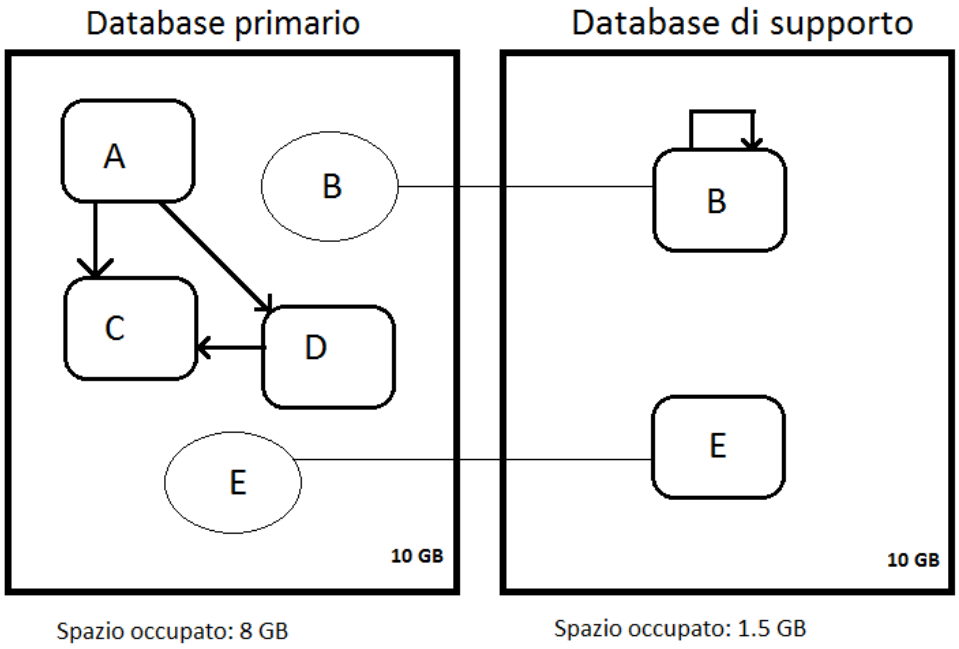

Figura 11

5.1.4 Integrazione alla patch

L'installazione al cliente delle nuove implementazioni avviene eseguendo la patch di aggiornamento del database (o dei database): un campo nella tabella impostazioni indica se al database ne è associato uno di supporto e, se sì, alla fine

dell'eseguibile viene accorpato il nuovo script che comporta le nuove modifiche sui dati appena aggiornati.

Prima volta

Il cliente possiede un database con una mole di dati prossima ai dieci gigabyte e gli deve essere installata la nuova implementazione.
Viene quindi creato il database di supporto e avviata la patch che, una volta aggiornato il database, avvia lo script che esegue lo split delle tabelle e la creazione dei sinonimi.

Volte successive

Nel caso in cui al cliente sia già stato installato il database di supporto e in passato era già stato eseguito lo script almeno una volta, lo script possiede delle funzionalità di aggiornamento, cioè si occupa di controllare e aggiornare delle tabelle che potrebbero essere presenti nel database primario solo come sinonimi e le aggiorna quindi dal database di supporto.

La tabella gettonata è [dbo].[Accessi], a cui vengono aggiornati, uno per uno, indici e trigger.

Esempio:

```
IF EXISTS( SELECT * FROM dbo.sysobjects WHERE id =
OBJECT_ID(N'[dbo].[Accessi]') AND type = 'U')
    BEGIN
        IF NOT EXISTS ( SELECT name FROM sysindexes WHERE name =
'IX_Accessi_1')
            BEGIN
                CREATE NONCLUSTERED INDEX IX_Accessi_1 ON
dbo.Accessi(IDUtente ASC, TipoAccesso ASC, IDServizio ASC, IDAccesso
ASC) INCLUDE(DataAccesso) WITH(SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY =
OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY];
            END;
    END;
IF NOT EXISTS( SELECT * FROM dbo.sysobjects WHERE id =
OBJECT_ID(N'[dbo].[Accessi]') AND type = 'U')
    BEGIN
        EXEC ('use '+@ndbsupp+';
         if exists (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
         begin
```

```
              IF NOT EXISTS (SELECT name FROM sysindexes WHERE name =
''IX_Accessi_1'')
              begin
                   CREATE NONCLUSTERED INDEX [IX_Accessi_1] ON
[dbo].[Accessi] (
                  [IDUtente] ASC,
                  [TipoAccesso] ASC,
                  [IDServizio] ASC,
                  [IDAccesso] ASC
                  )
                  INCLUDE ( [DataAccesso]) WITH (SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
              END
         END');
    END;
```

[dbo].[Accessi] è una tabella che quasi sicuramente viene selezionata per essere trasferita nel database di supporto. Viene quindi controllato in quale database (primario o di supporto) si trovi la tabella e viene creato se non esiste l'indice IX_Accessi_1.

6. CONCLUSIONI E CONSIDERAZIONI

Alla fine dell'operazione, il gestionale Inforyou Manager sarà collegato allo stesso identico database di prima, le cui informazioni ora sono in parte contenuta nel database di supporto.

