UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA
Dipartimento di Ingegneria “Enzo Ferrari”

Corso di laurea in Ingegneria Informatica

TimescaleDB: analisi di time-series di consumi energetici

Relatore

Prof.ssa Sonia Bergamaschi

Correlatore

Phd Luca Gagliardelli

Candidata

LUISA BOTTIGLIERI

Anno Accademico 2019/2020

Grazie, ai miei genitori e alle mie sorelle perché mi incoraggiano.

Grazie, a Mattia perché mi sostiene con i fatti.

Indice

1 Introduzione

L'analisi dei consumi energetici è, oggi, fondamentale per definire i profili differenti di produzione e di consumo dei singoli utenti, in modo da poter individuare modelli di sostenibilità energetica volti ad ottimizzare i consumi delle singole comunità di utenti. Analizzando i consumi, si possono comprendere quali provvedimenti attuare per ottimizzarli, calcolando modelli che visualizzino la quantità di energia impiegata lungo intervalli temporali precisi, ad esempio per settimana, per mese o per anno, e studiando le caratteristiche dell'ambiente in cui avvengono i consumi, ovvero considerare la presenza o meno di fonti dispersive di calore, come infissi di porte finestre non adeguate all'isolamento termico, o l'installazione di elettrodomestici troppo dispendiosi di energia elettrica.

L'obiettivo di questa tesi è proporre un modello di definizione e di analisi dei dati di consumo energetico per tracciare i profili di consumo delle comunità, attraverso l'applicazione delle funzionalità del DBMS TimescaleDB. I dati di consumo su cui il progetto si svolge sono contenuti in alcuni file forniti da un ente attivo nei settori dell'energia, che ha messo a disposizione al gruppo di ricerca "DB Group" (https://dbgroup.unimore.it/site/home.html) alcuni file sorgenti di esempio per poter sviluppare dei modelli. La particolarità di questi file è il contenuto di time-series data, tipologie di dati che presentano sempre la data e/o l'ora in cui sono stati registrati.

Il capitolo 2 introdurrà TimescaleDB, un DBMS specializzato nell'uso dei time-series data, e ne approfondirà le funzioni principali. Seguirà, nel capitolo 3, la progettazione di un database relativo ai consumi energetici tramite la progettazione del modello E/R e l'implementazione in relazionale con il linguaggio SQL. Nel capitolo 4 si affronterà l'inserimento dei dati contenuti nei file esemplificativi. Il capitolo 5 presenterà alcune interrogazioni sui dati per effettuare alcune analisi sui profili dei consumi, applicando le funzionalità principali di TimescaleDB. Nelle conclusioni, infine, verrà spiegato come le funzionalità di TimescaleDB possano essere applicate a una grande quantità di time-series data, dimostrando la flessibilità e le potenzialità del DBMS.

2 TimescaleDB

2.1 Panoramica

TimescaleDB è un database server open-source sviluppato dall'azienda Timescale per time-series data ottimizzato per l'inserimento veloce dei dati e per le query complesse. TimescaleDB è un'estensione di PostgreSQL[1]: si usa, quindi, come un tradizionale database relazionale che si basa sul linguaggio di full-SQL.

I database che contengono time-series data sono sempre caratterizzati da dimensioni notevoli, che sono destinate a crescere ulteriormente con l'arrivo di nuovi dati. Se si pensa che se si raccolgono dati ogni secondo, si può arrivare a un numero di 86400 righe per tabella al giorno, e in un anno intero si arriverebbe ad un totale di 31536000 righe.

I DBMS tradizionali non sono più sufficienti per gestire queste grandi quantità di time-series, perché non hanno funzionalità dedicate a ottimizzare l'interazione con questa tipologia di dati; TimescaleDB, invece, è stato sviluppato per implementare funzionalità dedicate alla gestione dei time-series data.

L'utilizzo di TimescaleDB è facilitato da alcune caratteristiche quali la connessione ad un qualunque componente che implementi PostgreSQL, le funzionalità e le ottimizzazioni offerte dal DBMS stesso, e il supporto robusto per le policy di conservazione dei dati, ma soprattutto è facilmente comprensibile per gli utenti già abituati alla piattaforma di PostgreSQL che devono solo apprendere le funzionalità aggiuntive e i princìpi dell'architettura su cui TimescaleDB si basa.

Oltre alla facilità di utilizzo, TimescaleDB si distingue per la sua scalabilità e affidabilità. È scalabile perché prevede un partizionamento del tempo e dello spazio dei dati sia per lo *scaling up* di un singolo nodo di dati, cioè per estenderlo o per renderlo più veloce (per aumentare il carico di lavoro), sia per lo *scaling out* futuro, ovvero garantire che in futuro si possano aggiungere parallelamente altri componenti: l'obiettivo del partizionamento è, quindi, la predisposizione all'aumento della quantità di dati.

[1] PostgreSQL è un DBMS open-source per la gestione di database.

È affidabile perché è progettato sulla base di PostgreSQL, già ampiamente consolidato, e, inoltre, offre per la gestione dei dati opzioni flessibili che sono compatibili con l'ambiente e gli strumenti già esistenti di PostgreSQL.

2.2 Time-series data

I time-series data sono dati che, insieme, rappresentano come un sistema, un processo o un comportamento cambia nel tempo. [1]

Le caratteristiche dei time-series data sono:

- la presenza di un timestamp, cioè un dato temporale (**time-centric**),
- l'uso principale di operazioni di inserimento di dati (**append-only**)
- l'appartenenza dei nuovi dati agli intervalli di tempo più recenti (**recent**).

Gli aggiornamenti sui dati esistenti, cioè la sovrascrittura, e il riempimento di dati mancanti vengono effettuati raramente quando si lavora con le serie temporali.

Considerando che i database presentano già fra i tipi di dati basilari quelli temporali, i time-series si distinguono perché necessitano di funzionalità che tengano conto delle loro tre caratteristiche principali. Infatti, ricordando che con i time-series data si lavora principalmente con operazioni di inserimento, sarebbe opportuno avere a disposizione uno strumento che ottimizzi l'inserimento dei dati e non la loro sovrascrittura; inoltre, considerando la loro caratteristica di essere time-centric e recent, sarebbe utile disporre di funzioni che implementino query complesse che selezionino i dati per determinati intervalli di tempo.

Il periodo di campionamento dei dati è variabile: si possono raccogliere dati ogni secondo, ogni quarto d'ora, una volta a settimana, ecc. In base al periodo di campionamento e alla quantità di dati raccolta nel periodo si costruiscono le strutture dati più adatte per rappresentare il modello desiderato e per garantirne adeguate prestazioni per la lettura dei dati: si rimanda al 2.4.2 per approfondire meglio la questione.

2.2.1 Scenari di utilizzo

I time-series data sono presenti in molti ambiti, se non tutti: si tratta, in generale, di raccolte di grandi quantità di dati durante operazioni di monitoraggio; alcuni esempi: sistemi di monitoraggio di computer che raccolgono dati sull'uso della CPU e del disco, sullo spazio disponibile in memoria, sulla disponibilità di periferiche di input/output, …; sistemi di scambio

finanziario; internet of Things, ovvero dati raccolti da sensori di dispositivi indossabili, di macchine industriali, di veicoli, …; sistemi di monitoraggio dell'ambiente che necessitano di raccogliere molti dati come la temperatura, l'umidità, la pressione, le polveri sottili, le emissioni di carbonio; ecc.

Sono tutte operazioni in cui, nelle righe di dati raccolti, compaiono sempre una colonna temporale (solitamente con la data e l'orario) e una o più colonne che contengono i dati misurati al tempo specificato nella colonna temporale.

2.3 Architettura: hypertables e chunks

Dal momento che è basato su PostgreSQL, TimescaleDB eredita tutte le funzionalità di PostgreSQL e, allo stesso tempo, aggiunge nuove funzioni importanti nelle query e nei modelli dei dati per gestire meglio i time-series data.

In particolare, TimescaleDB introduce i concetti di hypertables e di chunks.

2.3.1 hypertables

Una hypertable è una normale tabella di PostgreSQL trasformata, secondo criteri che verranno specificati di seguito, e appare all'utente come, appunto, una normale tabella singola; è il principale punto di interazione con i dati: è l'astrazione di una singola tabella continua su tutti gli intervalli di spazio e di tempo, così che si possa interrogare come una normale tabella tramite SQL [2].

È definita da uno schema standard con i nomi e i tipi delle colonne con almeno una colonna che specifichi un valore temporale e una colonna, opzionale, che specifichi una chiave di partizione ulteriore. Lo sviluppo di una singola istanza di TimescaleDB può contenere diverse hypertable, ognuna con uno schema differente.

Le hypertable si gestiscono come le tabelle di PostgreSQL con i comandi di CREATE, ALTER, DELETE: in TimescaleDB sono identici. L'unica differenza si presenta nella creazione, in quanto il comando tradizionale CREATE TABLE sarà seguito da quello che creerà la hypertable. In TimescaleDB, infatti, è possibile creare hypertable con due comandi SQL: CREATE TABLE (con la sintassi standard SQL) per creare la tabella normale, seguito da "SELECT create_hypertable()" per creare la hypertable.

Di seguito viene riportato un esempio di creazione di hypertable:

```
CREATE TABLE dati_temperatura (
 time        TIMESTAMPTZ NOT NULL,
 location    TEXT NOT NULL,
 temperature DOUBLE PRECISION
);
SELECT create_hypertable('dati_temperatura', 'time');
```

Gli indici sul tempo e la chiave di partizione sono creati automaticamente per la hypertable, con la possibilità di aggiungere ulteriori indici successivamente.

