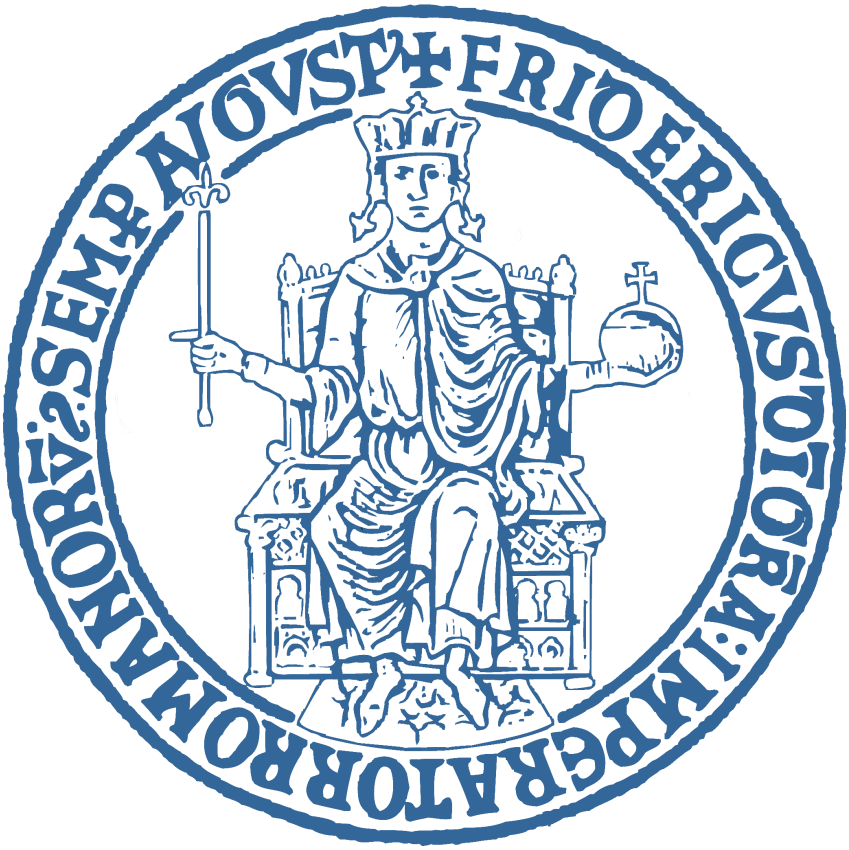
Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea Magistrale in Informatica

A.A. 2017-18

Progetto Basi di Dati II Modulo B

Autori

Bizzarri Flavio — **N97000281**

Cuomo Daniele — **N97000270**

Sommario

Si vuole realizzare un Data Warehouse di tipo ROLAP (Relational On-Line Analytical Processing) allo scopo di analizzarne le funzionalità che lo caratterizzano. Il campione scelto è un dataset fornito dall'Istituto Motori presso il Consiglio Nazionale delle Ricerche. Tale dataset fornisce una serie di rilevazioni effettuate durante un test di un'autovettura su strada nell'area metropolitana di Napoli. Fra i dati più importanti troviamo: l'orario della rilevazione, i consumi, le emissioni ed altri parametri interni al motore.

Indice

Capitolo 1

Progettazione

In questa capitolo sono esposte le scelte progettuali usate come linee guida per l'implementazione di una base di dati orientata al data warehousing.

1.1 Interrogazioni

Nella tabella 1.1 sono riportate le query che si intende sottoporre al sistema. Per ognuna di esse si è associato un numero identificativo ed una descrizione dell'analisi che si intende effettuare. Le interrogazioni hanno guidato lo sviluppo del sistema in ogni sua fase e hanno

🔑	Query	Obiettivo
I	Impatto ambientale medio in corse da 5 km (CO2 e NOx, massa)	Pensata per analizzare varianti della stessa interrogazione su diversi livelli di granularità
C	Consumo medio per intervalli di velocità prefissati, su tutto il dataset	Utile all'implementazione e l'analisi di viste materializzate che raggruppano i dati secondo delle fasce di velocità. Le fasce scelte, espresse in km/h, sono le seguenti: 0-50, 50-90, 90-130
E	Efficienza dell'auto per intervalli di RPM (rotazioni per minuto)	Altra interessante interrogazione creata allo scopo di sfruttare le viste materializzate. Il dataset fornisce tutti i parametri necessari al calcolo dell'efficienza, o rendimento istantaneo
M	Per ogni test, media di NOx, CO2, Potenza e Velocità	Quest'interrogazione serve a valutare le prestazioni ottenute dall'esecuzione su di un partizionamento verticale con le colonne sparse tra più tabelle
T	Media e deviazione standard delle temperature	Quest'interrogazione serve a valutare le prestazioni ottenute dall'esecuzione con le colonne concentrate su di un unica tabella restituita da un partizionamento verticale

permesso di sperimentare differenti tecniche di implementazione di un sistema ROLAP. Una definizione formale delle stesse in linguaggio SQL sarà presentata più avanti nel corso di questo documento.

1.2 Diagrammi

1.2.1 Schema Principale

Di seguito è riportato il diagramma UML che rappresenta lo schema dei fatti implementato secondo il modello relazionale ROLAP.

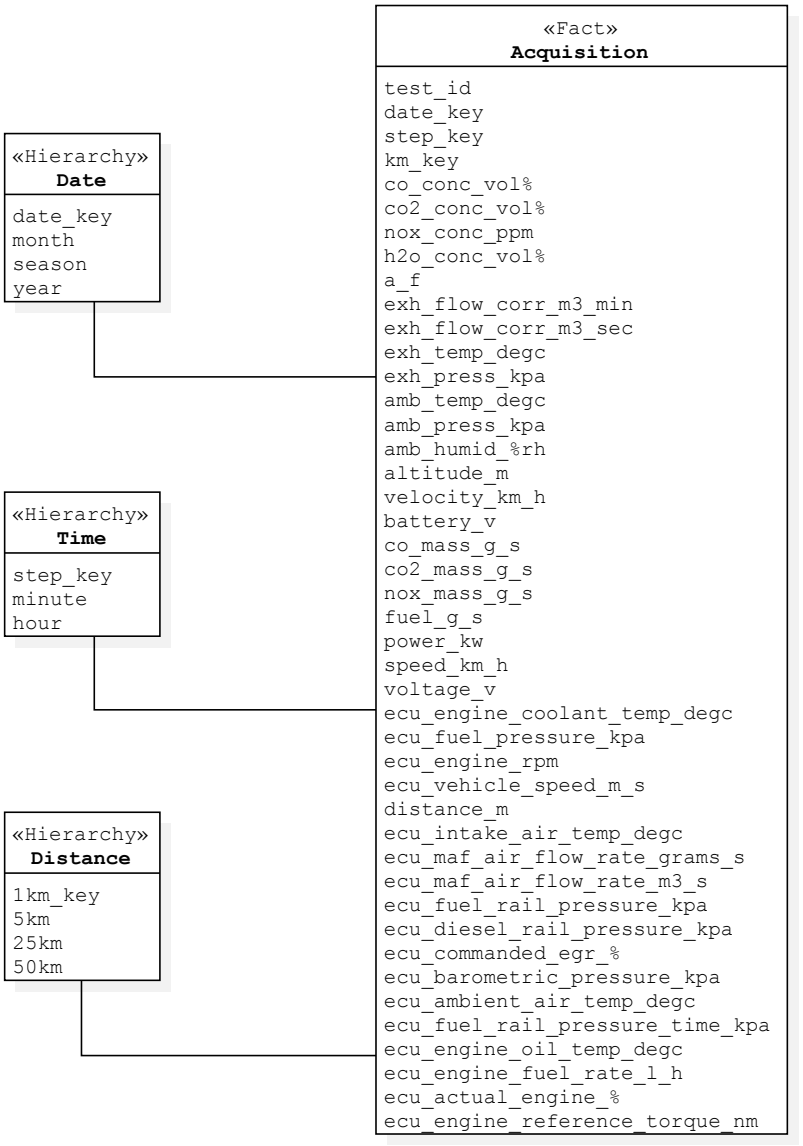


Figura 1.1: Diagaramma UML schema dei fatti

1.2.2 Partizionamento Verticale

È stata inoltre progettata e implementata una variante dello schema proposto che sfrutta la tecnica del partizionamento verticale ovvero la possibilità di dividere la tabella dei fatti in più tabelle, ognuna delle quali rappresenta una particolare sfaccettatura del *fatto*. Il partizionamento viene solitamente adoperato per agevolare quelle interrogazioni che riguardano solo una particolare area di interesse. Per poter applicare questa tecnica è necessario aggiungere una chiave tra le tabelle al fine di poterle ricongiungere per analizzare il dato nella sua completezza.