Il cliente, adoperando il gestionale, non conosce (e non è suo interesse conoscere) le modifiche apportate, in quanto le istruzioni di utilizzo non sono state alterate. Si accorge però di alcuni cambiamenti in termini di prestazioni: il software ha tempi di esecuzione migliori e può immagazzinare più informazioni. Questo è dovuto al fatto che tabelle con grande spazio di archiviazione ("Accessi" è la tabella che occupa più memoria) sono state rimpiazzate da oggetti di tipo sinonimo.

Il progetto è stato implementato con procedure e istruzioni consone all'adattamento al modello di database dell'azienda, ma, con l'idea di fondo, può essere pensato e sviluppato per qualsiasi database di altre aziende.

Il lavoro è stato implementato rispettando una scadenza e adatto a risolvere le esigenze richieste, ma è comunque possibile effettuare altre migliorie allo script: per esempio, l'aggiunta di stored procedures nello script, che rimpiazzerebbero blocchi di codice e migliorerebbero le prestazioni temporali.

7. SCRIPT

```
DECLARE @ndb NVARCHAR(MAX)= DB_NAME();

-------------lettura nome del database di supporto ..
DECLARE @ndbsupp NVARCHAR(MAX);

IF NOT EXISTS (SELECT * FROM impostazioni WHERE Descrizione =
'NomeDBSupporto')
    BEGIN
      INSERT INTO dbo.impostazioni (Descrizione,Valore) VALUES
      ('NomeDBSupporto', @ndb+'_supporto');
    END;

IF EXISTS(SELECT * FROM impostazioni WHERE Descrizione =
'NomeDBSupporto')
    BEGIN
       SET @ndbsupp =( SELECT Valore FROM dbo.impostazioni WHERE
      Descrizione = 'NomeDBSupporto');
    END;
-------------------------------------------------------------------------

--------------------------Tabelle temporanee che serviranno per lo split
CREATE TABLE #tab_senza_fk(nome NVARCHAR(MAX));

CREATE TABLE #COLS(nome_tab NVARCHAR(MAX),nome_col NVARCHAR(MAX));

CREATE TABLE #tcols(ntab     NVARCHAR(MAX), listcols NVARCHAR(MAX));

CREATE TABLE #temp_col(nomecol NVARCHAR(MAX));

CREATE TABLE #tab_ins(tab NVARCHAR(MAX));

CREATE TABLE #tab_triggers(TriggerName NVARCHAR(MAX), SqlContent
NVARCHAR(MAX));

CREATE TABLE #tab_mem(nome NVARCHAR(MAX), nrows INT, reserved
NVARCHAR(MAX), datasize NVARCHAR(MAX), index_size NVARCHAR(MAX), unused
NVARCHAR(MAX));

CREATE TABLE #tabnofksize(nome NVARCHAR(MAX), size INT);

CREATE TABLE #tab_tomove(nome NVARCHAR(MAX));

CREATE TABLE #scripts(nome NVARCHAR(MAX), script NVARCHAR(MAX));
-------------------------------------------------------------------------

-----------------------------------------------popolo tabelle temporanee
DELETE FROM #tab_senza_fk;
INSERT INTO #tab_senza_fk
SELECT name FROM sys.tables AS t WHERE t.name <> 'dtproperties' AND
t.name <> 'sysdiagrams' AND t.object_id NOT IN
(SELECT parent_object_id FROM sys.foreign_keys) AND t.object_id NOT IN (
SELECT referenced_object_id FROM sys.foreign_keys)
```

```
INSERT INTO #tab_senza_fk
SELECT name FROM sys.tables AS t WHERE t.name <> 'dtproperties' AND
t.name <> 'sysdiagrams' AND t.object_id IN
(SELECT parent_object_id FROM sys.foreign_keys WHERE
parent_object_id=referenced_object_id) AND t.object_id NOT IN
(SELECT parent_object_id FROM sys.foreign_keys WHERE
parent_object_id!=referenced_object_id) AND t.object_id NOT IN
(SELECT referenced_object_id FROM sys.foreign_keys WHERE
parent_object_id!=referenced_object_id)

--tengo AccessiSettimana e TicketSconti nel db originale per scelta
personale

IF EXISTS( SELECT * FROM #tab_senza_fk WHERE nome = 'AccessiSettimana')
    BEGIN
        DELETE FROM #tab_senza_fk WHERE nome = 'AccessiSettimana';
    END;

IF EXISTS( SELECT * FROM #tab_senza_fk WHERE nome = 'TicketSconti')
    BEGIN
        DELETE FROM #tab_senza_fk WHERE nome = 'TicketSconti';
    END;


DELETE FROM #tab_mem;
INSERT INTO #tab_mem
EXEC sp_msForEachTable 'EXEC sp_SpaceUsed ''?''';

declare @n nvarchar(max) = (select top 1 nome from #tab_mem where nome
like '%dbo%')
declare @n2 nvarchar(max)

while @n is not null
begin
    select @n2 = (replace(replace(@n,'[dbo].[',''),']',''))
    exec('update #tab_mem set nome = ''' + @n2 + ''' where nome = ''' +
@n + '''');

    select @n = (select top 1 nome from #tab_mem where nome like
'%dbo%')
end

DECLARE @totalekbtab INT;

SELECT @totalekbtab =( SELECT SUM(CONVERT( INT, REPLACE(reserved, ' KB',
'')) ) FROM #tab_mem);

DELETE FROM #tab_mem WHERE nome NOT IN ( SELECT nome FROM
#tab_senza_fk);

DELETE FROM #tabnofksize;
INSERT INTO #tabnofksize