2.3.2 chunks

I chunks sono strutture simili a contenitori che contengono le parti in cui TimescaleDB spezza le hypertable. Ogni chunk corrisponde a uno specifico intervallo temporale e a una regione dello spazio della chiave di partizione. Le partizioni non sono sovrapposte, così da poter minimizzare l'insieme di chunks coinvolti nelle query.

Sono implementati tramite le tabelle standard di un database; in PostgreSQL un chunk è, in realtà, una tabella figlia della tabella genitore hypertable. I chunks sono dimensionati in modo tale che i B-trees (le strutture ad albero che in PostgreSQL contengono gli indici delle tabelle) per gli indici di una tabella risiedano in memoria durante le operazioni di inserimento: ciò permette di gestire eventuali modifiche delle posizioni negli indici in quegli alberi. Se si evitano dei chunks troppo grandi, inoltre, si può risparmiare sulle costose operazioni di "vacuuming" quando si rimuovono i dati cancellati secondo le politiche di conservazione automatizzate. A tempo di esecuzione si possono eseguire queste operazioni semplicemente eliminando i chunks, invece che cancellare le singole righe.

Quando si crea una nuova hypertable, questa viene suddivisa in chunk di dimensione predefinita di 7 giorni, a meno che non si indichi una grandezza diversa nella funzione create_hypertable() (si rimanda al par. 2.4.1).

TimescaleDB effettua operazioni di compressione sui chunk vecchi, convertendo il chunk da una forma non compressa basata su riga ad una forma compressa colonnare, passando attraverso una hypertable. Il meccanismo è automatico: una volta che il chunk sarà diventato abbastanza

vecchio (vedere par. 2.4.2), sarà convertito temporaneamente dalla forma in riga alla forma in colonna.

Gli utenti non noteranno alcuna differenza fra i chunk compressi e quelli ancora “completi”, perché i dati memorizzati nella forma di riga saranno aggiunti in modo trasparente a quelli compressi e al momento delle query TimescaleDB decomprimerà il chunk. Nelle interrogazioni, perciò, i blocchi non compressi saranno elaborati normalmente, mentre i chunk compressi saranno prima decompressi e convertiti in un formato di riga standard al momento della query e/o prima di essere aggiunto o unito ad altri dati.

2.4 Funzionalità

2.4.1 Creazione di una hypertable

La funzione **create_hypertable**() crea una hypertable a partire da una tabella normale di PostgreSQL, rimpiazzandola, partizionata sul tempo e con la possibilità di partizionare su una o più colonne aggiuntive (come quella di un dato relativo allo spazio). La tabella di partenza di PostgreSQL non può essere una tabella già partizionata e nel caso in cui la tabella di partenza non sia vuota, è possibile trasferire i dati durante la creazione dell’hypertable usando l’opzione “migrate_data” (si veda più avanti), tenendo, però, presente che l’operazione può richiedere un molto tempo e che può presentare alcuni limiti quando la tabella contiene delle foreign key. Quando si converte una tabella SQL normale in una hypertable, si deve prestare attenzione alla gestione dei vincoli, perché una hypertable può contenere foreign keys verso colonne di tabelle SQL normali, ma non avviene il contrario. Inoltre, i vincoli di UNIQUE e PRIMARY KEY devono includere la chiave di partizionamento.

Come introdotto nel paragrafo 2.3.1, una volta creata la hypertable sarà possibile gestirla come una tabella normale usando i comandi tradizionali di PostgreSQL.

Vediamo, ora, gli argomenti della funzione.

Argomenti obbligatori:

relation	Identifica la tabella da convertire in una hypertable.
time_column_name	Nome della colonna che contiene il valore temporale, è anche la colonna primaria da partizionare.

	L’argomento “*time_column_name*” supporta i tipi timestamp (TIMESTAMP, TIMESTAMPTZ), DATE o intero (SMALLINT, INT, BIGINT), perciò non è obbligatorio usare un parametro basato sul tempo, purché si possa incrementare.

Argomenti facoltativi:

partitioning_column	Nome di una colonna aggiuntiva da partizionare. Se specificato, l’argomento number_partitions deve essere specificato.
number_partitions	Numero di partizioni da usare per la “partitioning_column”. Deve essere > 0.
chunk_time_interval	Intervallo nel tempo di evento che ogni chunk (vedere dopo) copre. Deve essere > 0. Dalla versione v0.11.0 di default è 7 giorni, per le versioni precedenti è impostato a 1 mese.
	Per le colonne temporali di tipo timestamp o DATE, dovrebbe essere specificato come tipo di intervallo o come valore integrale in microsecondi. Per tipi interi, deve essere specificato esplicitamente, perché il database non comprende diversamente a cosa riferisce il dato intero (secondi, millisecondi, …).
create_default_indexes	Valore booleano che indica se creare indici predefiniti sulle colonne di tempo/partizionamento. Di default è TRUE.
if_not_exists	Valore booleano che indica se mostrare avvertimenti (warning) nel caso in cui una tabella sia già stata convertita in hypertable o se rilevare eccezioni. Di default è FALSE.
partitioning_func	Funzione da usare per calcolare la partizione di un valore.
associated_schema_name	Nome dello schema per tabelle hypertable interne. Di default è “_timescaledb_internal”.

associated_table_prefix	Prefisso per i nomi di chunk di hypertable interne. Di default è "_hyper".
migrate_data	Da impostare a TRUE per trasferire tutti i dati dalla tabella relation verso il chunk della nuova hypertable. Se la tabella non è vuota, la funzione genererà un errore senza questa opzione. Di default è FALSE.
time_partitioning_func	Funzione per convertire i valori delle colonne di tempo primarie incompatibili in valori compatibili. La funzione deve essere IMMUTABLE.
replication_factor	Se impostato a 1 o a un valore superiore, creerà una hypertable distribuita; in questo caso è meglio considerare di usare la funzione dedicata "create_distributed_hypertable" (si veda più avanti per le hypetable distribuite).
data_nodes	Questo è l'insieme dei nodi di dati che sarebbero usati per questa tabella se fosse distribuita. Questo parametro non ha alcun impatto su una hypertable non distribuita. Se non vi è specificato alcun nodo di dati, una hypertable distribuita userà tutti i nodi di dati conosciuti da questa istanza.

La funzione restituisce:

hypertable_id	ID of the hypertable in TimescaleDB.
schema_name	Nome dello schema della tabella convertita.
table_name	Nome della tabella convertita.
created	TRUE se l'hypertable è stata creata, FALSE se non è stata creata alcuna hypertable o se "if_not_exists" è impostato a TRUE.

Se si usa il costrutto "SELECT * FROM create_hypertable(…)" è possibile vedere i valori di ritorno formattati in una tabella con le intestazioni delle colonne.

Seguendo i classici comandi di PostgreSQL si può modificare e/o eliminare una hypertable come si procede per una normale tabella, tramite i comandi di ALTER TABLE e DROP TABLE.

Di seguito vengono riportati alcuni esempi di utilizzo.

Convertire una tabella “conditions” in una hypertable partizionando il tempo sulla colonna “time”:

```
SELECT create_hypertable('conditions', 'time');
```

Convertire una tabella “conditions” in una hypertable, impostando il chunk_time_interval a 24 ore:

```
SELECT create_hypertable('conditions', 'time', chunk_time_interval => 86400000000);

SELECT create_hypertable('conditions', 'time', chunk_time_interval => INTERVAL '1 day');
```

Convertire una tabella “conditions” in una hypertable disattivando gli avvertimenti in caso la tabella sia già stata convertita:

```
SELECT create_hypertable('conditions', 'time', if_not_exists => TRUE);
```

2.4.2 Suggerimenti per il partizionamento del tempo

L’intervallo di tempo è impostato di default a 7 giorni (a partire dalla versione v0.11.0, nelle precedenti vale 1 mese, ma si può modificare tramite il parametro chunk_time_interval alla creazione della hypertable. Dopo la creazione, si può modificare chiamando la funzione set_chunk_time_interval.

La chiave nella scelta dell’intervallo è fare in modo che in memoria si trovi il chunk con il periodo di tempo più recente, così che gli accessi, che si presumono essere più frequenti per i tempi più recenti, siano più veloci. A tal proposito, è consigliabile impostare l’intervallo in modo tale che i chunk non compromettano più del 25% della memoria principale.

Se si vuole visualizzare la lunghezza dell’intervallo corrente per una hypertable, si può controllare il “_timescaledb_catalog” (nota: le lunghezze degli intervalli time-based sono riportate in microsecondi).

```
SELECT h.table_name, c.interval_length

FROM _timescaledb_catalog.dimension c

JOIN _timescaledb_catalog.hypertable h ON h.id = c.hypertable_id;


table_name  | interval_length

------------+-----------------

metrics     |    604800000000

(1 row)
```

Bisogna partire con un'idea generale sulla dimensione dei dati. Se si scrivono circa 2GB di dati al giorno e si hanno a disposizione 64GB di memoria, impostare l'intervallo ad una settimana dovrebbe andare bene. Se si vogliono scrivere, invece, 10GB al giorno per la stessa disponibilità di memoria, è più appropriato impostare l'intervallo ad 1 giorno. Questo intervallo potrebbe reggere anche se i dati fossero caricati in più batch, ad esempio se si caricano 70GB di dati alla settimana, con i dati corrispondenti ai record di tutta la settimana.