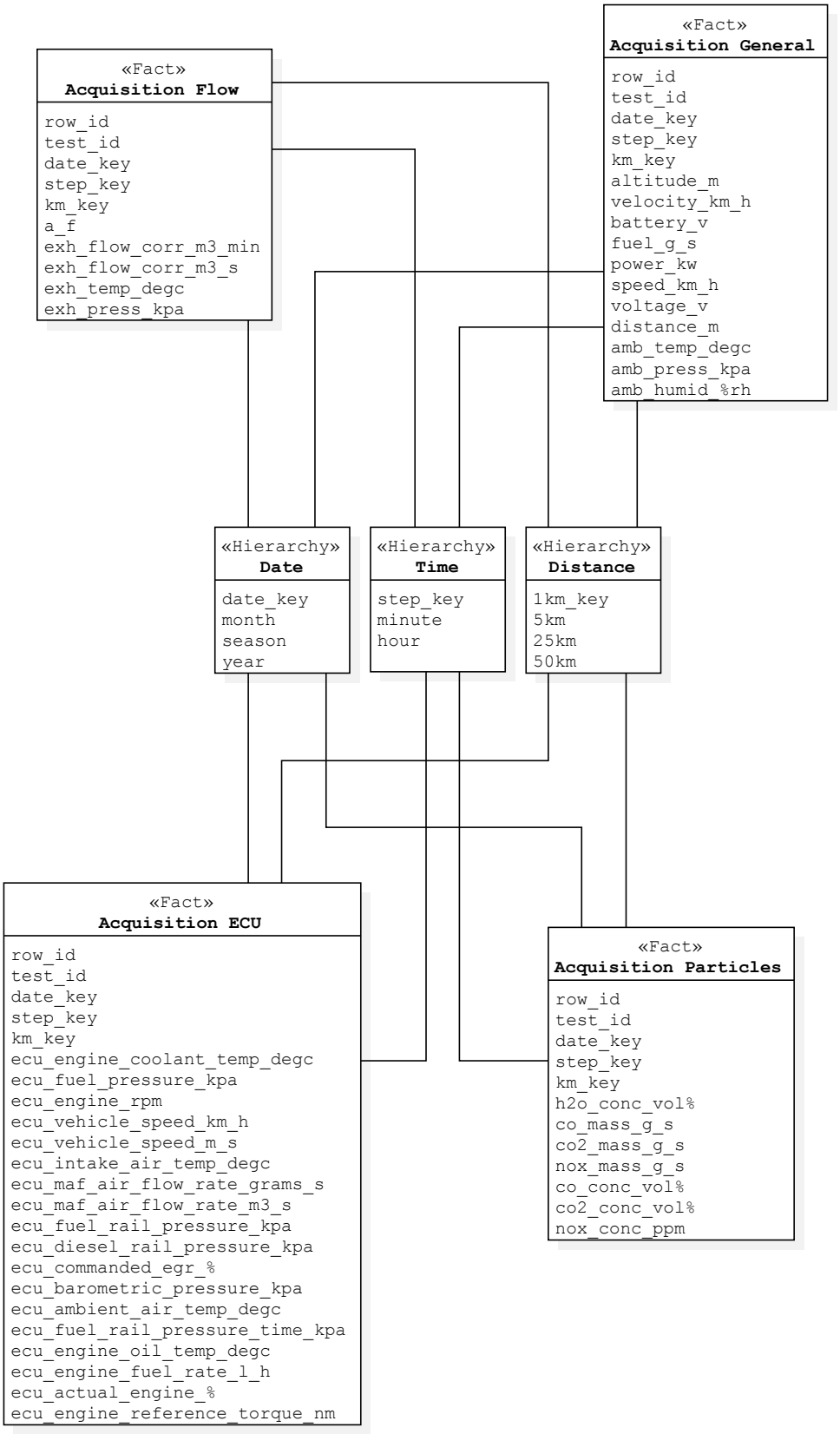


Figura 1.2: Diagramma UML schema dei fatti partizionato verticalmente

1.3 Extract, Transform, Load

Al fine di avere un numero rilevante di dati per l'analisi dei tempi è stato implementato un meccanismo di duplicazione dei file. La procedura ETL sviluppata si compone di differenti passaggi descritti nella seguente tabella.

#	Passaggio	Descrizione	Implementazione
1	Trasformazione dei file XLSX in formato CSV	Eliminazione delle righe contenenti dati inconsistenti Calcolo dei campi formula Creazione di una data fittizia Creazione di un test_id Salvataggio in formato CSV	Java
2	Importazione dei dati in tabella provvisoria	Viene sfruttata il comando copy del DBMS per importare i dati in maniera efficiente	PostgreSQL
3	Shutdown degli indici		PostgreSQL
4	Aggiornamento dello schema	Aggiornamento delle dimensioni Aggiornamento tabella dei fatti	PostgreSQL
5	Riattivazione degli indici		PostgreSQL
6	Aggiornamento viste		PostgreSQL

Per righe inconsistenti si intendono tutte quelle righe ove per valori indicanti volumi e/o concentrazioni vi sono valori negativi; inoltre viene dedotta la distanza percorsa ove mancante sfruttando la distanza percorsa e le velocità istantanee rilevate negli istanti precedenti.

1.4 Viste Materializzate

Sono state create le seguenti viste materializzate al fine di implementare efficacemente le query **C** ed **E**.
Il primo frammento dichiara una vista contenente per ogni corsa (*test_id*) il carburante consumato suddiviso per fasce di velocità.

```
CREATE MATERIALIZED VIEW IF NOT EXISTS fuel_compare_speed AS
SELECT t1.id, t1.fuel_litres AS consumo_litri_meno_di_50km_h, t2.fuel_litres
    AS consumo_litri_meno_di_90km_h, t3.fuel_litres AS
    consumo_litri_meno_di_130km_h
FROM
   (SELECT test_id AS id, ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
```

```
   WHERE speed_km_h <= 50
   GROUP BY test_id) t1
JOIN (
   SELECT test_id AS id, ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
   WHERE speed_km_h >=50 AND speed_km_h<=90
   GROUP BY test_id) t2 ON t1.id = t2.id
JOIN (
   SELECT test_id AS id , ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
   WHERE speed_km_h >=90
   GROUP BY test_id) t3 ON t1.id = t3.id;
```

L'estratto a seguire serve a calcolare una vista materializzata riportante l'efficienza dell'auto nelle diverse corse. I risultati sono stati distribuiti su 3 fasce di rotazioni per minuto (RPM).

```
CREATE MATERIALIZED VIEW IF NOT EXISTS efficiency_compare_rpm AS
SELECT t1.id, ROUND((t1.rendimento*100)::NUMERIC, 2) AS
    efficiency_perc_max2000rpm, ROUND((t2.rendimento*100)::NUMERIC,2) AS
    efficiency_perc_max3000rpm, ROUND((t3.rendimento*100)::NUMERIC,2) AS
    efficiency_perc_over3000rpm
FROM
   (SELECT test_id AS id, AVG(power_kw)/ (AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
   WHERE engine_RPM_by_ecu_rpm < 2000
   GROUP BY test_id) t1
JOIN (
   SELECT test_id AS id, AVG(power_kw)/ (AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
   WHERE engine_RPM_by_ecu_rpm >= 2000 AND engine_RPM_by_ecu_rpm <= 3000
   GROUP BY test_id
   ) t2 ON t1.id = t2.id
JOIN (
   SELECT test_id as id, AVG(power_kw)/( AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
```

```
WHERE engine_RPM_by_ecu_rpm >= 3000
GROUP BY test_id
)t3 ON t1.id = t3.id;
```

Capitolo 2

Analisi

In questo capitolo sono riportati i risultati e i tempi ottenuti per ogni fase: dalle procedure ETL all'esecuzione delle query.

2.1 Prestazioni ETL

2.1.1 Trasformazione in CSV

In questa sezione copriremo un'analisi dei tempi ottenuti nella fase che permette, a partire dai dati originali, di ottenere un file in formato CSV contenente i dati ripuliti da inconsistenze e in un formato adatto ad essere importato nel Datawarehouse. In particolare nella fase di puliza, a partire dal file XSLX, si estraggono unicamente i record dove non appaiono valori negativi per quantità intrinsecamente positive (concentrazione, volume, ecc.); per i record ove manca il valore "relative" e/o la distanza percorsa si procede ad un calcolo a partire dall'ultima rilevazione valida estratta. Inoltre tutte le righe che presentano una chiara assenza di dati (70% delle colonne) vengono scartate. Infine, durante la fase di trasformazione, per ogni file viene generata una data e un test_id e per separare la parte intera da quella decimale si sostituisce la virgola con il punto.
I dati ottenuti si riferiscono ad una media aritmetica ottenuta testando 200 file (∼1 milione di righe) generati a partire dal file originario fornito dall'Istituto Motori di Napoli.