SELECT nome, CONVERT( INT, REPLACE(reserved, ' KB', '')) AS sizeKB FROM
#tab_mem WHERE nome IN
( SELECT nome FROM #tab_senza_fk )
 ORDER BY sizeKB DESC;
```

```
--tot kb in tabelle
--quindi trasporto fino a @totalekbtab/2

DECLARE @contatore INT;
DECLARE @ntabelle INT;
DECLARE @pesotab INT;
DECLARE @peso INT;

SELECT @ntabelle = ( SELECT COUNT(*) FROM #tab_senza_fk);

SELECT @contatore = 1;
SELECT @peso = 0;

WHILE @contatore <= @ntabelle
    BEGIN
      SELECT @pesotab = ( SELECT size FROM ( SELECT ROW_NUMBER()
      OVER(ORDER BY nome) AS nrow, nome AS nom, CONVERT( INT,
      REPLACE(reserved, 'KB', '')) AS size FROM #tab_mem ) AS t WHERE
      nrow = @contatore);

        IF ((@peso + @pesotab) < @totalekbtab / 2) AND @pesotab > 0
            BEGIN
                SELECT @peso = @peso + @pesotab;
                DECLARE @nt NVARCHAR(100)=( SELECT nom FROM ( SELECT
ROW_NUMBER() OVER(ORDER BY nome) AS nrow, nome AS nom,
                                                    CONVERT( INT,
REPLACE(reserved, 'KB', '')) AS size FROM #tab_mem ) AS t WHERE nrow =
@contatore);
                INSERT INTO #tab_tomove VALUES(@nt);
            END;

        SELECT @contatore = @contatore + 1;
    END;

DELETE FROM #COLS;
INSERT INTO #COLS
SELECT TABLE_NAME, COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME IN ( SELECT nome FROM #tab_tomove) ORDER BY TABLE_NAME;

DELETE FROM #tab_ins;
INSERT INTO #tab_ins
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME IN ( SELECT nome FROM #tab_tomove) AND
COLUMNPROPERTY(OBJECT_ID(TABLE_NAME), COLUMN_NAME, 'IsIdentity') = 1
ORDER BY TABLE_NAME;

DELETE FROM #tab_triggers;
INSERT INTO #tab_triggers
       SELECT dbo.SysObjects.Name AS TriggerName,
              dbo.sysComments.Text AS SqlContent
       FROM dbo.SysObjects
            INNER JOIN dbo.sysComments ON dbo.SysObjects.ID =
dbo.sysComments.ID
       WHERE dbo.SysObjects.xType = 'TR'
             AND dbo.SysObjects.Name IN
       (
           SELECT sys.triggers.name
```

```
            FROM sys.triggers
                 INNER JOIN sys.tables ON sys.triggers.parent_id =
sys.tables.object_id
            WHERE sys.tables.name IN
            ( SELECT nome FROM #tab_tomove )
       );

DECLARE @table NVARCHAR(MAX);
SELECT @table =( SELECT TOP 1 nome FROM #tab_tomove WHERE nome NOT IN(
SELECT ntab FROM #tcols ));

WHILE @table IS NOT NULL
    BEGIN
        DECLARE @listacolonne NVARCHAR(MAX);
        DECLARE @col NVARCHAR(MAX);
        DELETE FROM #temp_col;
        SELECT @col =( SELECT TOP (1) nome_col FROM #COLS WHERE nome_tab
= @table AND nome_col NOT IN (
                SELECT * FROM #temp_col ));

        INSERT INTO #temp_col VALUES(@col);

        SELECT @listacolonne = @col;
        SELECT @col =( SELECT TOP (1) nome_col FROM #COLS WHERE nome_tab
= @table AND nome_col NOT IN
            ( SELECT * FROM #temp_col )
        );

        WHILE @col IS NOT NULL
            BEGIN
                SELECT @listacolonne = @listacolonne+', '+@col;
                INSERT INTO #temp_col
                VALUES(@col);
                SELECT @col = ( SELECT TOP (1) nome_col FROM #COLS WHERE
nome_tab = @table AND nome_col NOT IN
                    ( SELECT * FROM #temp_col)
                );
            END;

        INSERT INTO #tcols VALUES (@table, @listacolonne);

        SELECT @table = ( SELECT TOP 1 nome FROM #tab_tomove WHERE nome
NOT IN( SELECT ntab FROM #tcols));
    END;
-------------------------------------------------------------------------

-------------------------------------creazione tabelle nel db supporto
DECLARE @tabToCopy TABLE(nome NVARCHAR(MAX));
INSERT INTO @tabToCopy
EXEC ('select top(1) nome from #tab_tomove where nome not in (select
name from ['+@ndbsupp+'].sys.tables) order by nome');

DECLARE @tab_copy NVARCHAR(MAX);
SELECT @tab_copy =( SELECT * FROM @tabToCopy);
DELETE FROM @tabToCopy;

while @tab_copy is not null
```

```
begin

       DECLARE @table_name nvarchar(max)
       SELECT @table_name = 'dbo.'+rtrim(@tab_copy)

DECLARE
      @object_name nvarchar(max)
    , @object_id INT

SELECT
      @object_name = '[' + s.name + '].[' + o.name + ']'
    , @object_id = o.[object_id]
FROM sys.objects o WITH (NOWAIT)
JOIN sys.schemas s WITH (NOWAIT) ON o.[schema_id] = s.[schema_id]
WHERE s.name + '.' + o.name = @table_name
    AND o.[type] = 'U'
    AND o.is_ms_shipped = 0

DECLARE @SQL NVARCHAR(MAX) = ''