Anche se, in generale, è più sicuro avere chunk più piccoli, impostare intervalli troppo piccoli incrementerebbe la latenza delle query.

2.4.3 Le funzioni di TimescaleDB per analizzare i time-series data

TimescaleDB offre un insieme di funzioni utili per l'analisi dei dati memorizzati nelle hypertable: si basano principalmente sulla manipolazione dei periodi di tempo e la possibilità di leggere i dati ordinati per intervalli arbitrari. Di seguito se ne elencano le principali.

La funzione **first**() permette di leggere il primo valore di una colonna rispetto a come stata ordinata da un'altra. Bisogna specificare gli argomenti "value" (colonna di cui voglio leggere il primo valore) e "time" (colonna con cui ho "ordinato" la colonna "value").

La funzione **last**(), invece, legge l'ultimo valore di una colonna rispetto a come stata ordinata da un'altra colonna. Si devono specificare gli argomenti "value" e "time", come per la funzione first().

first() e **last**() non usano gli indici per orientarsi, ma la colonna "time" specificata e vengono usate in selezioni ordinate da un GROUP BY, quindi non rappresentano un'alternativa alla clausola "ORDER BY "time" DESC/ASC LIMIT 1" perché usa gli indici.

La funzione **time_bucket**() è la versione migliorata della funzione date_trunc() di PostgreSQL: rispetto alla precedente, introduce la possibilità di gestire intervalli di tempo arbitrari invece di quelli prestabiliti da PostgreSQL come second, minute, hour, ecc. Gli argomenti obbligatori della funzione sono la lunghezza dell'intervallo temporale "bucket_width" (es: '5 minutes') e la colonna "time" da "spezzare" in bucket. I due parametri elencati valgono anche per input temporali di tipo intero, facendo attenzione solo alla "bucket_width" che va espressa con un intero.

time_bucket_gapfill() è molto simile a time_bucket() e rispetto ad essa aggiunge il riempimento degli intervalli senza valore, intervalli compresi fra i parametri "start" e "finish". Può essere usata solo in query aggregate. Dalla versione 1.3.0, "start" e "finish" sono argomenti opzionali e sono dedotti dalla clausola WHERE se non sono indicati come argomenti della funzione. Gli argomenti obbligatori sono la larghezza dell'intervallo "bucket_width" e "time", ovvero la colonna da "spezzare" negli intervalli dalla larghezza prima specificata.

La funzione **interpolate**() effettua un'interpolazione per i valori mancanti. Può essere usato solo in una query con la funzione time_bucket_gapfill(). Non può essere innestata in altre chiamate a funzione. Richiede l'argomento "value", ovvero il valore da interpretare; si possono specificare anche gli argomenti "prev" e "next", rispettivamente un'espressione per guardare i valori precedenti all'intervallo del gapfill e una per i valori successivi.

La funzione **locf**() ("last observation carried forward", "ultima osservazione portata avanti") permette di riportare e visualizzare l'ultimo valore di un gruppo di aggregazione se quello corrente non ha valore. Quando si usa questa funzione, quindi, se un gruppo di aggregazione non avesse valore, per lui verrebbe visualizzato quello precedente. Può essere usata solo con la funzione time_bucket_gapfill() e non può essere innestata in altre chiamate a funzione. Richiede l'argomento obbligatorio "value", ovvero il valore da riportare; accetta anche altri due argomenti: "prev", l'espressione di ricerca per valori precedenti al gapfill indicato, e "treat_null_as_missing", per indicare alla funzione di ignorare i valori NULL e di riportare soltanto i valori diversi da NULL.

Al capitolo 5 vi sono le applicazioni di alcune di queste funzioni.

3 Progettazione del database dei consumi energetici

In questo capitolo viene proposto un modello per la definizione dei dati. In appendice sono state riportate alcune note sui software usati per sviluppare il progetto.

3.1 Requisiti

I dati di consumo per ogni appartamento sono stati raccolti da un sensore installato sul contatore di riferimento.

L'utenza, nel nostro caso di tipo domestica, è identificata dai dati anagrafici del cliente a cui è intestata, oltre ad essere identificata dal numero del contratto stipulato con l'azienda fornitrice di energia. Nel database è stato scelto di identificare il cliente con il suo codice fiscale (CF) e di indicarne il nome, il cognome, l'indirizzo e-mail e l'indirizzo di residenza; verrà usato spesso il termine utente con lo stesso significato di cliente. In generale, un utente può disporre di più utenze, anche solo banalmente per il possedimento di più appartamenti a lui intestati, perciò verrà specificato che un utente può disporre di più utenze, ricordando, però, che il contatore fa riferimento al singolo appartamento e quindi a una singola utenza.

L'edificio condominiale sarà rappresentato da un'entità identificata da un codice generico (ID_edificio) e descritta da un attributo che indicherà la presenza o meno di cappotto termico e di pannelli solari. Gli appartamenti collocati nell'edificio sono identificati dal codice dell'edificio e dal numero interno dell'appartamento stesso e vengono specificati la presenza di infissi a risparmio energetico e la classe energetica.

Il contatore è installato nell'edificio condominiale e fa riferimento ad una singola utenza, ovvero ad un singolo appartamento del condominio: nel database, perciò, il contatore, e quindi il sensore, deve essere localizzato rispetto all'appartamento a cui fa riferimento e all'edificio in cui l'appartamento e il contatore si trovano. Il contatore, nonché il sensore, viene perciò identificato da un codice POD ricavato dal codice del contratto energetico (questo è il motivo per cui non sono stati forniti dati sui profili delle utenze, per evitare di associare il codice POD a informazioni personali riservate), dal codice dell'edificio in cui si trova e dal numero interno dell'appartamento a cui fa riferimento. Per il contatore viene indicata anche la potenza di accumulo.

Per avere una visione più completa sui consumi energetici, è consigliabile modellare un'entità che tenga traccia dei carichi elettrici, ovvero di qualunque dispositivo che consuma corrente, e due associazioni che colleghino il carico al contatore di riferimento e al locale in cui sono installati, così che l'utente possa verificare quali sono i dispositivi che consumano più energia. Nello schema proposto, un carico elettrico è identificato da un numero progressivo e dall'appartamento e dall'edificio in cui è installato, ed è descritto dalla sua classe energetica e dalla potenza installata. Dei carichi elettrici, inoltre, vengono specificati meglio gli impianti di condizionamento, che di solito sono i dispositivi che consumano di più nella stagione invernale e/o estiva, a seconda che l'impianto disponga di climatizzazione invernale e/o estiva. Nello schema, l'impianto di condizionamento è stato rappresentato da un'entità figlia (subset) dell'entità "carico elettrico" ed ha questi attributi: il numero di serie del dispositivo (num_serie, opzionale: è utile conoscerlo per risalire a ulteriori specifiche elettriche relative ai consumi, ma non è detto che venga registrato come valore) e due attributi opzionali che indicano se l'impianto dispone di climatizzazione invernale e\o estiva.

Per il contatore viene specificata anche la fonte dell'energia venduta all'azienda e poi fornita al contatore: l'energia può provenire da fonti di carbone, gas naturale, prodotti petroliferi, ecc., e da fonti di energia rinnovabili come quella solare o eolica. Data l'importanza delle fonti di energia rinnovabili, si propone, nello schema, di modellare un sottoinsieme dedicato a queste fonti, modellandolo come classe figlia delle fonti di energia. Come identificatore si è scelto un ID generico.

Il sensore, il dispositivo che raccoglie i dati sui consumi, è installato sul contatore, quindi viene identificato da esso, e raccoglie i dati sul consumo energetico dell'appartamento.

I dati raccolti contengono sempre il codice POD e la data in cui sono stati registrati. Vengono raccolti anche dati inerenti alla corrente consumata, al voltaggio, alla potenza attiva e a quella reattiva oppure all'energia attiva e a quella reattiva: dipende dal contenuto dei file sorgenti a disposizione.