Righe XSLX	Righe CSV	Righe perse	Peso file XSLX	Peso file CSV
5764	5706	1%	2.138KB	2.283KB

Fase di pulizia	Fase di trasformazione	Tempo totale
3.04s	0,05s	3,05s

Questa fase mette in luce la buona qualità dei dati forniti dall'istituto: ci si aspetta in media di perdere pochissimi dati a causa di inconsistenze. Per quanto riguarda invece

le dimensioni si nota come il formato Comma-separated values sia leggermente meno efficiente nella compressione rispetto al formato proprietario di Microsoft® ma al tempo stesso permetta una più veloce elaborazione.

2.1.2 Import in Tabella Temporanea

Al fine di importare i file CSV nella tabella dei fatti ci si appoggia ad una tabella temporanea al fine di agevolare le operazioni successive. Questa tabella viene troncata alla fine della procedura. L'import sfrutta la funzione *COPY* messa a disposizione dal DBMS PostgreSQL.

Numero file	Numero righe	Tempo
1	5706	214ms
50	285.300	8,428s
100	570.600	16.522s

Tabella 2.1: I dati si riferiscono ad una media di 5 esecuzioni

I risultati mostrano la bontà della funzione *COPY* che sfruttando un inserimento batch di 1000 righe per volta riesce ad abbattere in modo consistente i tempi.

2.1.3 Import nello Schema

In questa fase i dati, precedentemente inseriti in una tabella temporanea, vengono travasati nello schema proposto dopo aver disabilitato tutti gli indici. Le prove effettuate prendono in considerazione diverse dimensioni della tabella dei fatti oltre al caso in cui lo schema risulti partizionato.

Dimensione tab. fatti	Numero righe importate	Partizionamento	Tempo
0	285.300	No	3,925s
0	285.300	Si	7,753s
570.600	285.300	No	3,730s
570.600	285.300	Si	7,404s
1.141.200	285.300	No	4,535s
1.141.200	285.300	Si	8,572s

Tabella 2.2: I dati si riferiscono ad una media di 5 esecuzioni

Dalla tabella emerge come il partizionamento comporti quasi un raddoppio del tempo necessario all'inserimento: ciò è dovuto al dover spalmare un singolo record della tabella temporanea su 4 differenti tabelle. Questo slow-down è risolvibile pensando a 4 inserimenti in parallelo: infatti le 4 tabelle, anche se logicamente collegate, durante l'inserimento non necessitano di condividere alcuna informazione.

2.1.4 Aggiornamento degli Indici

In questa fase vengono riattivati gli indici delle chiavi tra tabella dei fatti e dimensioni. Questi indici sono assolutamente necessari per velocizzare le query ma possono essere disabilitati durante la fase di update dello schema.

Dimensione tab. fatti	Partizionamento	Tempo
285.300	No	2,530s
285.300	Si	11,534s
570.600	No	6,194s
570.600	Si	24,715s
1.141.200	No	16,903s
1.141.200	Si	67,469s
1.426.500	No	24,141s
1.426.500	Si	98,218s

Tabella 2.3: I dati si riferiscono ad una media di 5 esecuzioni

Dai dati sopra mostrati emerge ancora una volta come l'introduzione di un partizionamento verticale comporti un notevole rallentamento. In particolare la forbice tra i tempi registrati aumenta all'aumentare della dimensione dei fatti. Ciò induce a pensare attentamente all'introduzione di un partizionamento in fase di progettazione valutando il rapporto costo/benefici.

2.1.5 Aggiornamento delle Viste Materializzate

A valle dell'inserimento dei nuovi record si rende necessario l'aggiornamento delle viste materializzate utilizzate per velocizzare le query **E** ed **M**.

Dimensione tab. fatti	Efficiency_compare_rpm_mv	Fuel_compare_speed_mv
285.300	0,307s	0,238s
570.600	0,983s	0,898s
1.141.200	2,464s	2,173s
1.426.500	3,004s	2,863s

Tabella 2.4: I dati si riferiscono ad una media di 5 esecuzioni

I tempi osservati mostrano come un raddoppio del numero di record comporti un più che raddoppio del tempo necessario all'aggiornamento delle viste. Notevole è infatti l'incremento di tempo per materializzare la vista con 1.141.200 record: 2.5 volte quello necessario per materializzare la vista con la metà dei record.

2.2 Prestazioni Query

In questa sezione verranno analizzate le query implementate nelle differenti implementazioni e presentati gli snippet di codice SQL

2.2.1 Query I

Impatto ambientale medio in corse da 5 km

A seguire sono proposte due diverse formulazioni della stessa interrogazione. Il primo estratto chiede al sistema di recuperare l'appartenenza ad una certa tratta (di 5 km) dalla tabella *distance_hierarchy*. Nella seconda forma la tratta viene calcolata, eseguendo quindi un'operazione di roll-up implicito.

Roll-up esplicito

```
SELECT F.test_id, H.km_5, AVG(F.co2_mass_g_s), AVG(F.nox_mass_g_s)
FROM acquisition_fact F INNER JOIN distance_hierarchy H
   ON (F.km_key = H.km_key)
GROUP BY F.test_id, H.km_5
```

Roll-up implicito

```
SELECT F.test_id, FLOOR(F.km_key/5), AVG(F.co2_mass_g_s), AVG(F.nox_mass_g_s)
FROM acquisition_fact F
GROUP BY F.test_id, FLOOR(F.km_key/5)
```

Esito

L'estrazione dei tempi di esecuzione delle due query può fornire una misura di utilità per una dimensione. In questo caso è interessante notare come un *fatto* possa essere ricondotto alla sua dimensione di appartenenza attraverso un calcolo richiedente 2 operazioni elementari.

2.2.2 Query C

Consumo medio per intervalli di velocità prefissati

L'utilizzo della vista materializzata permette di abbattere i tempi di esecuzione della query fino a un trecentesimo: in questo modo il costo di aggiornamento della vista viene immediatamente ammortizzato.

Dimensione tab. fatti	Senza vista	Con vista
570.600	0,905s	0,020
1.141.200	2,286s	0,026s
1.426.500	2,925s	0.028s

Tabella 2.5: Test effettuati su schema non partizionato

```
SELECT *
FROM fuel_compare_speed_mv;

SELECT t1.id, t1.fuel_litres AS consumo_litri_meno_di_50km_h, t2.fuel_litres
    AS consumo_litri_meno_di_90km_h, t3.fuel_litres AS
    consumo_litri_meno_di_130km_h
FROM
   (SELECT test_id AS id, ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
   WHERE speed_km_h <= 50
   GROUP BY test_id) t1
JOIN (
   SELECT test_id as id, ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
   WHERE speed_km_h >= 50 AND speed_km_h <= 90
   GROUP BY test_id) t2 ON t1.id = t2.id
JOIN (
   SELECT test_id AS id , ROUND(SUM((fuel_g_s)/1000)::NUMERIC/0.8,2) AS
      fuel_litres, ROUND(AVG(velocity_km_h)::NUMERIC,1) AS average_speed_km_h
   FROM acquisition_fact
   WHERE speed_km_h >= 90
   GROUP BY test_id) t3 ON t1.id = t3.id;
```

2.2.3 Query E

Efficienza dell'auto per intervalli di RPM

Si conferma quanto osservato nel caso della query **C**: utilizzare una vista materializzata permette di abbattere i tempi di esecuzione della query e il costo del refresh risulta ampiamente ammortizzato.

Dimensione tab. fatti	Senza vista	Con vista
570.600	0,937s	0,018s
1.141.200	2,182s	0,028s
1.426.500	2,892s	0,030s

Tabella 2.6: Test effettuati su schema non partizionato

```
SELECT *
FROM efficiency_compare_rpm_mv;

SELECT t1.id, ROUND((t1.rendimento*100)::NUMERIC, 2) AS
    efficiency_perc_max2000rpm, ROUND((t2.rendimento*100)::NUMERIC,2) AS
    efficiency_perc_max3000rpm, ROUND((t3.rendimento*100)::NUMERIC,2) AS
    efficiency_perc_over3000rpm
FROM
   (SELECT test_id AS id, AVG(power_kw)/ (AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
   WHERE engine_RPM_by_ecu_rpm < 2000
   GROUP BY test_id) t1
JOIN (
   SELECT test_id AS id, AVG(power_kw)/ (AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
   WHERE engine_RPM_by_ecu_rpm >= 2000 AND engine_RPM_by_ecu_rpm <= 3000
   GROUP BY test_id) t2 ON t1.id = t2.id
JOIN (
   SELECT test_id AS id, AVG(power_kw)/( AVG(fuel_g_s)/1000 * 458000) AS
      rendimento
   FROM acquisition_fact
   WHERE engine_RPM_by_ecu_rpm >= 3000
   GROUP BY test_id) t3 ON t1.id = t3.id;
```

2.2.4 Query M

Per ogni test, media di NOx, CO2, Potenza e Velocità

Questa interrogazione è stata formulata per analizzare le performance del sistema in condizioni sconvenienti. Più precisamente, si intende osservare il calo di prestazioni dovuto all'overhead necessario alla ricostruzione dell'informazione a partire dalle tabelle partizionate. Si intende quindi operare su 4 colonne distribuite su 2 tabelle ottenute dal partizionamento verticale illustrato nella sezione 1.2.