;WITH index_column AS
(
    SELECT
          ic.[object_id]
        , ic.index_id
        , ic.is_descending_key
        , ic.is_included_column
        , c.name
    FROM sys.index_columns ic WITH (NOWAIT)
    JOIN sys.columns c WITH (NOWAIT) ON ic.[object_id] = c.[object_id]
AND ic.column_id = c.column_id
    WHERE ic.[object_id] = @object_id
),
fk_columns AS
(
     SELECT
          k.constraint_object_id
        , cname = c.name
        , rcname = rc.name
    FROM sys.foreign_key_columns k WITH (NOWAIT)
    JOIN sys.columns rc WITH (NOWAIT) ON rc.[object_id] =
k.referenced_object_id AND rc.column_id = k.referenced_column_id
    JOIN sys.columns c WITH (NOWAIT) ON c.[object_id] =
k.parent_object_id AND c.column_id = k.parent_column_id
    WHERE k.parent_object_id = @object_id
)
SELECT @SQL = 'CREATE TABLE ' + @object_name + CHAR(13) + '(' + CHAR(13)
+ STUFF((
    SELECT CHAR(9) + ', [' + c.name + '] ' +
        CASE WHEN c.is_computed = 1
            THEN 'AS ' + cc.[definition]
            ELSE UPPER(tp.name) +
                CASE WHEN tp.name IN ('varchar', 'char', 'varbinary',
'binary')
                       THEN '(' + CASE WHEN c.max_length = -1 THEN 'MAX'
ELSE CAST(c.max_length AS VARCHAR(5)) END + ')'
                     WHEN tp.name IN ('nvarchar', 'nchar')
```

```
                        THEN '(' + CASE WHEN c.max_length = -1 THEN 'MAX'
ELSE CAST(c.max_length / 2 AS VARCHAR(5)) END + ')'
                    WHEN tp.name IN ('datetime2', 'time2',
'datetimeoffset')
                        THEN '(' + CAST(c.scale AS VARCHAR(5)) + ')'
                    WHEN tp.name = 'decimal'
                        THEN '(' + CAST(c.[precision] AS VARCHAR(5)) +
',' + CAST(c.scale AS VARCHAR(5)) + ')'
                    ELSE ''
                END +
                CASE WHEN c.collation_name IS NOT NULL THEN ' COLLATE '
+ c.collation_name ELSE '' END +
                CASE WHEN c.is_nullable = 1 THEN ' NULL' ELSE ' NOT
NULL' END +
                CASE WHEN dc.[definition] IS NOT NULL THEN ' DEFAULT' +
dc.[definition] ELSE '' END +
                CASE WHEN ic.is_identity = 1 THEN ' IDENTITY(' +
CAST(ISNULL(ic.seed_value, '0') AS CHAR(1)) +
                                    ',' +
CAST(ISNULL(ic.increment_value, '1') AS CHAR(1)) + ')' ELSE '' END
        END + CHAR(13)
    FROM sys.columns c WITH (NOWAIT)
    JOIN sys.types tp WITH (NOWAIT) ON c.user_type_id = tp.user_type_id
    LEFT JOIN sys.computed_columns cc WITH (NOWAIT) ON c.[object_id] =
cc.[object_id] AND c.column_id = cc.column_id
    LEFT JOIN sys.default_constraints dc WITH (NOWAIT) ON
c.default_object_id != 0 AND c.[object_id] = dc.parent_object_id AND
c.column_id = dc.parent_column_id
    LEFT JOIN sys.identity_columns ic WITH (NOWAIT) ON c.is_identity = 1
AND c.[object_id] = ic.[object_id] AND c.column_id = ic.column_id
    WHERE c.[object_id] = @object_id
    ORDER BY c.column_id
    FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1, 2, CHAR(9) +
' ')
    + ISNULL((SELECT CHAR(9) + ', CONSTRAINT [' + k.name + '] PRIMARY
KEY (' +
                    (SELECT STUFF((
                         SELECT ', [' + c.name + '] ' + CASE WHEN
ic.is_descending_key = 1 THEN 'DESC' ELSE 'ASC' END
                         FROM sys.index_columns ic WITH (NOWAIT)
                         JOIN sys.columns c WITH (NOWAIT) ON
c.[object_id] = ic.[object_id] AND c.column_id = ic.column_id
                         WHERE ic.is_included_column = 0
                             AND ic.[object_id] = k.parent_object_id
                             AND ic.index_id = k.unique_index_id
                         FOR XML PATH(N''), TYPE).value('.',
'NVARCHAR(MAX)'), 1, 2, ''))
            + ')' + CHAR(13)
            FROM sys.key_constraints k WITH (NOWAIT)
            WHERE k.parent_object_id = @object_id
                AND k.[type] = 'PK'), '') + ')'  + CHAR(13)
    + ISNULL((SELECT (
        SELECT CHAR(13) +
             'ALTER TABLE ' + @object_name + ' WITH'
            + CASE WHEN fk.is_not_trusted = 1
                THEN ' NOCHECK'
                ELSE ' CHECK'
```

```
            END +
            ' ADD CONSTRAINT [' + fk.name  + '] FOREIGN KEY('
            + STUFF((
              SELECT ', [' + k.cname + ']'
              FROM fk_columns k
              WHERE k.constraint_object_id = fk.[object_id]
              FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1,
2, '')
             + ')' +