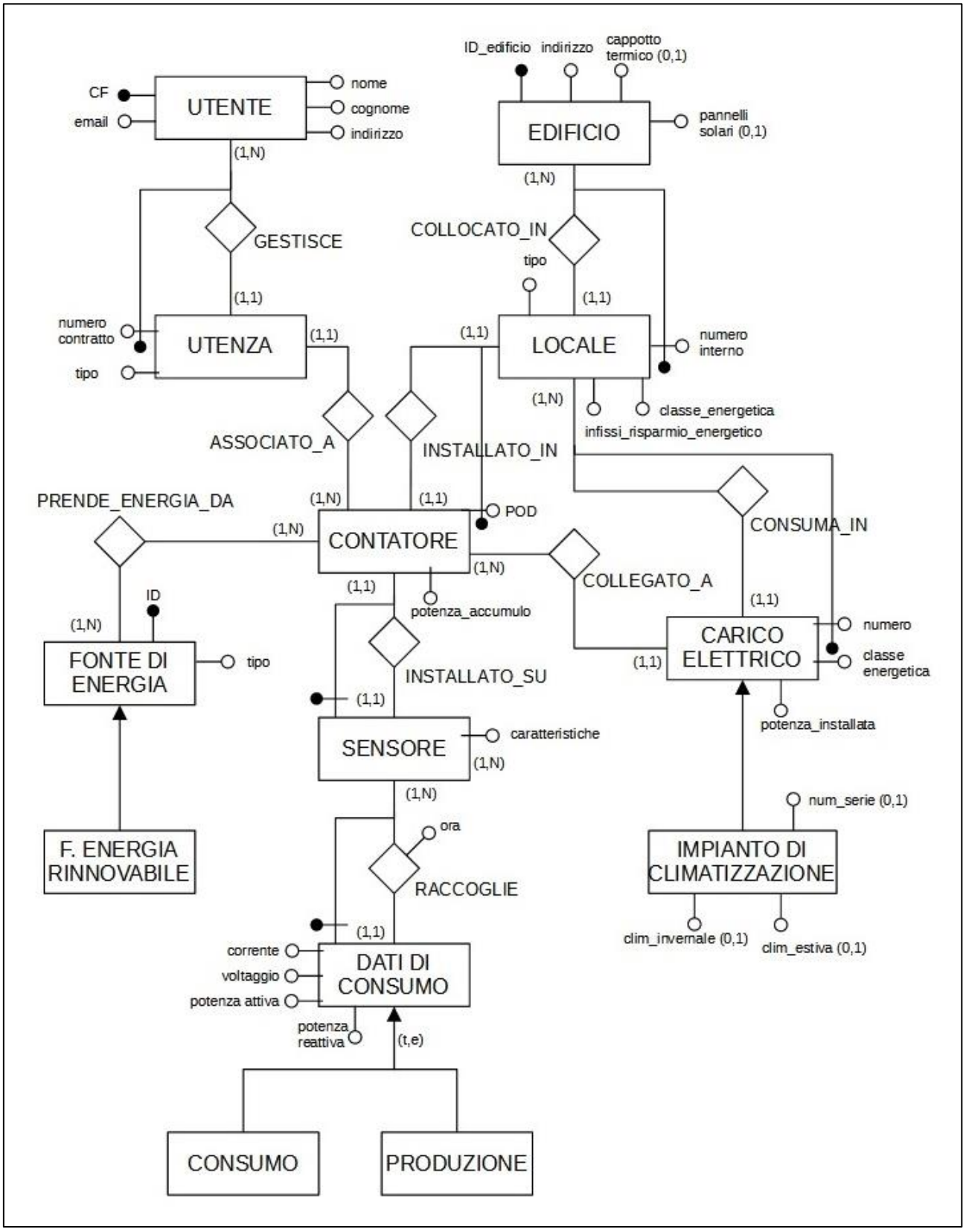
Sulla base dello schema descritto, è stato realizzato con il linguaggio SQL il database tramite il DBMS PostgreSQL. Nei paragrafi seguenti vengono mostrati lo schema E/R e il database proposti.

Nello schema E/R rappresentato, l'entità "locale" rappresenta, in realtà, un appartamento come descritto sopra: si è scelto di modellarlo con un termine più generico, perché lo schema potrebbe

essere esteso anche ad un locale condominiale, ad esempio le scale o il locale caldaie, oppure agli uffici di un edificio aziendale, motivo per cui all'entità è stato anche affiancato l'attributo "tipo" che identifica proprio il tipo di locale fra i tre elencati. Di conseguenza, anche alla classe "utenza" viene aggiunto un attributo "tipo", perché i costi dei consumi variano in base ai contratti stipulati con le aziende fornitrici di energia, le quali offrono piani tariffari diversi agli utenti a seconda che si tratti di locali aziendali, domestici o condominiali.

Riferito sempre allo schema E/R, si sono aggiunte per completezza due classi figlie dei "dati di consumo": si distinguono i dati di consumo da quelli di produzione inseriti in rete. In questo progetto l'attenzione è rivolta principalmente ai dati di consumo, perciò i quelli di produzione non verranno considerati.

3.2 Schema E/R


UTENTE
CF
email
nome
cognome
indirizzo
(1,N)
GESTISCE
(1,1)
UTENZA
numero contratto
tipo
(1,1)
EDIFICIO
ID_edificio
indirizzo
cappotto termico (0,1)
pannelli solari (0,1)
(1,N)
COLLOCATO_IN
tipo
(1,1)
LOCALE
(1,1)
numero interno
(1,N)
classe_energetica
infissi_risparmio_energetico
ASSOCIATO_A
INSTALLATO_IN
PRENDE_ENERGIA_DA
(1,N)
(1,1)
POD
(1,N)
CONTATORE
(1,N)
(1,1)
potenza_accumulo
CONSUMA_IN
COLLEGATO_A
(1,1)
CARICO ELETTRICO
numero
classe energetica
potenza_installata
(1,1)
ID
(1,N)
FONTE DI ENERGIA
tipo
INSTALLATO_SU
(1,1)
SENSORE
caratteristiche
(1,N)
(1,N)
ora
RACCOGLIE
(1,1)
corrente
voltaggio
potenza attiva
potenza reattiva
DATI DI CONSUMO
(t,e)
F. ENERGIA RINNOVABILE
num_serie (0,1)
IMPIANTO DI CLIMATIZZAZIONE
clim_invernale (0,1)
clim_estiva (0,1)
CONSUMO
PRODUZIONE

3.3 Definizione dei dati con SQL

3.3.1 Definizione dei dati statici

```
CREATE DATABASE dati_energetici;


CREATE TABLE utente (
    CF varchar(50) PRIMARY KEY,
    nome varchar(50) NOT NULL,
    cognome varchar(50) NOT NULL,
    email varchar(50) NOT NULL,
    indirizzo varchar(60) NOT NULL
);


CREATE TABLE edificio (
    ID_edificio bigint PRIMARY KEY,
    indirizzo varchar(60) NOT NULL,
    cappotto_termico boolean,
    pannelli_solari boolean
);


CREATE TYPE tipo_locale AS ENUM ('appartamento', 'ufficio aziendale', 'locale condominiale');
CREATE TABLE locale (
        numero_interno int NOT NULL,
        ID_edificio bigint NOT NULL,
        tipo tipo_locale NOT NULL,
```

```
        classe_energetica varchar(10),
        infissi_risp_en boolean,
        PRIMARY KEY (numero_interno, ID_edificio),
        FOREIGN KEY (ID_edificio) REFERENCES edificio
);

CREATE TABLE contatore (
    POD varchar(50) NOT NULL,
    num_interno_locale int NOT NULL,
    ID_edificio bigint NOT NULL,
    potenza_accumulo double precision,
    PRIMARY KEY (POD, num_interno_locale, ID_edificio),
    FOREIGN KEY (num_interno_locale, ID_edificio) REFERENCES locale (numero_interno, ID_edificio)
);

CREATE TYPE tipo_utenza AS ENUM ('domestica', 'aziendale', 'condominiale');
CREATE TABLE utenza (
    numero_contratto varchar(50) NOT NULL,
    CF_utente varchar(50) NOT NULL,
    tipo tipo_utenza NOT NULL,
    POD varchar(50) NOT NULL,
    num_interno_locale int NOT NULL,
    ID_edificio bigint NOT NULL,
    PRIMARY KEY (numero_contratto, CF_utente),
```

```
    FOREIGN KEY (CF_utente) REFERENCES utente (CF),
    FOREIGN KEY (POD, num_interno_locale, ID_edificio) REFERENCES contatore (POD, num_interno_locale, ID_edificio)
);

CREATE TABLE carico_elettrico (
    num int NOT NULL,
    num_interno_locale int NOT NULL,
    ID_edificio bigint NOT NULL,
    POD varchar(50) NOT NULL,
    potenza_installata double precision,
    differibilita double precision,
    attenuabilita double precision,
    classe_energetica varchar(10),
    PRIMARY KEY (num, num_interno_locale, ID_edificio),
    FOREIGN KEY (POD, num_interno_locale, ID_edificio) REFERENCES contatore(POD, num_interno_locale, ID_edificio),
    FOREIGN KEY (num_interno_locale, ID_edificio) REFERENCES locale (numero_interno, ID_edificio)
);

CREATE TABLE impianto_di_climatizzazione (
    num int NOT NULL,
    num_interno_locale int NOT NULL,
    ID_edificio bigint NOT NULL,
```

```
    num_serie varchar(50) UNIQUE,
    climatizzazione_invernale boolean,
    climatizzazione_estiva boolean,
    PRIMARY KEY (num, num_interno_locale, ID_edificio),
    FOREIGN KEY (num, num_interno_locale, ID_edificio) REFERENCES carico_elettrico
);

CREATE TABLE sensore (
    POD varchar(50) NOT NULL,
    num_interno_locale int NOT NULL,
    ID_edificio bigint NOT NULL,
    caratteristiche varchar(100),
    PRIMARY KEY (POD, num_interno_locale, ID_edificio),
    FOREIGN KEY (POD, num_interno_locale, ID_edificio) REFERENCES
contatore(POD, num_interno_locale, ID_edificio)
);
```

3.3.2 Definizione dei dati dinamici

I dati esemplificativi su cui è stato sviluppato il progetto sono contenuti in tre file in formato ".csv". Il primo file fa riferimento a diversi appartamenti e contiene i consumi energetici di una giornata (29-08-2020) con dati raccolti ogni secondo; i dati sono stati raccolti da un dispositivo NILM[2].

[2] I NILM sono dispositivi che permettono di raccogliere in tempo reale tutti i dati di consumo al secondo e sono installato presso i contatori, ovvero i POD. Questi sensori forniscono con elevata accuratezza (margine di errore dell'1%) le misure di potenza attiva, reattiva, tensione e corrente aggiornate al secondo. [7]

Il secondo e il terzo file contengono, rispettivamente, i consumi energetici relativi all'arco temporale di due anni circa, con dati raccolti da contatori di nuova generazione[3], di due appartamenti in uno stesso condominio. I dati di questi due file sono stati forniti da ARERA.