Senza partizionamento

```
SELECT F.test_id, AVG(F.co2_mass_g_s), AVG(F.nox_mass_g_s), AVG(F.power_kw),
    AVG(F.speed_km_h)
FROM acquisition_fact F
GROUP BY F.test_id
```

Con partizionamento

```
SELECT F.test_id, AVG(F.co2_mass_g_s), AVG(F.nox_mass_g_s), AVG(F.power_kw),
    AVG(F.speed_km_h)
FROM acquisition_fact_particles P, acquisition_fact_general G
WHERE P.row_id = G.row_id
GROUP BY F.test_id
```

Dimensione tab. fatti	Partizionamento	Tempo
570.600	No	6,194s
570.600	Si	24,715s
1.141.200	No	16,903s
1.141.200	Si	67,469s
1.426.500	No	24,141s
1.426.500	Si	98,218s

2.2.5 Query T

Media e deviazione standard delle temperature

Inversamente allo scopo della query proposta precedentemente, in questo caso si vuole misurare lo speedup raggiungibile grazie ad un buon partizionamento. Per questo motivo l'interrogazione richiede informazioni di una stessa categoria, memorizzate su di una tabella dedicata alle acquisizioni effettuate dall'Engine Control Unit (ECU).

Interrogazione sull'intero dataset

```
SELECT F.test_id, AVG(F.engine_coolant_temperature_by_ecu_degc),
    STDDEV(F.engine_coolant_temperature_by_ecu_degc),
    AVG(F.ambient_air_temperature_by_ecu_degc),
    STDDEV(F.ambient_air_temperature_by_ecu_degc),
    AVG(F.engine_oil_temperature_by_ecu_degc),
    STDDEV(F.engine_oil_temperature_by_ecu_degc)
FROM acquisition_fact F
GROUP BY F.test_id
```

Interrogazione su tabella ECU

```
SELECT E.test_id, AVG(E.engine_coolant_temperature_by_ecu_degc),
    STDDEV(E.engine_coolant_temperature_by_ecu_degc),
    AVG(E.ambient_air_temperature_by_ecu_degc),
    STDDEV(E.ambient_air_temperature_by_ecu_degc),
    AVG(E.engine_oil_temperature_by_ecu_degc),
    STDDEV(E.engine_oil_temperature_by_ecu_degc)
FROM acquisition_fact_ecu E
GROUP BY E.test_id
```

Dimensione tab. fatti	Partizionamento	Tempo
570.600	No	6,194s
570.600	Si	24,715s
1.141.200	No	16,903s
1.141.200	Si	67,469s
1.426.500	No	24,141s
1.426.500	Si	98,218s

Capitolo 3

Conclusioni

I risultati ottenuti sono chiaramente legati all'architettura hardware e software utilizzata per l'esecuzione delle procedure. Durante l'intero progetto è stata utilizzata una piattaforma con le seguenti specifiche:

- Xiaomi Notebook Air 13
- Intel Core i7
- SSD da 256GB
- RAM da 8GB
- Windows 10

I tempi sono necessariamente soggetti a limiti fisici e rumori dovuti a:

1. Capacità hardware del calcolatore
2. Scheduling del sistema operativo.
3. Parametri di configurazione del DBMS.

Per ammortizzare la perturbazione, le fasi di cronometraggio prevedono l'acquisizione di più campioni dello stesso tipo, dai quali si è poi ricavata la media.
I valori ottenuti vanno inoltre scalati rispetto alla dimensione del dataset. A tal proposito si è simulata una cardinalità massiccia di acquisizioni (circa 1.5 milioni).

A valle dell'analisi condotta si è arrivati alle seguenti riflessioni:

- Avere i dati precalcolati può avere un impatto notevole a favore dell'esecuzione della query, nonostante il numero di accessi alle pagine di memoria sia maggiore,

almeno quando il calcolo è relativamente complesso e dipendente da più misure. Di contro, avere dati precalcolati, porta ad un aumento generale dei tempi di ETL. Quindi in caso di aumento notevole delle dimensioni del database potrebbe risultare necessario, per soddisfare le deadline dell'ETL, utilizzare uno schema senza i dati precalcolati, o almeno mantenere come precalcolati solo quelli più utilizzati.

- Quando una gerarchia è basata su unità di misura fisiche (ad esempio distanza o tempo), può convenire effettuare operazioni di raggruppamento su espressioni calcolabili sui fatti, invece che utilizzare l'implementazione tradizionale che prevede una tabella separata per ogni gerarchia e utilizzo di join, che possono appesantire notevolmente il calcolo.

- Dal punto di vista dello spazio fisico occupato è stato osservato che, con circa due milioni di registrazioni nella tabella dei fatti, memorizzare i dati precalcolati porta ad un aumento di circa il 17.5% (da 639 a 744 MB). In un contesto di Data Warehouse un incremento del genere non può essere comunque considerato critico, soprattutto rispetto al guadagno in termini di tempo.

- L'utilizzo della vista materializzata ha mostrato, come previsto, un vantaggio notevole, con un rallentamento accettabile della fase di ETL di aggiornamento delle viste.

- Il partizionamento non ha mostrato evidenti miglioramenti che ne giustifichino l'utilizzo. Infatti, si è registrato un peggioramento netto della fase di ETL (in cui la durata dei passi 3 e 4 triplica) e delle query che prevedono la ricostruzione parziale o totale del fatto, avvantaggiando solamente (e naturalmente) le query circoscritte ad una singola partizione. Tuttavia, prima di screditare completamente questa tecnica, bisognerebbe prendere in considerazione che questa andrebbe valutata in contesti in cui il numero e l'importanza di query circoscritte a singole partizioni potrebbe effettivamente portare a dei benefici.

Appendice A

Sorgenti Java

Main.java

```
import ETL.Cleaner;
import ETL.Transformer;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.util.concurrent.TimeUnit;

public final class Main {

    private static final String LOCATION_EXCEL="./excel/test Napoli
        20-10-2017.xlsx";

    private static final String PATH="./excel/duplicates";

    private static final int NUM_TEST=15;

    private static final boolean DUPLICATE=true, CLEAN=true, TRANSFORM=true;

    public static void main(String[] args) {

        Multiplier multiplier=new Multiplier();
        Cleaner cleaner =new Cleaner();
```

```
Transformer transformer = new Transformer();
Long startTime;

//Duplicate files
if (DUPLICATE) {
    System.out.println("Duplico i file");
    startTime = System.currentTimeMillis();
    createDirectory(PATH);
    for (int i = 0; i < NUM_TEST; i++) {
        multiplier.setWorkbook(loadEXCEL());
        multiplier.duplicate(PATH);
    }
    System.out.println("Tempo impiegato per duplicare: "+
        TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis()-startTime)/(double)
}

//Clean files
if (CLEAN) {
    startTime = System.currentTimeMillis();
    System.out.println("Pulisco i file");
    createDirectory(PATH+"/csv");
    for (int i = 0; i < NUM_TEST; i++) {
        Workbook workbook = getWorkbook(i);
        if (workbook != null) {
            cleaner.setWorkbook(workbook);
            cleaner.setCsvFile(createCSV(workbook.getSheetName(0)));
            cleaner.clean();
        }
    }
    System.out.println("Tempo impiegato per pulire: "+
        TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis()-startTime)/(double)
}


//Transform files
if (TRANSFORM) {
    System.out.println("Transformo i file");
    startTime = System.currentTimeMillis();
    for (int i = 0; i < NUM_TEST; i++) {
        BufferedReader reader = loadCSV(i);
```

```
            if (reader != null) {
                transformer.setReader(reader);
                transformer.transform(i+1);
                transformer.setCsvFile(createCSV(i));
            }
        }
        System.out.println("Tempo impiegato per trasformare: "+
            TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis()-startTime)/(double)
    }
}


private static PrintWriter createCSV(String file){
    try {
       return new PrintWriter(PATH+"/csv/"+file+".csv",
          StandardCharsets.UTF_8);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

private static PrintWriter createCSV(int pos){
    File folder=new File(PATH+"/csv/");
    try {
        return new
            PrintWriter(folder.listFiles()[pos].toPath().toString(),StandardCharsets.UTF_8
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}

private static XSSFWorkbook loadEXCEL(){
    File excelFile=new File(LOCATION_EXCEL);
    try {
```

```
            return new XSSFWorkbook(excelFile);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InvalidFormatException e) {
            e.printStackTrace();
        }
        return null;
    }

    private static XSSFWorkbook getWorkbook(int pos){
        File folder=new File(PATH);
        try {
            if (pos<folder.listFiles().length &&
                folder.listFiles()[pos].isFile())
            return new XSSFWorkbook(folder.listFiles()[pos]);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InvalidFormatException e) {
            e.printStackTrace();
        }
        return null;
    }

    private static BufferedReader loadCSV(int pos){
        File folder=new File(PATH+"/csv/");
        try {
            if (pos<folder.listFiles().length &&
                folder.listFiles()[pos].isFile())
                return
                    Files.newBufferedReader(folder.listFiles()[pos].toPath());
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    private static void createDirectory(String path) {
        File file = new File(path);
        if (!file.exists())
            file.mkdir();
```

```
    }
}
```