            ' REFERENCES [' + SCHEMA_NAME(ro.[schema_id]) + '].[' +
ro.name + '] ('
            + STUFF((
              SELECT ', [' + k.rcname + ']'
              FROM fk_columns k
              WHERE k.constraint_object_id = fk.[object_id]
              FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1,
2, '')
             + ')'
          + CASE
              WHEN fk.delete_referential_action = 1 THEN ' ON DELETE
CASCADE'
              WHEN fk.delete_referential_action = 2 THEN ' ON DELETE
SET NULL'
              WHEN fk.delete_referential_action = 3 THEN ' ON DELETE
SET DEFAULT'
              ELSE ''
            END
          + CASE
              WHEN fk.update_referential_action = 1 THEN ' ON UPDATE
CASCADE'
              WHEN fk.update_referential_action = 2 THEN ' ON UPDATE
SET NULL'
              WHEN fk.update_referential_action = 3 THEN ' ON UPDATE
SET DEFAULT'
              ELSE ''
            END
          + CHAR(13) + 'ALTER TABLE ' + @object_name + ' CHECK
CONSTRAINT [' + fk.name  + ']' + CHAR(13)
        FROM sys.foreign_keys fk WITH (NOWAIT)
        JOIN sys.objects ro WITH (NOWAIT) ON ro.[object_id] =
fk.referenced_object_id
        WHERE fk.parent_object_id = @object_id
        FOR XML PATH(N''), TYPE).value('.', 'NVARCHAR(MAX)')), '')
    + ISNULL(((SELECT
         CHAR(13) + 'CREATE' + CASE WHEN i.is_unique = 1 THEN ' UNIQUE'
ELSE '' END
              + ' NONCLUSTERED INDEX [' + i.name + '] ON ' +
@object_name + ' (' +
              STUFF((
              SELECT ', [' + c.name + ']' + CASE WHEN
c.is_descending_key = 1 THEN ' DESC' ELSE ' ASC' END
              FROM index_column c
              WHERE c.is_included_column = 0
                  AND c.index_id = i.index_id
              FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1,
2, '') + ')'
              + ISNULL(CHAR(13) + 'INCLUDE (' +
```

```
                    STUFF((
                    SELECT ', [' + c.name + ']'
                    FROM index_column c
                    WHERE c.is_included_column = 1
                        AND c.index_id = i.index_id
                    FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'),
1, 2, '') + ')', '')  + CHAR(13)
        FROM sys.indexes i WITH (NOWAIT)
        WHERE i.[object_id] = @object_id
            AND i.is_primary_key = 0
            AND i.[type] = 2
        FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)')
    ), '')

        INSERT INTO #scripts (nome, script) VALUES (@tab_copy, @SQL );

        ---crea e popola tabella
        EXEC ('use '+@ndbsupp+';
                    declare @scr nvarchar(max)
                    select @scr = (select script from #scripts where
nome = '''+@tab_copy+''')
                    exec(@scr)

                    exec sp_rename '''+@tab_copy+''' ,
''temp_identity''

                    create synonym sintab for
['+@ndb+'].[dbo].['+@tab_copy+']

                    if exists (select * from #tab_ins where tab =
'''+@tab_copy+''')
                    begin
                        SET IDENTITY_INSERT dbo.temp_identity ON
                    end

                    declare @listcols as nvarchar(max) = (select
listcols from #tcols where ntab = '''+@tab_copy+''')

                    exec(''INSERT INTO temp_identity('' + @listcols +
'') SELECT '' + @listcols + '' FROM sintab'')


                    if exists (select * from #tab_ins where tab =
'''+@tab_copy+''')
                    begin
                        SET IDENTITY_INSERT dbo.temp_identity OFF
                    end

                    DROP synonym sintab

                    Exec sp_rename ''temp_identity'',
'''+@tab_copy+''''');

        ---leggi prossima tab
        INSERT INTO @tabToCopy
        EXEC ('select top(1) nome from #tab_tomove where nome not in
(select name from ['+@ndbsupp+'].sys.tables) order by nome');
```

```
        SELECT @tab_copy = ( SELECT * FROM @tabToCopy );
        DELETE FROM @tabToCopy;
        -----
    END;
-------------------------------------------------------------------------

------------------------------------------------------ creo triggers
EXEC ('use '+@ndbsupp+';

declare @trig nvarchar(max) = (select top 1 TriggerName from
#tab_triggers)
declare @sqltrig nvarchar(max) = (select top 1 SqlContent from
#tab_triggers)

while @trig is not null
begin
      exec (@sqltrig)

      exec(''delete from #tab_triggers where TriggerName = '''''' +