Di seguito vengono riportate le definizioni delle tabelle che conterranno i dati.

```
CREATE TABLE consumi1 (
    POD varchar(50) NOT NULL,
    ora TIMESTAMP NOT NULL,
    corrente double precision,
    voltaggio double precision,
    potenza_attiva double precision,
    potenza_reattiva double precision,
    PRIMARY KEY (POD, ora)
    FOREIGN KEY (POD) REFERENCES sensore (POD)
);
```

```
CREATE TABLE consumicasa1 (
    pod varchar(50) NOT NULL,
    data_lettura timestamp without time zone NOT NULL,
    data_ricezione timestamp without time zone,
    motivazione varchar(50),
    tipo_flusso varchar(50),
    annomese_riferimento varchar(50),
```

[3] Si tratta di misuratori elettronici (o smart meter) con sistema di telegestione di seconda generazione, la cui installazione è iniziata a partire dal 2017. Il distributore acquisisce i consumi effettuati in ciascuno dei 96 quarti d'ora giornalieri (dati quartorari) e li trasmette giornalmente (per il tramite del Sistema Informativo Integrato) al venditore. Da: *Il portale consumi, domande frequenti, Cosa sono i misuratori di seconda generazione (o 2G) per l'energia elettrica?* [4]

```
    energia_attiva double precision,
    energia_reattiva double precision
);
```

La definizione della tabella “consumicasa2” è analoga a quella di “consumicasa1”.

3.4 Estensione del database a TimescaleDB

Dopo aver definito la struttura dei dati, è possibile convertire le tabelle interessate, quelle riferite ai consumi, in hypertable: dove possibile, per sfruttare al meglio le potenzialità di TimescaleDB è consigliabile procedere prima con la definizione delle tabelle normali e immediatamente dopo con la loro conversione in hypertable e non con l’inserimento dei dati, per evitare di doverli trasferire dalla tabella vecchia alla nuova (si rimanda al paragrafo 2.4.1).

Prima di procedere alla creazione delle hypertable, è necessario attivare l’estensione di TimescaleDB per il database, assicurandosi di eseguire l’operazione dopo aver effettuato la connessione al database interessato:

```
CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Per la conversione della tabella in hypertable si applica la funzione create_hypertable() introdotta al capitolo 1 (paragrafo 2.4.1):

```
SELECT create_hypertable('consumi1', 'ora');
SELECT create_hypertable('consumicasa1', 'data_lettura');
SELECT create_hypertable('consumicasa2', 'data_lettura');
```

4 Inserimento dei dati

In presenza di hypertable, si può procedere all'inserimento dei dati principalmente in due modi: si possono trasportare quelli già presenti nella tabella di partenza nella hypertable (vedere par. 2.4.1) oppure, in modo più efficiente, inserirli solo dopo la creazione della hypertable. Si possono inserire i dati usando anche comandi classici come INSERT INTO oppure tramite una SELECT; dipende dal caso in analisi, soprattutto per quanto riguarda la quantità di dati da inserire: in presenza di time-series data, solitamente si deve inserire una grande quantità di dati, perciò è consigliabile usare o il comando \COPY di PostgreSQL o un altro comando che verrà introdotto nel paragrafo successivo.

Nel nostro caso si parte da tabelle vuote, quindi si procederà ad inserire i dati nelle tabelle solo dopo averle trasformate in hypertable per evitare tempi attesa troppo lunghi (par. 2.4.1). Poiché i dati dinamici sono stati forniti in file in formato ".csv", si potrebbe procedere con il comando \COPY che permette l'inserimento da un file sorgente specificando l'estensione e il delimitatore che separa i dati nelle righe, ma l'operazione risulterebbe molto lenta perché la dimensione dei file è molto grande, caratteristica tipica dei file che raccolgono time-series data. La soluzione viene proposta nel paragrafo seguente.

4.1 Il comando timescaledb-parallel-copy per inserire i dati dinamici

Per questo progetto è stato scelto di usare **timescaledb-parallel-copy** [3] per l'inserimento dei dati dinamici: è uno strumento offerto da TimescaleDB che sfrutta l'architettura parallela del processore eseguendo il comando \COPY in parallelo sui core della CPU per ottimizzare i tempi di inserimento.

Digitando "timescaledb-parallel-copy" --help è possibile leggere la lista delle opzioni da affiancare al comando; qui vengono elencate le principali.

copy-options *string*	Opzione aggiuntiva da passare a COPY (es: NULL 'NULL') (default "CSV").
db-name *string*	Database che contiene la tabella di destinazione.
file *string*	File da cui leggere i dati, invece che dallo standard input.

limit *int*	Numero di righe da inserire complessivamente. Per inserirle tutte indicare 0.
reporting-period *duration*	Periodo per segnalare le statistiche di inserimento; se 0, i risultati intermedi non saranno riportati.
skip-header	Salta la prima riga del file di input.
split *string*	Carattere divisorio degli elementi (default ",").
table *string*	Tabella di destinazione per gli inserimenti (default "test_table").
workers *int*	Numero di richieste/COPY parallele da fare (default 1).

```
timescaledb-parallel-copy --db-name dati_energetici          \
--table consumi1 --file Condominio_2020-08-29.csv --workers 4  \
--reporting-period 15s --skip-header
```

Nell'esempio sopra sono stati inseriti i dati dal file "Condominio_2020-08-29.csv" nella tabella "consumi1" nel database "dati_energetici" ed è stato specificato di usare 4 core (workers), di riportare informazioni sull'inserimento ogni 15 secondi e di saltare la prima riga del file perché è l'intestazione. Una volta terminate le diverse \COPY viene restituita una stringa con la scritta "COPY" e il numero di righe copiate.

4.2 Inserimento dei dati statici

Nelle informazioni fornite non sono presenti dati statici da poter inserire nel modello del database; pertanto, in questo progetto è stato deciso di inserire dati statici esemplificativi che non hanno alcuna corrispondenza con la realtà, ma rappresentano solo esempi di supporto per le interrogazioni svolte nel capitolo 5.

5 Analisi dei dati con TimescaleDB

In questo capitolo vengono proposte alcune interrogazioni sui dati inseriti nel database per mostrare le funzioni principali di TimescaleDB e per suggerire qualche modello di analisi dei consumi. La prima tabella “consumi 1” (par. 3.3.2) verrà usata principalmente per mostrare le potenzialità delle funzioni di TimescaleDB (par. 2.4.3), mentre sulle altre due, “consumicasa1” e “consumicasa2”, verranno eseguite interrogazioni più approfondite per l’analisi dei consumi.

5.1 Interrogazioni su time-series data raccolti al secondo

Si consideri la tabella “consumi1” (par. 3.3.2) relativa ai dati di consumi energetici di diversi contatori di una giornata. Dal momento che si hanno a disposizione dati per ogni secondo, sfruttando le funzioni time_bucket() o time_bucket_gapfill() di TimescaleDB si possono selezionare intervalli di tempo abbastanza piccoli in cui visualizzare le medie dei valori, il valore massimo, ecc.

Query 1: calcolare il valore medio della potenza attiva ogni 5 minuti per ogni POD.

```
SELECT pod, time_bucket('5 minutes', ora) AS cinque_minuti,
avg(potenza_attiva) AS potenza_attiva
FROM consumi1
GROUP BY pod, cinque_minuti
ORDER BY cinque_minuti DESC LIMIT 10;
```

```
   pod     |    cinque_minuti    |   potenza_attiva
----------+---------------------+--------------------
 00000100 | 2020-08-29 23:55:00 |  84.99453333333331
 00000104 | 2020-08-29 23:55:00 | 124.91302666666665
 00000108 | 2020-08-29 23:55:00 |                  0
 00000114 | 2020-08-29 23:55:00 | 157.21337999999997
 00000115 | 2020-08-29 23:55:00 | 248.32111999999998
 00000126 | 2020-08-29 23:55:00 | 120.67065666666659
 00000128 | 2020-08-29 23:55:00 |          273.00459
 00000133 | 2020-08-29 23:55:00 | 146.87102333333337
 00000137 | 2020-08-29 23:55:00 |                  0
 00000139 | 2020-08-29 23:55:00 |  69.33239999999998
```

Query 2. Visualizzare il primo e l'ultimo valore di una colonna rispetto ad un intervallo temporale, per esempio 5 minuti, rispettivamente con le funzioni first() e last():

```
SELECT pod, time_bucket('5 minutes', ora) AS interval,
  last(potenza_attiva, ora), first(potenza_attiva, ora)
FROM consumi1
WHERE ora >= TIMESTAMP '2020-08-09'
GROUP BY POD, interval
ORDER BY interval DESC LIMIT 10;
```

```
   pod    |      interval       |  last   |  first
----------+---------------------+---------+---------
 00000100 | 2020-08-29 23:55:00 |   33.71 |  31.864
 00000104 | 2020-08-29 23:55:00 | 112.461 | 134.822
 00000108 | 2020-08-29 23:55:00 |       0 |       0
 00000114 | 2020-08-29 23:55:00 | 120.527 | 182.647
 00000115 | 2020-08-29 23:55:00 | 159.032 | 287.851
 00000126 | 2020-08-29 23:55:00 |  48.696 | 180.966
 00000128 | 2020-08-29 23:55:00 | 216.754 | 289.218
 00000133 | 2020-08-29 23:55:00 | 145.827 | 149.238
 00000137 | 2020-08-29 23:55:00 |       0 |       0
 00000139 | 2020-08-29 23:55:00 |  68.248 |  67.489
(10 rows)
```

Query 3. Per ogni intervallo di tempo, ad esempio 2 ore, trovare i valori massimi e minimi:

```
SELECT pod, time_bucket('2 hours', ora) AS giorno,
max(corrente) AS max_corrente, max(voltaggio) AS max_voltaggio,
min(potenza_attiva) AS min_potenza_attiva,
min(potenza_reattiva) AS min_potenza_reattiva
FROM consumi1
GROUP BY pod, giorno
ORDER BY pod ASC LIMIT 5;
```

```
   pod    |       giorno        | max_corrente | max_voltaggio |
----------+---------------------+--------------+---------------+
 00000100 | 2020-08-29 00:00:00 |        1.085 |           233 |
 00000100 | 2020-08-29 02:00:00 |        0.822 |           233 |
 00000100 | 2020-08-29 04:00:00 |        0.891 |           233 |
 00000100 | 2020-08-29 06:00:00 |        0.877 |           232 |
 00000100 | 2020-08-29 08:00:00 |         0.31 |           234 |

| min_potenza_attiva | min_potenza_reattiva
+--------------------+----------------------
|             26.636 |                -1.12
|              27.21 |               -8.287
|             25.446 |              -10.894
|             25.964 |                -8.06
|                  0 |                   0
(5 rows)
```

Query 4. Per ogni POD, visualizzare la media della potenza attiva per l'intera giornata:

```
SELECT pod, time_bucket('24 hours', ora) AS giorno, avg(potenza_attiva) AS potenza_attiva

FROM consumi1

GROUP BY pod, giorno

ORDER BY pod ASC LIMIT 10;
```

```
   pod    |       giorno        |   potenza_attiva
----------+---------------------+--------------------
 00000100 | 2020-08-29 00:00:00 |  37.61752027777825
 00000104 | 2020-08-29 00:00:00 | 126.86746179398268
 00000108 | 2020-08-29 00:00:00 |  5.917085601851854
 00000114 | 2020-08-29 00:00:00 |  90.96456346064878
 00000115 | 2020-08-29 00:00:00 | 223.38836667823955
 00000126 | 2020-08-29 00:00:00 |  184.2929245601871
 00000128 | 2020-08-29 00:00:00 | 255.82758322916357
 00000133 | 2020-08-29 00:00:00 | 179.37076722222037
 00000137 | 2020-08-29 00:00:00 |                  0
 00000139 | 2020-08-29 00:00:00 |  83.30119591435222
(10 rows)
```

5.2 Interrogazioni sui time-series data al quarto d'ora

Si consideri, ora, la tabella "consumicasa1" (par. 3.3.2). Si propongono di seguito alcune query utili a dividere i consumi per anno e per le fasce orarie di consumo proposte da ARERA.

Orario	Feriali	Sabato	Domenica e Festivi
Dalle ore 7:00 alle ore 8:00	F2	F2	F3
Dalle ore 8:00 alle ore 19:00	F1	F2	F3
Dalle ore 19:00 alle ore 23:00	F2	F2	F3
Dalle ore 23:00 alle ore 7:00	F3	F3	F3

Figura 1 Fasce orario di consumo per l'energia elettrica di ARERA.

Dal sito di ARERA si apprende che le fasce orarie di consumo sono periodi di tempo ai quali possono corrispondere diversi prezzi dell'energia elettrica.

Le fasce tradizionali stabilite dall'ARERA si distinguono in:

- F1 (ore di punta): lun-ven dalle 8.00 alle 19.00, escluse festività nazionali;
- F2 (ore intermedie): lun-ven dalle 7.00 alle 8.00 e dalle 19.00 alle 23.00, sabato dalle 7.00 alle 23.00, escluse festività nazionali;
- F3 (ore fuori punta): lun-sab dalle 23.00 alle 7.00 e la domenica e i festivi tutta la giornata;

Nel caso di clienti domestici serviti in maggior tutela i consumi sono distinti in fascia F1, come sopra definita, e fascia F23, che comprende tutte le ore incluse nelle fasce F2 e F3. [4]

Poiché in questo progetto sono presenti solo dati relativi a utenze domestiche, verrà proposta una query secondo le due fasce orarie a loro dedicate (F1 e F23), ma le successive distingueranno la fascia 2 e la fascia 3 per fornire un modello completo nel caso in cui si vogliano applicare le query a utenze diverse da quelle domestiche.

Di seguito le interrogazioni per leggere la media dei consumi mensili per le fasce F1 e F23.

Query 5. Visualizzare il valore medio dell'energia attiva per ogni mese negli orari corrispondenti alla fascia 1.

```
SELECT time_bucket('30 days', data_lettura) AS mese,
avg(energia_attiva)*1000 AS F1_kW

FROM consumicasa1

WHERE (date_part('hour', data_lettura) >= 8 AND date_part('hour', data_lettura) < 19)
AND date_part('dow', data_lettura) BETWEEN 1 AND 5

GROUP BY mese

ORDER BY mese ASC

LIMIT 13;
```

```
        mese         |       f1_kw
---------------------+--------------------
 2019-01-23 00:00:00 |
 2019-02-22 00:00:00 |   48.7584134615385
 2019-03-24 00:00:00 | 45.929513888888984
 2019-04-23 00:00:00 |  49.62715517241372
 2019-05-23 00:00:00 |  53.00902777777784
 2019-06-22 00:00:00 |           67.90625
 2019-07-22 00:00:00 | 55.485119047618944
 2019-08-21 00:00:00 |  45.99548611111128
 2019-09-20 00:00:00 |  48.35763888888878
 2019-10-20 00:00:00 | 55.126736111111484
 2019-11-19 00:00:00 |  68.84645061728384
 2019-12-19 00:00:00 |  60.26813271604931
 2020-01-18 00:00:00 |  71.51006944444423
(13 rows)
```

Query 6. Visualizzare il valore medio dell'energia attiva per ogni mese negli orari corrispondenti alla fascia F23:

```
SELECT time_bucket('30 days', data_lettura) AS mese,
avg(energia_attiva)*1000 AS f23_kw

FROM consumicasa1

GROUP BY mese

EXCEPT

SELECT time_bucket('30 days', data_lettura) AS mese,
avg(energia_attiva)*1000 AS consumiF1

FROM consumicasa1

WHERE (date_part('hour',data_lettura) >= 8 AND date_part('hour',data_lettura) < 19)

AND date_part('dow', data_lettura) BETWEEN 1 AND 5

GROUP BY mese

LIMIT 13;
```

```
        mese         |       f23_kw
---------------------+--------------------
 2020-09-14 00:00:00 |  69.13333333333331
 2020-07-16 00:00:00 | 114.58020833333283
 2019-12-19 00:00:00 | 60.268132716049195
 2019-04-23 00:00:00 |  49.62715517241412
 2019-09-20 00:00:00 | 48.357638888889404
 2020-08-15 00:00:00 |  69.38124999999981
 2020-03-18 00:00:00 |   79.3253472222221
 2019-02-22 00:00:00 | 48.758413461538815
 2020-06-16 00:00:00 |  96.59131944444407
 2019-06-22 00:00:00 |  67.90624999999991
 2019-08-21 00:00:00 |   45.9954861111114
 2019-07-22 00:00:00 | 55.485119047619406
 2020-05-17 00:00:00 |  84.50694444444476
(13 rows)
```

Query 7. Visualizzare i consumi nella fascia F2 per ogni mese:

```
SELECT time_bucket('30 days', data_lettura) AS mese, avg(energia_attiva)*1000 AS
consumiF2
FROM consumicasa1
WHERE (date_part('hour',data_lettura) = 7
AND date_part('dow', data_lettura) BETWEEN 1 AND 6)
OR ((date_part('hour',data_lettura) >= 8 AND date_part('hour',data_lettura) < 19 )
    AND date_part('dow', data_lettura) = 6)
OR ((date_part('hour',data_lettura) >= 19 AND date_part('hour',data_lettura) < 23 )
    AND date_part('dow', data_lettura) BETWEEN 1 AND 6)
GROUP BY mese
ORDER BY mese ASC;
```

Query 8. Visualizzare i consumi nella fascia F3 per ogni mese:

```
SELECT time_bucket('30 days', data_lettura) AS mese,
avg(energia_attiva)*1000 AS consumiF3

FROM consumicasa1

WHERE date_part('dow', data_lettura) = 0

OR data_lettura IN (SELECT giorno FROM giornifestivi)

OR (date_part('hour',data_lettura) >= 23)

OR (date_part('hour',data_lettura) <= 7)

GROUP BY mese

ORDER BY mese ASC;
```

```
        mese         |     consumif3
---------------------+--------------------
 2019-01-23 00:00:00 |
 2019-02-22 00:00:00 |  46.33154121863801
 2019-03-24 00:00:00 | 42.052898550724706
 2019-04-23 00:00:00 |  49.68925233644858
 2019-05-23 00:00:00 |  48.59848484848467
 2019-06-22 00:00:00 |  60.72619047619053
 2019-07-22 00:00:00 | 42.923076923076735
 2019-08-21 00:00:00 |  43.51818181818174
 2019-09-20 00:00:00 |  42.72954545454542
 2019-10-20 00:00:00 |  49.35724637681168
 2019-11-19 00:00:00 | 61.466981132075745
 2019-12-19 00:00:00 |  51.69077834179381
 2020-01-18 00:00:00 |  66.56086956521753
 2020-02-17 00:00:00 |   57.0598484848486
 2020-03-18 00:00:00 | 59.431060606060704
 2020-04-17 00:00:00 |  56.63030303030306
 2020-05-17 00:00:00 |  65.65434782608718
 2020-06-16 00:00:00 |  69.34696969696998
 2020-07-16 00:00:00 |   66.3553030303034
 2020-08-15 00:00:00 |  54.77391304347842
 2020-09-14 00:00:00 |  43.96111111111108
(21 rows)
```

5.3 Confronto fra i consumi relativi a un anno di due appartamenti

È possibile isolare i consumi relativi ad un anno. Di seguito, le istruzioni in SQL per creare una tabella, “consumicasa1_2019”, che contenga i consumi di ogni mese del 2019 relativi alla tabella “consumicasa1”:

```
CREATE TABLE consumi2019 (
    mese timestamp NOT NULL,
    consumi_fascia1_kw double precision,
    consumi_fascia2_kw double precision,
    consumi_fascia3_kw double precision
);
```

```
INSERT INTO consumi2019 (mese, consumi_fascia1_kw)
SELECT time_bucket('30 days', data_lettura) as mese, avg(energia_attiva)*1000
FROM consumicasa1
WHERE (date_part('hour',data_lettura) >= 8 AND date_part('hour',data_lettura) < 19
AND date_part('dow', data_lettura) BETWEEN 1 AND 5)
AND date_part('year', data_lettura) = 2019
GROUP BY mese;
```

Dopo aver eseguito altre due INSERT analoghe all’ultima, una per “consumi_fascia2_kw” e l’altra per “consumi_fascia3_kw”, si può creare una tabella finale per i consumi relativi all’anno 2019 in questo modo:

```
CREATE TABLE consumicasa1_2019 AS
SELECT mese, sum(consumi_fascia1_kw) AS energia_attiva_f1_kw,
sum(consumi_fascia2_kw) AS energia_attiva_f2_kw,
sum(consumi_fascia3_kw) AS energia_attiva_f3_kw
FROM consumi2019
GROUP BY mese
ORDER BY mese ASC;
```

Così, interrogando la tabella "consumicasa1_2019" si possono visualizzare i consumi (valori medi) per ogni mese del 2019.

Query 9. Visualizzare tutti i dati della tabella "consumicasa1_2019":

```
SELECT *
FROM consumicasa1_2019;
```

```
        mese         | energia_attiva_f1_kw | energia_attiva_f2_kw | energia_attiva_f3_kw
---------------------+----------------------+----------------------+---------------------
 2019-01-23 00:00:00 |                      |                      |
 2019-02-22 00:00:00 |    43.46291866028706 |     59.4103773584905 |   53.298611111111114
 2019-03-24 00:00:00 |   49.753246753246785 |   47.949704142011775 |    42.87152777777775
 2019-04-23 00:00:00 |    47.90367965367966 |    51.78106508875729 |    56.08854166666661
 2019-05-23 00:00:00 |    50.68491735537175 |   62.563218390804515 |   60.411458333333314
 2019-06-22 00:00:00 |     57.8385167464113 |     91.0485714285715 |    77.76458333333326
 2019-07-22 00:00:00 |    66.52159090909083 |   62.278963414633985 |    39.31666666666668
 2019-08-21 00:00:00 |    43.78512396694214 |    52.59482758620692 |    48.21093750000002
 2019-09-20 00:00:00 |    51.20346320346301 |    54.85810810810814 |    45.54166666666665
 2019-10-20 00:00:00 |    57.42099567099575 |    62.96153846153842 |    56.69965277777774
 2019-11-19 00:00:00 |    63.95655080213922 |    85.84772727272738 |    76.21041666666665
 2019-12-19 00:00:00 |   52.649621212121104 |   58.053427419354946 |   40.920673076923066
(12 rows)
```

Si può costruire una tabella “consumicasa2_2019” anche per i consumi contenuti in “consumicasa2” in modo analogo a come è stata creata la tabella “consumicasa1_2019”.

Query 10. Visualizzare tutti i dati della tabella “consumicasa2_2019”:

```
SELECT *

FROM consumicasa2_2019;
```

```
        mese         | energia_attiva_f1_kw | energia_attiva_f2_kw | energia_attiva_f3_kw
---------------------+----------------------+----------------------+---------------------
 2019-01-23 00:00:00 |                      |                      |
 2019-02-22 00:00:00 |   23.403409090909115 |   41.095486111111164 |   27.679563492063547
 2019-03-24 00:00:00 |    22.63825757575762 |   39.282608695652186 |   29.643145161290512
 2019-04-23 00:00:00 |   26.772727272727252 |    34.89367816091954 |   33.442424242424266
 2019-05-23 00:00:00 |   24.690082644628134 |   46.295977011494244 |    35.89318181818189
 2019-06-22 00:00:00 |     31.3545454545454 |    45.50416666666664 |     42.3202898550726
 2019-07-22 00:00:00 |   28.067099567099447 |    39.28994082840234 |    30.32943925233655
 2019-08-21 00:00:00 |    26.67871900826452 |    37.51580459770122 |   31.915151515151514
 2019-09-20 00:00:00 |    23.27922077922073 |    49.90540540540542 |    29.54469696969708
 2019-10-20 00:00:00 |   24.968614718614674 |   27.983727810650862 |   25.399999999999963
 2019-11-19 00:00:00 |   30.257231404958656 |    39.54454022988502 |    32.33636363636367
 2019-12-19 00:00:00 |   20.558080808080835 |   30.766233766233782 |    27.05442176870747
(12 rows)
```

Si può notare come i consumi del primo appartamento siano in media più alti del secondo, anche solo visualizzando i valori massimi per le varie fasce orarie, come nella query successiva.

Query 11. Visualizzare i valori massimi dell'energia attiva per le prime due fasce orarie delle tabelle "consumicasa1_2019" e "consumicasa2_2019":

```
SELECT max(A.energia_attiva_f1_kw) AS max_casa1_F1,
max(B.energia_attiva_f1_kw) AS max_casa2_F1,
max(A.energia_attiva_f2_kw) AS max_casa1_F2,
max(B.energia_attiva_f2_kw) AS max_casa2_F2
FROM consumicasa1_2019 A, consumicasa2_2019 B
WHERE A.mese = B.mese
GROUP BY A.mese
ORDER BY A.mese ASC;
```

```
    max_casa1_f1    |    max_casa2_f1    |    max_casa1_f2    |    max_casa2_f2
--------------------+--------------------+--------------------+--------------------
                    |                    |                    |
  43.46291866028706 | 23.403409090909115 |   59.4103773584905 | 41.095486111111164
 49.753246753246785 |  22.63825757575762 | 47.949704142011775 | 39.282608695652186
  47.90367965367966 | 26.772727272727252 |  51.78106508875729 |  34.89367816091954
  50.68491735537175 | 24.690082644628134 | 62.563218390804515 | 46.295977011494244
   57.8385167464113 |   31.3545454545454 |   91.0485714285715 |  45.50416666666664
  66.52159090909083 | 28.067099567099447 | 62.278963414633985 |  39.28994082840234
  43.78512396694214 |  26.67871900826452 |  52.59482758620692 |  37.51580459770122
  51.20346320346301 |  23.27922077922073 |  54.85810810810814 |  49.90540540540542
  57.42099567099575 | 24.968614718614674 |  62.96153846153842 | 27.983727810650862
  63.95655080213922 | 30.257231404958656 |  85.84772727272738 |  39.54454022988502
 52.649621212121104 | 20.558080808080835 | 58.053427419354946 | 30.766233766233782
(12 rows)
```

Le ragioni di questo risultato possono essere diverse: nel primo appartamento potrebbero risiedere più persone rispetto al secondo e, di conseguenza, i consumi del primo risultano maggiori del secondo. Poiché l'informazione delle presenze in un appartamento non è nota, il modo migliore di confrontare questi dati è indagare sulle caratteristiche dell'appartamento circa l'installazione di carichi elettrici nell'appartamento, la presenza o meno di pannelli solari e/o di infissi che garantiscano un buon grado di isolamento termico, ricordando che i due appartamenti si trovano nello stesso condominio.

Query 12. Visualizzare tutti i dati relativi all'edificio e all'appartamento in cui si trova il contatore dell'appartamento a cui la tabella "consumicasa1" fa riferimento:

```
SELECT s.POD, e.ID_edificio, e.indirizzo, l.numero_interno,
l.tipo, l.classe_energetica, e.cappotto_termico, l.infissi_risp_en, e.pannelli_solari
FROM sensore s, locale l, edificio e, consumicasa1 c
WHERE s.num_interno_locale = l.numero_interno
AND l.ID_edificio = e.ID_edificio AND s.POD = c.POD
GROUP BY s.POD, e.ID_edificio, e.indirizzo, l.numero_interno,
l.tipo, l.classe_energetica, e.cappotto_termico, l.infissi_risp_en, e.pannelli_solari
```

```
      pod       | id_edificio |            indirizzo             | numero_interno |
----------------+-------------+----------------------------------+----------------+
 IT000E00000001 |           2 | via Giacomo Leopardi, 34, Modena |              1 |

|     tipo     | classe_energetica | cappotto_termico | infissi_risp_en | pannelli_solari
+--------------+-------------------+------------------+-----------------+----------------
| appartamento | C                 | t                | f               | f
(1 row)
```

Query 13. Visualizzare tutti i dati relativi all'edificio e all'appartamento in cui si trova il contatore dell'appartamento a cui la tabella "**consumicasa2**" fa riferimento. La query differisce dalla precedente soltanto per la tabella "consumicasa2" che sostituisce "consumicasa1":