Multiplier.java

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;

public final class Multiplier {

    private XSSFWorkbook workbook;

    public void setWorkbook(XSSFWorkbook workbook) {
        this.workbook = workbook;
    }


    public void duplicate(String path){
        Cell cell;
        String fictitiousDate;

        final Sheet sheet = workbook.getSheetAt(0);
        final int rowDim = sheet.getLastRowNum();
        Row currRow = sheet.getRow(0);
        final int colDim = currRow.getLastCellNum();

        currRow.createCell(colDim, CellType.STRING).setCellValue("Date");

        fictitiousDate = createRandomDate();
        for (int i = 2; i < rowDim; i++) {
            currRow = sheet.getRow(i);
            cell = currRow.createCell(colDim, CellType.STRING);
            cell.setCellValue(fictitiousDate);
```

```
        }

        workbook.setSheetName(0, fictitiousDate);
        try {
            workbook.write(new
                FileOutputStream(path+"/"+fictitiousDate+".xlsx"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private String createRandomDate(){
        // Get a new random instance, seeded from the clock
        Random rnd = new Random();

        // Get an Epoch value roughly between 2000 and 2018
        // -946684800000L = January 1, 2000
        // Add up to 18 years to it (using modulus on the next long)
        long ms = 946684800000L + (Math.abs(rnd.nextLong()) % (18L * 365 * 24
            * 60 * 60 * 1000));
        //LocalDate date =
            Instant.ofEpochMilli(ms).atZone(ZoneId.systemDefault()).toLocalDate();
        return new SimpleDateFormat("dd-MM-yyyy").format(new Date(ms));
    }


}
```

Cleaner.java

```
package ETL;

import org.apache.poi.ss.usermodel.*;

import java.io.PrintWriter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Iterator;
```

```
public final class Cleaner {

    private static final DataFormatter DATA_FORMATTER = new DataFormatter();

    private static final SimpleDateFormat SIMPLE_DATE_FORMAT= new
        SimpleDateFormat("hh:mm:ss");


    private PrintWriter csvFile;

    private Workbook workbook;

    private Sheet sheet;

    private FormulaEvaluator evaluator;

    public void setCsvFile(PrintWriter csvFile) {
        this.csvFile = csvFile;
    }

    public void setWorkbook(Workbook workbook) {
        this.workbook = workbook;
    }

    public void clean(){
        evaluator= workbook.getCreationHelper().createFormulaEvaluator();
        Row lastRow=null, row;
        Iterator<Row> rowIterator=workbook.getSheetAt(0).rowIterator();
        metaDataOnCSV(rowIterator.next(),rowIterator.next());
        for (Sheet actualSheet:workbook) {
            //Remove bad rows from each sheet
            sheet=actualSheet;
            removeBadRows();

            //Skip header
            rowIterator=sheet.rowIterator();
            rowIterator.next();
            rowIterator.next();
```

```
            //Process Row data
            while (rowIterator.hasNext()) {
                row=rowIterator.next();
                processRow(row,lastRow);
            }
        }
        csvFile.close();
    }

    private void metaDataOnCSV(Row row1, Row row2){
        StringBuilder builder=new StringBuilder();
        for (int i = 0; i < row1.getLastCellNum(); i++) {
            String
                temp=DATA_FORMATTER.formatCellValue(row1.getCell(i)).toLowerCase();
            temp=temp.replace('
                ','_').replace('/','_').replace(".","").replace('(','_').replace(')','_');
            builder.append(temp);
            if (i!=0 && i!=6 && i!=row1.getLastCellNum()-1)
                builder.append("_");
            if (i==23)
                builder.append("kw");
            else
                builder.append(DATA_FORMATTER.formatCellValue(row2.getCell(i)).toLowerCase().
            if (i< row1.getLastCellNum()-1)
                builder.append(';');
        }
        csvFile.println(builder.toString().replace("__","_"));
    }

    private void removeBadRows(){
        sheet.removeRow(sheet.getRow(sheet.getLastRowNum()));
        sheet.removeRow(sheet.getRow(sheet.getLastRowNum()));
    }

    private void processRow(Row row, Row lastRow){
        StringBuilder builder=new StringBuilder();
        String value;
        for (int i = 0; i < row.getLastCellNum(); i++) {
            Cell currentCell=row.getCell(i);
```

```
if (currentCell!=null && currentCell.getCellTypeEnum() ==
    CellType.FORMULA)
   value=
      Double.toString(evaluator.evaluate(currentCell).getNumberValue());
else
    value=DATA_FORMATTER.formatCellValue(currentCell);
//CHECK ABSOLUTE
if (i==0 && value.isEmpty() && lastRow!=null){
    value=DATA_FORMATTER.formatCellValue(lastRow.getCell(0));
    if (value.isEmpty())
        return;
    try {
        value=addSecond(value);
        row.createCell(0,CellType.STRING).setCellValue(value);

    } catch (ParseException e) {
        e.printStackTrace();
        return;
    }
}

//CHECK DISTANCE
if (i==31 && value.isEmpty() && lastRow!=null){
    value=DATA_FORMATTER.formatCellValue(lastRow.getCell(31));
    if (value.isEmpty())
        break;

    double dist=Double.valueOf(value);

    value=DATA_FORMATTER.formatCellValue(row.getCell(17));

    if (value.isEmpty())
        break;
    double vel=Double.valueOf(value);
    vel/=3.6;

    value=String.valueOf(dist+vel);
    currentCell.setCellValue(value);
}
```

```
            //CHECK NEGATIVE VALUE
            if  (i!=11 && i!=16 && !value.isEmpty() &&
                value.toCharArray()[0]=='-')
                return;
          //  if (i!=15 && i!=16)
                value=value.replace(',','.');
            builder.append(value);
            if (i< row.getLastCellNum()-1)
                builder.append(';');
        }
        csvFile.println(builder.toString());
    }

    private String addSecond(String value) throws ParseException{
        Date date= SIMPLE_DATE_FORMAT.parse(value);
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        cal.add(Calendar.SECOND,1);
        return SIMPLE_DATE_FORMAT.format(cal.getTime());
    }
}
```

Transformer.java

```
package ETL;

import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Map;

public final class Transformer {
```

```
private static final SimpleDateFormat SIMPLE_DATE_FORMAT= new
    SimpleDateFormat("hh:mm");

private BufferedReader reader;

private CSVParser csvParser;

private String exportedCsv;

private StringBuilder builder;

public void setReader(BufferedReader reader) {
    this.reader = reader;
}

public void transform(int j){
    builder=new StringBuilder();

    try {
        csvParser= new CSVParser(reader,
            CSVFormat.DEFAULT.withDelimiter(';').withHeader().withIgnoreHeaderCase());
        Map<String,Integer> mapHeader=csvParser.getHeaderMap();
        int size =csvParser.getHeaderMap().keySet().size();
        for (int i = 0; i < size-1; i++) {
            for (String headerString: mapHeader.keySet()) {
                if (mapHeader.get(headerString)==i) {
                    builder.append(headerString);
                    if (i<size-2)
                        builder.append(';');
                }
            }
        }
        //Remove latitude and longitude from header
        String header=builder.toString();
        header=header.replace("latitude_n_s;","");
        header=header.replace("longitude_w_e;","");
        header=header.replace("absolute","time");

        //Add header to final CSV
```

```
            builder=new StringBuilder();
            builder.append("test_id;");
            builder.append("date;");
            builder.append(header);
            builder.append('\n');
            List<CSVRecord>records= csvParser.getRecords();
            String time=cleanTime(records.get(0).get(0));
            size--;
            for (CSVRecord record: records){
                //setTestID
                builder.append(j);
                builder.append(';');
                //setDateTest
                builder.append(record.get(size));
                builder.append(';');
                //setTimeTest
                builder.append(time);
                builder.append(';');

                //set others
                for (int i = 1; i < size; i++) {

                    //Remove latitude and longitude from every row
                    if (i==14 || i==15)
                        continue;

                    builder.append(record.get(i));
                    if (i<size-1)
                        builder.append(';');
                }
                builder.append('\n');
            }
            csvParser.close();
            reader.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    private String cleanTime(String value) {
        Date date= null;
        try {
            date = SIMPLE_DATE_FORMAT.parse(value);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return SIMPLE_DATE_FORMAT.format(date)+":00";
    }

    public void setCsvFile(PrintWriter csvFile){
        String temp=builder.toString();
        csvFile.print(temp.substring(0,temp.length()-1));
        csvFile.close();
    }
}
```

Appendice B

Sorgenti SQL

createFunction.sql