@trig + '''''''')

      select @trig = (select top 1 TriggerName from #tab_triggers)
      select @sqltrig = (select top 1 SqlContent from #tab_triggers)
end');
-------------------------------------------------------------------------
-----------------------------------------Elimino tabelle splittate
DECLARE @tab_nofk NVARCHAR(MAX)=( SELECT TOP 1 * FROM #tab_tomove WHERE
nome IN ( SELECT name FROM sys.tables));

DECLARE @q_nofk NVARCHAR(MAX);
WHILE @tab_nofk IS NOT NULL
    BEGIN
        SELECT @q_nofk = 'drop table '+RTRIM(@tab_nofk)+';';
        EXEC (@q_nofk);
        SELECT @tab_nofk = ( SELECT TOP (1) * FROM #tab_tomove WHERE
nome IN ( SELECT name FROM sys.tables ));
    END;
-------------------------------------------------------------------------

-----------------------------------CONTROLLO INDICI Accessi (da patch)
IF EXISTS ( SELECT * FROM dbo.sysobjects WHERE id=
OBJECT_ID(N'[dbo].[Accessi]') AND type ='U' )
begin
    IF NOT EXISTS( SELECT name FROM sysindexes WHERE name =
'IX_Acc_DataAccesso')
         BEGIN
              CREATE INDEX IX_Acc_DataAccesso ON Accessi(DataAccesso);
         END;
end
IF NOT EXISTS ( SELECT * FROM dbo.sysobjects WHERE id=
OBJECT_ID(N'[dbo].[Accessi]') AND type ='U' )
BEGIN
     EXEC ('use '+@ndbsupp+';
      if EXISTS (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
      begin
```

```
            if not exists (SELECT name FROM sysindexes WHERE name =
''IX_Acc_DataAccesso'')
            begin

              CREATE INDEX IX_Acc_DataAccesso ON Accessi(DataAccesso);
            END
      END');
END;

IF EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        IF NOT EXISTS
        (
            SELECT name
            FROM sysindexes
            WHERE name = 'IX_AccessiPerZona'
        )
            BEGIN

                /* Accessi Zona*/
                CREATE NONCLUSTERED INDEX IX_AccessiPerZona ON
dbo.Accessi
                (IDFamigliaTerminale ASC, Indirizzo ASC, IDConcentratore
ASC, TipoAccesso ASC, Tessera ASC
                ) INCLUDE(IDAccesso, IDUtente, IDServizio, DataAccesso,
NomeUtente, TipoServizio, Descrizione, IDAutomazione, IDIscrizione)
                      WITH(SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY];
            END;
    END;
IF NOT EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        EXEC ('use '+ @ndbsupp+';
      IF EXISTS (SELECT * FROM dbo.sysobjects WHERE id =
OBJECT_ID(N''[dbo].[Accessi]'') AND type =''U'')
      begin
            IF NOT EXISTS (SELECT name FROM sysindexes WHERE
name=''IX_AccessiPerZona'')
            BEGIN


                  /* Accessi Zona*/
                  CREATE NONCLUSTERED INDEX [IX_AccessiPerZona] ON
[dbo].[Accessi]
                  (
```

```
                    [IDFamigliaTerminale] ASC,
                    [Indirizzo] ASC,
                    [IDConcentratore] ASC,
                    [TipoAccesso] ASC,
                    [Tessera] ASC
                    )
                    INCLUDE ( [IDAccesso],
                    [IDUtente],
                    [IDServizio],
                    [DataAccesso],
                    [NomeUtente],
                    [TipoServizio],
                    [Descrizione],
                    [IDAutomazione],
                    [IDIscrizione]) WITH (SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]


             END
      END');
    END;

IF EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        IF NOT EXISTS
        (
            SELECT name
            FROM sysindexes
            WHERE name = 'IX_AccessiPrimoAccesso'
        )
            BEGIN
                CREATE NONCLUSTERED INDEX IX_AccessiPrimoAccesso ON
dbo.Accessi(DataAccesso DESC, IDFamigliaTerminale ASC, IDAccesso ASC,
                    Indirizzo ASC, IDUtente ASC) WITH(SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON
[PRIMARY];
            END;
    END;
IF NOT EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        EXEC ('use '+@ndbsupp+';
      IF EXISTS (SELECT * FROM dbo.sysobjects WHERE id =
OBJECT_ID(N''[dbo].[Accessi]'') AND type =''U'')
      begin
```

```
            IF NOT EXISTS (SELECT name FROM sysindexes WHERE
name=''IX_AccessiPrimoAccesso'')
            BEGIN

            CREATE NONCLUSTERED INDEX [IX_AccessiPrimoAccesso] ON
[dbo].[Accessi]
            (
                  [DataAccesso] DESC,
                  [IDFamigliaTerminale] ASC,