```
      pod       | id_edificio |            indirizzo             | numero_interno |
----------------+-------------+----------------------------------+----------------+
 IT000E00000002 |           2 | via Giacomo Leopardi, 34, Modena |              2 |

|     tipo     | classe_energetica | cappotto_termico | infissi_risp_en | pannelli_solari
+--------------+-------------------+------------------+-----------------+----------------
| appartamento | A                 | t                | t               | f
(1 row)
```

Mettendo a confronto le caratteristiche dei due appartamenti, si può leggere che il primo, quello dai consumi più alti, non ha gli infissi delle porte finestre a risparmio energetico, mentre il secondo, dai consumi più bassi, sì: questo confronto può rappresentare per il cliente

consumatore un suggerimento per ottimizzare i consumi nel proprio appartamento, ad esempio tentare un investimento nell'installazione di infissi termicamente più isolanti di quelli già presenti.

Si potrebbero anche visualizzare i carichi elettrici installati in un appartamento, per individuare quale consuma energia più degli altri e, eventualmente, limitarne l'uso per ottimizzare i consumi.

Query 14. Visualizzare tutti i dati dei carichi elettrici installati nell'appartamento a cui la tabella "consumicasa1" fa riferimento:

```
SELECT S.pod, C.num AS num_carico, C.potenza_installata, C.descrizione
FROM sensore S, carico_elettrico C, consumicasa1 C1
WHERE S.pod = C.pod
AND S.pod = C1.pod
GROUP BY S.pod, C.num, C.potenza_installata, C.descrizione
```

```
      pod       | num_carico | potenza_installata |  descrizione
----------------+------------+--------------------+----------------
 IT000E00000001 |          1 |                2.1 | lavastoviglie
 IT000E00000001 |          2 |               0.09 | lampada
 IT000E00000001 |          3 |                3.5 | climatizzatore
(3 rows)
```

Infine, è utile visualizzare a parte eventuali dettagli registrati degli impianti di climatizzazione installati nell'appartamento, in quanto rappresentano una tipologia di carico che consuma molta energia.

Query 15. Visualizzare tutti i dati degli impianti di climatizzazione installati nell'appartamento a cui la tabella "consumicasa1" fa riferimento:

```
SELECT S.pod, C.num AS n_carico, C.potenza_installata AS potenza, C.descrizione,
I.climatizzazione_invernale AS clim_invernale,
I.climatizzazione_estiva AS clim_estiva
FROM sensore S, carico_elettrico C, consumicasa1 C1, impianto_di_climatizzazione I
WHERE S.pod = C.pod
AND S.pod = C1.pod
AND I.num = C.num
AND I.num_interno_locale = C.num_interno_locale
AND I.ID_edificio = C.ID_edificio
GROUP BY S.pod, C.num, C.potenza_installata, C.descrizione,
I.climatizzazione_invernale,
I.climatizzazione_estiva
```

```
      pod       | n_carico | potenza |  descrizione   | clim_invernale | clim_estiva
----------------+----------+---------+----------------+----------------+-------------
 IT000E00000001 |        3 |     3.5 | climatizzatore | f              | t
(1 row)
```

Conclusioni

In questa tesi sono state presentate le funzionalità principali di TimescaleDB e sono state applicate al database progettato per analizzare i time-series dei consumi energetici di alcuni appartamenti. Nel paragrafo 5.3, in particolare, è stata usata la funzione time_bucket() per creare una tabella dedicata ai consumi energetici relativi a un anno, sulla quale si sono, successivamente, eseguite interrogazioni standard basate principalmente su GROUP BY.

Si immagini di applicare le funzioni illustrate (par. 2.4) ad una quantità di dati maggiore, registrata su un periodo temporale più lungo: le funzioni come time_bucket() acquisiscono ancora più potenzialità, in quanto è possibile selezionare intervalli diversi secondo necessità, e il ruolo dei chunks assume ancora più importanza per l'accesso veloce dei dati più recenti in memoria. L'analisi dei dati, inoltre, in questo progetto si è basata su file "conclusi", cioè che contengono dati registrati in un periodo di tempo concluso, ma se si avessero a disposizione dati in tempo reale, le query con TimescaleDB possono diventare molto più potenti: si possono selezionare, ad esempio, dati registrati ogni cinque minuti nell'ultima settimana combinando, nella query, la funzione time_bucket() e la funzione now(). Di seguito viene riportato un esempio, dove si suppone che la tabella "consumi1" sia aggiornata in tempo reale.

Query 16. Visualizzare il valore medio della potenza_attiva registrata ogni cinque minuti nella tabella "consumi1" nell'ultima settimana:

```
SELECT pod, time_bucket('5 minutes', ora) AS cinque_minuti,
avg(potenza_attiva) AS potenza_attiva
FROM consumi1
WHERE ora > now() - INTERVAL '1 week'
GROUP BY pod, cinque_minuti
ORDER BY cinque_minuti DESC;
```

Con una maggiore quantità di dati, poi, si possono avere molti più edifici di quelli considerati in questo progetto, da cui consegue che le query come la precedente si possono arricchire di join con altre tabelle per visualizzare informazioni sulle caratteristiche dei luoghi di consumo (come visto al par. 5.3): con molti edifici a disposizione, si possono anche raggruppare i risultati

delle interrogazioni per ID_edificio, così da individuare precise aree geografiche della città in cui si indentificano comunità energetiche locali, ovvero si cerca di identificare i profili di consumo energetico rispetto ad una località, che sia una strada, un complesso di edifici o un quartiere; in questo modo si possono localizzare eventuali interventi da parte delle aziende riguardo le infrastrutture di fornitura energetica o l'installazione di nuovi contatori, e si possono illustrare ai consumatori soluzioni migliori per il consumo di energia, nonché proposte di contratti economicamente più favorevoli di quelli attuali.

Infine, nell'ipotesi di disporre di molti più dati, aumenta la probabilità che in qualche riga nelle tabelle manchino alcuni valori; in questo caso, nell'analisi dei dati assume una grande importanza la funzione time_bucket_gapfill() alla quale si può associare la funzione locf() o interpolate(): le righe del result set della query senza valore verranno riempite con un'interpolazione o con la ripetizione di un dato precedente, rendendo, così, l'analisi dei dati completa e continua, e ponendo una base valida per la costruzione di eventuali grafici, per visualizzare l'andamento dei dati nel tempo.

Appendice

Indicazioni sui software utilizzati per lo sviluppo del progetto

Per sviluppare il progetto si è scelta la piattaforma Docker con PostgreSQL versione 12, sul sistema operativo Windows 10.

Prima è stata scaricata l'immagine di TimescaleDB per Docker dal sito https://hub.docker.com/r/timescale/timescaledb/ (si deve indicare la versione giusta di PostgreSQL), eseguendo la riga seguente sulla riga di comando:

```
docker pull timescale/timescaledb: latest-pg12
```

Poi, è stato eseguito, sempre dalla linea di comando, la riga seguente per creare il container "timescaledb" sull'immagine scaricata precedente:

```
docker run -d –name timescaledb -p 5432:5432 -e POSTGRES_PASSWORD=password \
timescale/timescaledb:latest-pg12
```

e, infine, si esegue il comando "exec" per connettersi al container eseguendo il comando "psql", così da connettersi immediatamente a postgres, con l'utente "postgres":

```
docker exec -it timescaledb psql -U postgres
```

Da qui si possono creare database manualmente digitando righe di comandi uno alla volta, oppure creare script file che possono essere eseguiti da postgres tramite il comando \i:

```
\i file.sql
```

In alternativa, si può gestire il database dall'interfaccia pgAdmin, fornita insieme a PostreSQL al momento dell'installazione su Windows 10 o tramite download dal sito https://www.pgadmin.org/.

Bibliografia

[1] Timescale, «What Is Time-series Data?,» 2018. [Online]. Available: https://docs.timescale.com/latest/introduction/time-series-data. [Consultato il giorno 4 Marzo 2021].

[2] Timescale, «Architecture & Concepts,» [Online]. Available: https://docs.timescale.com/latest/introduction/architecture. [Consultato il giorno 4 Marzo 2021].

[3] timescale, «timescaledb-parallel-copy,» 2021. [Online]. Available: https://github.com/timescale/timescaledb-parallel-copy. [Consultato il giorno marzo 2021].

[4] ARERA, «Il portale consumi, domande frequenti,» [Online]. [Consultato il giorno marzo 2021].

[5] Timescale, «Hypertable Basics,» [Online]. Available: https://docs.timescale.com/latest/using-timescaledb/hypertables. [Consultato il giorno 4 Marzo 2021].

[6] Timescale, «Distributed Hypertables,» [Online]. Available: https://docs.timescale.com/latest/using-timescaledb/distributed-hypertables. [Consultato il giorno 2021 Marzo 7].

[7] selfuser.it, «RILEVARE E MONITORARE I CONSUMI DELLA COMUNITA',» [Online]. Available: https://www.selfuser.it/attivita/dispositivi-di-misura/#:~:text=I%20dispositivi%20sono%20di%20tipo,e%20corrente%20aggiornate%20al%20secondo.. [Consultato il giorno marzo 2021].

[8] Timescale, «TimescaleDB API Reference,» [Online]. Available: https://docs.timescale.com/latest/api. [Consultato il giorno marzo 2021].

[9] Timescale, «TimescaleDB Documentation,» [Online]. Available: https://docs.timescale.com/latest/main. [Consultato il giorno marzo 2021].