```
--Function to switch index status before and after insert--
CREATE OR REPLACE FUNCTION set_index (enable boolean) RETURNS VOID AS
  $$
  BEGIN
    IF(enable= true) THEN
      CREATE INDEX if not exists acquisition_fact_date_index on
          acquisition_fact using hash(date_key);
      CREATE INDEX if not exists acquisition_fact_step_index on
          acquisition_fact using hash(step_key);
      CREATE INDEX if not exists acquisition_fact_km_index on acquisition_fact
          using hash(km_key);
    ELSE
      DROP INDEX IF EXISTS acquisition_fact_date_index;
      DROP INDEX IF EXISTS acquisition_fact_step_index;
      DROP INDEX IF EXISTS acquisition_fact_km_index;
    END IF;
  END;
  $$
  LANGUAGE plpgsql;

--Function to extract the season from month--
CREATE OR REPLACE FUNCTION get_season(month smallint) RETURNS seasons AS
  $$
  BEGIN
    RETURN (
          CASE
```

```
            WHEN month in (1,2,12) THEN
              'winter'
            WHEN month in (3,4,5) THEN
              'spring'
            WHEN month in (6,7,8) THEN
              'summer'
            ELSE
              'autumn'
            END
    );
  END;
  $$ LANGUAGE plpgsql;

--Function to populate temp_table with CSV records
CREATE OR REPLACE FUNCTION import_records_temp_table(file_path TEXT) RETURNS
    VOID AS
  $$
  DECLARE fn_i TEXT; -- Variable to hold name of current CSV file being
      inserted
  DECLARE mytable TEXT; -- Variable to hold name of table to insert data into
  BEGIN
    mytable:='temp_table';
    DROP TABLE IF EXISTS files;
    CREATE TEMP TABLE files AS
      SELECT file_path || pg_ls_dir AS fn -- get all of the files in the
          directory, prepending with file path
      FROM pg_ls_dir(file_path);

    --insert csv records into temp_table
    LOOP
      fn_i := (select fn from files limit 1); -- Pick the first file
      raise notice 'fn: %', fn_i;
      EXECUTE 'COPY ' || mytable || ' FROM ''' || fn_i || ''' DELIMITER '';''
          CSV HEADER';
      DELETE FROM files WHERE fn = fn_i; -- Delete the file just inserted from
          the queue
      EXIT WHEN (SELECT COUNT(*) FROM files) = 0;
    END LOOP;

  END;
```

```
  $$
  LANGUAGE plpgsql;

--Function to populate DataMart--
CREATE OR REPLACE FUNCTION import_records_into_data_mart(file_path TEXT)
    RETURNS VOID AS
  $$
  BEGIN
  --PERFORM import_records_temp_table(file_path);--
  PERFORM set_index(false);

  --populate Fact Table--
  INSERT INTO acquisition_fact (
                              test_id,
                              date_key,
                              step_key,
                              km_key,
                              co_concentration_vol_per ,
                              co2_concentration_vol_perc ,
                              nox_concentration_ppm ,
                              h2O_conc_vol_perc ,
                              a_f  ,
                              exh_flow_corr_m3_min ,
                              exh_flow_corr_m3_s ,
                              exh_Temp_degC ,
                              exh_press_kPa ,
                              amb_temp_degC ,
                              amb_press_kPa ,
                              amb_humid_RH ,
                              altitude_m ,
                              velocity_km_h ,
                              battery_V ,
                              co_mass_g_s ,
                              co2_mass_g_s ,
                              nox_mass_g_s ,
                              fuel_g_s ,
                              power_kW ,
                              speed_km_h ,
                              voltage_V ,
                              engine_coolant_temperature_by_ecu_degC ,
```

```
                        fuel_pressure_by_ecu_kPa ,
                        engine_RPM_by_ecu_rpm ,
                        vehicle_speed_by_ecu_km_h ,
                        vehicle_speed_by_ecu_m_s ,
                        distance_m ,
                        intake_air_temperature_by_ecu_degC ,
                        maf_air_flow_rate_by_ecu_grams_sec ,
                        maf_air_flow_rate_by_ecu_m3_s ,
                        fuel_rail_pressure_by_ecu_kPa ,
                        fuel_rail_pressure_diesel_by_ecu_kPa ,
                        commande_EGR_by_ecu_perc ,
                        barometric_pressure_by_ecu_kPa ,
                        ambient_air_temperature_by_ecu_degC ,
                        fuel_rail_pressure_Time_by_ecu_kPa ,
                        engine_oil_temperature_by_ecu_degC ,
                        engine_fuel_rate_by_ecu_L_h ,
                        actual_engine_by_ecu_perc ,
                        engine_reference_torque_by_ecu_Nm )
SELECT      T.test_id,
            T.date,
            T.relative_s,
            (T.distance_m::int/1000),
            T.co_concentration_vol_per ,
            T.co2_concentration_vol_perc ,
            T.nox_concentration_ppm ,
            T.h2O_conc_vol_perc ,
            T.a_f  ,
            T.exh_flow_corr_m3_min ,
            T.exh_flow_corr_m3_s ,
            T.exh_Temp_degC ,
            T.exh_press_kPa ,
            T.amb_temp_degC ,
            T.amb_press_kPa ,
            T.amb_humid_RH ,
            T.altitude_m ,
            T.velocity_km_h ,
            T.battery_V ,
            T.co_mass_g_s ,
            T.co2_mass_g_s ,
            T.nox_mass_g_s ,
```

```
            T.fuel_g_s ,
            T.power_kW ,
            T.speed_km_h ,
            T.voltage_V ,
            T.engine_coolant_temperature_by_ecu_degC ,
            T.fuel_pressure_by_ecu_kPa ,
            T.engine_RPM_by_ecu_rpm ,
            T.vehicle_speed_by_ecu_km_h ,
            T.vehicle_speed_by_ecu_m_s ,
            T.distance_m ,
            T.intake_air_temperature_by_ecu_degC ,
            T.maf_air_flow_rate_by_ecu_grams_sec ,
            T.maf_air_flow_rate_by_ecu_m3_s ,
            T.fuel_rail_pressure_by_ecu_kPa ,
            T.fuel_rail_pressure_diesel_by_ecu_kPa ,
            T.commande_EGR_by_ecu_perc ,
            T.barometric_pressure_by_ecu_kPa ,
            T.ambient_air_temperature_by_ecu_degC ,
            T.fuel_rail_pressure_Time_by_ecu_kPa ,
            T.engine_oil_temperature_by_ecu_degC ,
            T.engine_fuel_rate_by_ecu_L_h ,
            T.actual_engine_by_ecu_perc ,
            T.engine_reference_torque_by_ecu_Nm
  FROM temp_table as T;

--populate Distance Hierarchy--
  INSERT INTO distance_hierarchy
  SELECT distinct T.distance_m::int /1000 , T.distance_m::int/5000,
      T.distance_m::int/250000, T.distance_m::int/500000
  FROM temp_table T
  WHERE T.distance_m::int /1000 NOT IN (SELECT U.km_key FROM
      distance_hierarchy U);

--populate Date Hierarchy--
  INSERT INTO date_hierarchy
  SELECT DISTINCT T.date, extract(month from T.date),
      get_season(extract(month from T.date)::smallint) ,extract(YEAR from
      T.date)
  FROM temp_table T
  WHERE T.date NOT IN (SELECT U.date_key FROM date_hierarchy U);
```

```
    --populate Time Hierarchy--
    INSERT INTO time_hierarchy
    SELECT distinct T.relative_s, T.relative_s/60, T.relative_s/3600
    FROM temp_table T;

  TRUNCATE TABLE temp_table;

  --refresh mv--
   -- refresh materialized view efficiency_compare_rpm;
  --  refresh materialized view fuel_compare_speed;

    --enable index--
  PERFORM set_index(true);

  END;
  $$
  LANGUAGE plpgsql;
```

createFunctionPart.sql

```
--Function to switch index status before and after insert--
CREATE OR REPLACE FUNCTION set_index (enable boolean) RETURNS VOID AS
  $$
  BEGIN
    IF(enable= true) THEN
      CREATE INDEX if not exists acquisition_fact_general_date_index on
          acquisition_fact_general using hash(date_key);
      CREATE INDEX if not exists acquisition_fact_general_step_index on
          acquisition_fact_general using hash(step_key);
      CREATE INDEX if not exists acquisition_fact_general_km_index on
          acquisition_fact_general using hash(km_key);

      CREATE INDEX if not exists acquisition_fact_flow_date_index on
          acquisition_fact_flow using hash(date_key);
      CREATE INDEX if not exists acquisition_fact_flow_step_index on
          acquisition_fact_flow using hash(step_key);
      CREATE INDEX if not exists acquisition_fact_flow_km_index on
          acquisition_fact_flow using hash(km_key);
```