                  [IDAccesso] ASC,
                  [Indirizzo] ASC,
                  [IDUtente] ASC
            )WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

            END
      END');
    END;

IF EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        IF NOT EXISTS
        (
            SELECT name
            FROM sysindexes
            WHERE name = 'IX_Accessi_1'
        )
            BEGIN
                CREATE NONCLUSTERED INDEX IX_Accessi_1 ON
dbo.Accessi(IDUtente ASC, TipoAccesso ASC, IDServizio ASC, IDAccesso
ASC) INCLUDE(DataAccesso) WITH(SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY =
OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY];
            END;
    END;
IF NOT EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        EXEC ('use '+@ndbsupp+';
      if exists (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
      begin
            IF NOT EXISTS (SELECT name FROM sysindexes WHERE name =
''IX_Accessi_1'')
            begin
                  CREATE NONCLUSTERED INDEX [IX_Accessi_1] ON
[dbo].[Accessi]
```

```
                    (
                    [IDUtente] ASC,
                    [TipoAccesso] ASC,
                    [IDServizio] ASC,
                    [IDAccesso] ASC
                    )
                    INCLUDE ( [DataAccesso]) WITH (SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

              END
       END');
    END;

IF EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        IF NOT EXISTS
        (
            SELECT name
            FROM sysindexes
            WHERE name = '_IX_Accessi_IDIscrizione'
        )
            BEGIN
                CREATE NONCLUSTERED INDEX _IX_Accessi_IDIscrizione ON
dbo.Accessi(IDIscrizione ASC, DataAccesso ASC) WITH(SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON
[PRIMARY];
            END;
    END;
IF NOT EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        EXEC ('use '+@ndbsupp+';
       if exists (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
       begin
              IF not EXISTS (SELECT name FROM sysindexes WHERE name =
''_IX_Accessi_IDIscrizione'')
              BEGIN
                    CREATE NONCLUSTERED INDEX _IX_Accessi_IDIscrizione
ON [dbo].[Accessi]
                    (
                          [IDIscrizione] ASC,
                          [DataAccesso] ASC
                    )WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
              END
```

```
      END');
    END;

IF EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        IF NOT EXISTS
        (
            SELECT Name
            FROM sysindexes
            WHERE id =
            (
                SELECT OBJECT_ID('[Accessi]')
            )
                  AND name = 'IX_Accessi_Tessera'
        )
            BEGIN
                CREATE NONCLUSTERED INDEX IX_Accessi_Tessera ON
dbo.Accessi(Tessera ASC) INCLUDE(IDAccesso) WITH(SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY];
            END;
    END;
IF NOT EXISTS
(
    SELECT *
    FROM dbo.sysobjects
    WHERE id = OBJECT_ID(N'[dbo].[Accessi]')
          AND type = 'U'
)
    BEGIN
        EXEC ('use '+@ndbsupp+';
      if exists (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
      begin
            IF not exists (SELECT Name FROM sysindexes   WHERE id =
(SELECT OBJECT_ID(''[Accessi]'')) AND name = ''IX_Accessi_Tessera'')
            BEGIN

                  CREATE NONCLUSTERED INDEX [IX_Accessi_Tessera] ON
[dbo].[Accessi]
                  (
                        [Tessera] ASC
                  )
                  INCLUDE ( [IDAccesso]) WITH (SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

            END
      END');
    END;

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[Accessi]') and type ='U')
```

```
begin
IF COL_LENGTH('dbo.Accessi','TGSincronizzato') IS NULL
BEGIN
       ALTER TABLE Accessi
       ADD TGSincronizzato BIT NOT NULL
       CONSTRAINT DF_Accessi_TGSincronizzato DEFAULT 0
END
END

IF NOT EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[Accessi]') and type ='U')
BEGIN
    EXEC('use ' + @ndbsupp + ';
         if exists (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
         begin
              IF COL_LENGTH(''dbo.Accessi'',''TGSincronizzato'') IS
NULL
              BEGIN
                   ALTER TABLE Accessi
                   ADD TGSincronizzato BIT NOT NULL