```
    CREATE INDEX if not exists acquisition_fact_particles_date_index on
        acquisition_fact_particles using hash(date_key);
    CREATE INDEX if not exists acquisition_fact_particles_step_index on
        acquisition_fact_particles using hash(step_key);
    CREATE INDEX if not exists acquisition_fact_particles_km_index on
        acquisition_fact_particles using hash(km_key);

    CREATE INDEX if not exists acquisition_fact_ecu_date_index on
        acquisition_fact_ecu using hash(date_key);
    CREATE INDEX if not exists acquisition_fact_ecu_step_index on
        acquisition_fact_ecu using hash(step_key);
    CREATE INDEX if not exists acquisition_fact_ecu_km_index on
        acquisition_fact_ecu using hash(km_key);

  ELSE
    DROP INDEX IF EXISTS acquisition_fact_general_date_index;
    DROP INDEX IF EXISTS acquisition_fact_general_step_index;
    DROP INDEX IF EXISTS acquisition_fact_general_km_index;

    DROP INDEX IF EXISTS acquisition_fact_flow_date_index;
    DROP INDEX IF EXISTS acquisition_fact_flow_step_index;
    DROP INDEX IF EXISTS acquisition_fact_flow_km_index;

    DROP INDEX IF EXISTS acquisition_fact_particles_date_index;
    DROP INDEX IF EXISTS acquisition_fact_particles_step_index;
    DROP INDEX IF EXISTS acquisition_fact_particles_km_index;

    DROP INDEX IF EXISTS acquisition_fact_ecu_date_index;
    DROP INDEX IF EXISTS acquisition_fact_ecu_step_index;
    DROP INDEX IF EXISTS acquisition_fact_ecu_km_index;
  END IF;
END;
$$
LANGUAGE plpgsql;

--Function to extract the season from month--
CREATE OR REPLACE FUNCTION get_season(month smallint) RETURNS seasons AS
  $$
  BEGIN
    RETURN (
```

```
        CASE
          WHEN month in (1,2,12) THEN
            'winter'
          WHEN month in (3,4,5) THEN
            'spring'
          WHEN month in (6,7,8) THEN
            'summer'
          ELSE
            'autumn'
          END
    );
  END;
  $$ LANGUAGE plpgsql;

--Function to populate temp_table with CSV records
CREATE OR REPLACE FUNCTION import_records_temp_table(file_path TEXT) RETURNS
    VOID AS
  $$
  DECLARE fn_i TEXT; -- Variable to hold name of current CSV file being
      inserted
  DECLARE mytable TEXT; -- Variable to hold name of table to insert data into
  BEGIN
    mytable:='temp_table';
    DROP TABLE IF EXISTS files;
    CREATE TEMP TABLE files AS
      SELECT file_path || pg_ls_dir AS fn -- get all of the files in the
          directory, prepending with file path
      FROM pg_ls_dir(file_path);

    --insert csv records into temp_table
    LOOP
      fn_i := (select fn from files limit 1); -- Pick the first file
      raise notice 'fn: %', fn_i;
      EXECUTE 'COPY ' || mytable || ' FROM ''' || fn_i || ''' DELIMITER '';''
          CSV HEADER';
      DELETE FROM files WHERE fn = fn_i; -- Delete the file just inserted from
          the queue
      EXIT WHEN (SELECT COUNT(*) FROM files) = 0;
    END LOOP;
```

```
  END;
  $$
  LANGUAGE plpgsql;

--Function to populate DataMart--
CREATE OR REPLACE FUNCTION import_records_into_data_mart(file_path TEXT)
    RETURNS VOID AS
  $$
  BEGIN
  --PERFORM import_records_temp_table(file_path);--
  PERFORM set_index(false);

  --populate Fact General Table--
  INSERT INTO acquisition_fact_general (
                              test_id,
                              date_key,
                              step_key,
                              km_key,
                              amb_temp_degC ,
                              amb_press_kPa ,
                              amb_humid_RH ,
                              altitude_m ,
                              velocity_km_h ,
                              battery_V ,
                              fuel_g_s ,
                              power_kW ,
                              speed_km_h ,
                              voltage_V ,
                              distance_m )
    SELECT    T.test_id,
              T.date,
              T.relative_s,
              (T.distance_m::int/1000),
              T.amb_temp_degC ,
              T.amb_press_kPa ,
              T.amb_humid_RH ,
              T.altitude_m ,
              T.velocity_km_h ,
              T.battery_V ,
              T.fuel_g_s ,
```

```
            T.power_kW ,
            T.speed_km_h ,
            T.voltage_V ,
            T.distance_m
  FROM temp_table as T;

--populate Fact Flow Table--
 INSERT INTO acquisition_fact_flow (
                            test_id,
                            date_key,
                            step_key,
                            km_key,
                            a_f  ,
                            exh_flow_corr_m3_min ,
                            exh_flow_corr_m3_s ,
                            exh_Temp_degC ,
                            exh_press_kPa )
  SELECT    T.test_id,
            T.date,
            T.relative_s,
            (T.distance_m::int/1000),

            T.a_f  ,
            T.exh_flow_corr_m3_min ,
            T.exh_flow_corr_m3_s ,
            T.exh_Temp_degC ,
            T.exh_press_kPa
  FROM temp_table as T;

--populate Fact Particles Table--
 INSERT INTO acquisition_fact_particles (
                            test_id,
                            date_key,
                            step_key,
                            km_key,
                            co_concentration_vol_per ,
                            co2_concentration_vol_perc ,
                            nox_concentration_ppm ,
                            h2O_conc_vol_perc ,
```

```
                            co_mass_g_s ,
                            co2_mass_g_s ,
                            nox_mass_g_s )
  SELECT    T.test_id,
            T.date,
            T.relative_s,
            (T.distance_m::int/1000),
            T.co_concentration_vol_per ,
            T.co2_concentration_vol_perc ,
            T.nox_concentration_ppm ,
            T.h2O_conc_vol_perc ,
            T.co_mass_g_s ,
            T.co2_mass_g_s ,
            T.nox_mass_g_s
  FROM temp_table as T;

   --populate Fact ECU Table--
INSERT INTO acquisition_fact_ecu (
                            test_id,
                            date_key,
                            step_key,
                            km_key,
                            engine_coolant_temperature_by_ecu_degC ,
                            fuel_pressure_by_ecu_kPa ,
                            engine_RPM_by_ecu_rpm ,
                            vehicle_speed_by_ecu_km_h ,
                            vehicle_speed_by_ecu_m_s ,
                            intake_air_temperature_by_ecu_degC ,
                            maf_air_flow_rate_by_ecu_grams_sec ,
                            maf_air_flow_rate_by_ecu_m3_s ,
                            fuel_rail_pressure_by_ecu_kPa ,
                            fuel_rail_pressure_diesel_by_ecu_kPa ,
                            commande_EGR_by_ecu_perc ,
                            barometric_pressure_by_ecu_kPa ,
                            ambient_air_temperature_by_ecu_degC ,
                            fuel_rail_pressure_Time_by_ecu_kPa ,
                            engine_oil_temperature_by_ecu_degC ,
                            engine_fuel_rate_by_ecu_L_h ,
                            actual_engine_by_ecu_perc ,
                            engine_reference_torque_by_ecu_Nm )
```

```
  SELECT    T.test_id,
            T.date,
            T.relative_s,
            (T.distance_m::int/1000),
            T.engine_coolant_temperature_by_ecu_degC ,
            T.fuel_pressure_by_ecu_kPa ,
            T.engine_RPM_by_ecu_rpm ,
            T.vehicle_speed_by_ecu_km_h ,
            T.vehicle_speed_by_ecu_m_s ,
            T.intake_air_temperature_by_ecu_degC ,
            T.maf_air_flow_rate_by_ecu_grams_sec ,
            T.maf_air_flow_rate_by_ecu_m3_s ,
            T.fuel_rail_pressure_by_ecu_kPa ,
            T.fuel_rail_pressure_diesel_by_ecu_kPa ,
            T.commande_EGR_by_ecu_perc ,
            T.barometric_pressure_by_ecu_kPa ,
            T.ambient_air_temperature_by_ecu_degC ,
            T.fuel_rail_pressure_Time_by_ecu_kPa ,
            T.engine_oil_temperature_by_ecu_degC ,
            T.engine_fuel_rate_by_ecu_L_h ,
            T.actual_engine_by_ecu_perc ,
            T.engine_reference_torque_by_ecu_Nm
  FROM temp_table as T;

--populate Distance Hierarchy--
  INSERT INTO distance_hierarchy
  SELECT distinct T.distance_m::int /1000 , T.distance_m::int/5000,
     T.distance_m::int/250000, T.distance_m::int/500000
  FROM temp_table T
  WHERE T.distance_m::int /1000 NOT IN (SELECT U.km_key FROM
     distance_hierarchy U);