                   CONSTRAINT DF_Accessi_TGSincronizzato DEFAULT 0
              END
         END')
END
-------------------------------------------------------------------------

------------------------CONTROLLO TRIGGER IN TABELLA ACCESSI (da patch)
IF EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[Accessi]') and type ='U')
begin
    IF EXISTS ( select * from dbo.sysobjects where name =
'tr_Accessi_NotifyAccessQueue')
    BEGIN
         DROP TRIGGER tr_Accessi_NotifyAccessQueue;
    END

    EXEC('
    CREATE TRIGGER dbo.tr_Accessi_NotifyAccessQueue
         ON  dbo.Accessi
             AFTER INSERT
         AS
         BEGIN
               SET NOCOUNT ON;

              DECLARE @Message VARCHAR(MAX)
               SET @Message = ''<ChangedRows>'';

               SELECT @Message = @Message +
                     ISNULL(''<Inserted IDAccesso="'' +
CAST(ISNULL(IDAccesso, 0) AS VARCHAR) +
                     ''" IDAzienda="'' + CAST(ISNULL(IDAzienda, 0) AS
VARCHAR) +''" />'', '''')
               FROM
                     Inserted t0 INNER JOIN Terminali t1
```

```
                    ON t0.IDConcentratore = t1.IDConcentratore AND
t0.IDFamigliaTerminale = t1.IDFamigliaTerminale AND t0.Indirizzo =
t1.Indirizzo
                SET @Message = @Message + ''</ChangedRows>''
                DECLARE @DialogHandle UNIQUEIDENTIFIER
               BEGIN DIALOG @DialogHandle
               FROM SERVICE [IYAccessNotifications]
               TO SERVICE ''IYAccessNotifications''
               ON CONTRACT
[http://schemas.microsoft.com/SQL/Notifications/PostQueryNotification]
               WITH ENCRYPTION = OFF, LIFETIME = 30;
               SEND ON CONVERSATION @DialogHandle
               MESSAGE TYPE
[http://schemas.microsoft.com/SQL/Notifications/QueryNotification]
(@Message);
          END')
end


IF NOT EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[Accessi]') and type ='U')
BEGIN

    EXEC('
    use ' + @ndbsupp + ';
    IF EXISTS (select * from dbo.sysobjects where id =
object_id(N''[dbo].[Accessi]'') and type =''U'')
    begin
          IF EXISTS ( select * from dbo.sysobjects where name =
''tr_Accessi_NotifyAccessQueue'')
          BEGIN
               DROP TRIGGER tr_Accessi_NotifyAccessQueue;
          END

          EXEC(''
          CREATE TRIGGER dbo.tr_Accessi_NotifyAccessQueue
               ON  dbo.Accessi
                   AFTER INSERT
               AS
               BEGIN
                     SET NOCOUNT ON;

                    DECLARE @Message VARCHAR(MAX)
                     SET @Message = ''''<ChangedRows>'''';

                     SELECT @Message = @Message +
                           ISNULL(''''<Inserted IDAccesso="''''' +
CAST(ISNULL(IDAccesso, 0) AS VARCHAR) +
                           ''''" IDAzienda="''''' +
CAST(ISNULL(IDAzienda, 0) AS VARCHAR) +''''" />'''', '''''''')
                     FROM
                           Inserted t0 INNER JOIN Terminali t1
                           ON t0.IDConcentratore = t1.IDConcentratore
AND t0.IDFamigliaTerminale = t1.IDFamigliaTerminale AND t0.Indirizzo =
t1.Indirizzo
                     SET @Message = @Message + ''''</ChangedRows>''''
                     DECLARE @DialogHandle UNIQUEIDENTIFIER
```

```
                    BEGIN DIALOG @DialogHandle
                    FROM SERVICE [IYAccessNotifications]
                    TO SERVICE ''''IYAccessNotifications''''
                    ON CONTRACT
[http://schemas.microsoft.com/SQL/Notifications/PostQueryNotification]
                    WITH ENCRYPTION = OFF, LIFETIME = 30;
                    SEND ON CONVERSATION @DialogHandle
                    MESSAGE TYPE
[http://schemas.microsoft.com/SQL/Notifications/QueryNotification]
(@Message);
                END'')
    end');
END
-------------------------------------------------------------------------

------------------------------ In db, creo sinonimi delle tab_senza_fk

DECLARE @tab_sin_nofk NVARCHAR(MAX)=( SELECT TOP (1) nome FROM
#tab_tomove WHERE nome NOT IN( SELECT name FROM sys.synonyms ));

DECLARE @q_sin_nofk NVARCHAR(MAX);
WHILE @tab_sin_nofk IS NOT NULL
    BEGIN
        SELECT @q_sin_nofk = 'create synonym '+RTRIM(@tab_sin_nofk)+'
for ['+@ndbsupp+'].[dbo].['+RTRIM(@tab_sin_nofk)+']';
        EXEC (@q_sin_nofk);
        SELECT @tab_sin_nofk = ( SELECT TOP (1) nome FROM #tab_tomove
WHERE nome NOT IN ( SELECT name FROM sys.synonyms ));
    END;
```