--populate Date Hierarchy--
  INSERT INTO date_hierarchy
  SELECT DISTINCT T.date, extract(month from T.date),
     get_season(extract(month from T.date)::smallint) ,extract(YEAR from
     T.date)
  FROM temp_table T
  WHERE T.date NOT IN (SELECT U.date_key FROM date_hierarchy U);
```

```
    --populate Time Hierarchy--
    INSERT INTO time_hierarchy
    SELECT distinct T.relative_s, T.relative_s/60, T.relative_s/3600
    FROM temp_table T;

  TRUNCATE TABLE temp_table;
  --PERFORM set_index(true);
  END;
  $$
  LANGUAGE plpgsql;
```

createTable.sql

```
--Temporary table to store csv records--
CREATE TABLE if not exists temp_table(
  test_id int,
  date Date,
  time Time,
  relative_s int,
  co_concentration_vol_per double precision,
  co2_concentration_vol_perc double precision,
  nox_concentration_ppm double precision,
  h2O_conc_vol_perc double precision,
  a_f double precision,
  exh_flow_corr_m3_min double precision,
  exh_flow_corr_m3_s double precision,
  exh_Temp_degC double precision,
  exh_press_kPa double precision,
  amb_temp_degC double precision,
  amb_press_kPa double precision,
  amb_humid_RH double precision,
  altitude_m double precision,
  velocity_km_h double precision,
  battery_V double precision,
  co_mass_g_s double precision,
  co2_mass_g_s double precision,
  nox_mass_g_s double precision,
  fuel_g_s double precision,
  power_kW double precision,
```

```
  speed_km_h double precision,
  voltage_V double precision,
  engine_coolant_temperature_by_ecu_degC double precision,
  fuel_pressure_by_ecu_kPa double precision,
  engine_RPM_by_ecu_rpm double precision,
  vehicle_speed_by_ecu_km_h double precision,
  vehicle_speed_by_ecu_m_s double precision,
  distance_m double precision,
  intake_air_temperature_by_ecu_degC double precision,
  maf_air_flow_rate_by_ecu_grams_sec double precision,
  maf_air_flow_rate_by_ecu_m3_s double precision,
  fuel_rail_pressure_by_ecu_kPa double precision,
  fuel_rail_pressure_diesel_by_ecu_kPa double precision,
  commande_EGR_by_ecu_perc double precision,
  barometric_pressure_by_ecu_kPa double precision,
  ambient_air_temperature_by_ecu_degC double precision,
  fuel_rail_pressure_Time_by_ecu_kPa double precision,
  engine_oil_temperature_by_ecu_degC double precision,
  engine_fuel_rate_by_ecu_L_h double precision,
  actual_engine_by_ecu_perc double precision,
  engine_reference_torque_by_ecu_Nm double precision
);

--Season ENUM--
DROP TYPE if exists seasons;
CREATE TYPE seasons as ENUM('autumn','winter','spring','summer');

--Date Hierarchy Table--
CREATE TABLE if not exists date_hierarchy(
  date_key Date UNIQUE ,
  month smallint,
  season seasons,
  year smallint
);

--Time Hierarchy Table--
CREATE TABLE if not exists time_hierarchy(
  step_key int,
  minute smallint,
  hour smallint
```

```
);

--Distance Hierarchy Table--
CREATE TABLE if not exists distance_hierarchy(
  km_key int UNIQUE,
  km_5 int,
  km_25 int,
  km_50 int
);

--Acquisition Fact Table--
CREATE TABLE if not exists acquisition_fact(
  test_id int,
  date_key date,
  step_key int,
  km_key int,
  co_concentration_vol_per double precision,
  co2_concentration_vol_perc double precision,
  nox_concentration_ppm double precision,
  h2O_conc_vol_perc double precision,
  a_f double precision,
  exh_flow_corr_m3_min double precision,
  exh_flow_corr_m3_s double precision,
  exh_Temp_degC double precision,
  exh_press_kPa double precision,
  amb_temp_degC double precision,
  amb_press_kPa double precision,
  amb_humid_RH double precision,
  altitude_m double precision,
  velocity_km_h double precision,
  battery_V double precision,
  co_mass_g_s double precision,
  co2_mass_g_s double precision,
  nox_mass_g_s double precision,
  fuel_g_s double precision,
  power_kW double precision,
  speed_km_h double precision,
  voltage_V double precision,
  engine_coolant_temperature_by_ecu_degC double precision,
  fuel_pressure_by_ecu_kPa double precision,
```

```
  engine_RPM_by_ecu_rpm double precision,
  vehicle_speed_by_ecu_km_h double precision,
  vehicle_speed_by_ecu_m_s double precision,
  distance_m double precision,
  intake_air_temperature_by_ecu_degC double precision,
  maf_air_flow_rate_by_ecu_grams_sec double precision,
  maf_air_flow_rate_by_ecu_m3_s double precision,
  fuel_rail_pressure_by_ecu_kPa double precision,
  fuel_rail_pressure_diesel_by_ecu_kPa double precision,
  commande_EGR_by_ecu_perc double precision,
  barometric_pressure_by_ecu_kPa double precision,
  ambient_air_temperature_by_ecu_degC double precision,
  fuel_rail_pressure_Time_by_ecu_kPa double precision,
  engine_oil_temperature_by_ecu_degC double precision,
  engine_fuel_rate_by_ecu_L_h double precision,
  actual_engine_by_ecu_perc double precision,
  engine_reference_torque_by_ecu_Nm double precision
);

--Index for fact table--
CREATE INDEX if not exists acquisition_fact_date_index on acquisition_fact
    using hash(date_key);

CREATE INDEX if not exists acquisition_fact_step_index on acquisition_fact
    using hash(step_key);

CREATE INDEX if not exists acquisition_fact_km_index on acquisition_fact using
    hash(km_key);


--Materialized View for Query 2--
CREATE MATERIALIZED VIEW IF NOT EXISTS fuel_compare_speed AS
  SELECT t1.id, t1.fuel_litres as consumo_litri_meno_di_50km_h, t2.fuel_litres
      as consumo_litri_meno_di_90km_h, t3.fuel_litres as
      consumo_litri_meno_di_130km_h
  FROM
       (SELECT test_id as id, round(SUM((fuel_g_s)/1000)::numeric/0.8,2) AS
          fuel_litres, round(AVG(velocity_km_h)::numeric,1) as
          average_speed_km_h
        FROM acquisition_fact
```

```
        where speed_km_h <=50
        GROUP BY test_id) t1
         JOIN (
             SELECT test_id as id, round(SUM((fuel_g_s)/1000)::numeric/0.8,2)
                AS fuel_litres, round(AVG(velocity_km_h)::numeric,1) as
                average_speed_km_h
             FROM acquisition_fact
             where speed_km_h >=50 and speed_km_h<=90
             GROUP BY test_id
             ) t2
           ON t1.id=t2.id
         JOIN (
             SELECT test_id as id , round(SUM((fuel_g_s)/1000)::numeric/0.8,2)
                AS fuel_litres, round(AVG(velocity_km_h)::numeric,1) as
                average_speed_km_h
             FROM acquisition_fact
             where speed_km_h >=90
             GROUP BY test_id
             )t3
           ON t1.id=t3.id;


--Materialized View for Query 3--
CREATE MATERIALIZED VIEW IF NOT EXISTS efficiency_compare_rpm AS
  SELECT t1.id, round((t1.rendimento*100)::numeric, 2) as
      efficiency_perc_max2000rpm, round((t2.rendimento*100)::numeric,2) as
      efficiency_perc_max3000rpm, round((t3.rendimento*100)::numeric,2) as
      efficiency_perc_over3000rpm
  FROM
       (SELECT test_id as id, avg(power_kw)/ (avg(fuel_g_s)/1000 * 458000) as
          rendimento
        FROM acquisition_fact
        where engine_RPM_by_ecu_rpm <2000
        GROUP BY test_id) t1
         JOIN (
             SELECT test_id as id, avg(power_kw)/ (avg(fuel_g_s)/1000 *
                458000) as rendimento
             FROM acquisition_fact
             where engine_RPM_by_ecu_rpm >=2000 and engine_RPM_by_ecu_rpm
                <=3000
```

```
            GROUP BY test_id
            ) t2
          ON  t1.id=t2.id
        JOIN (
            SELECT test_id as id, avg(power_kw)/( avg(fuel_g_s)/1000 *
               458000) as rendimento
            FROM acquisition_fact
            where engine_RPM_by_ecu_rpm >=3000
            GROUP BY test_id
            )t3
          ON t1.id=t3.id;
```

createTablePart.sql

```
--Temporary table to store csv records--
CREATE TABLE if not exists temp_table(
  test_id int,
  date Date,
  time Time,
  relative_s int,
  co_concentration_vol_per double precision,
  co2_concentration_vol_perc double precision,
  nox_concentration_ppm double precision,
  h2O_conc_vol_perc double precision,
  a_f double precision,
  exh_flow_corr_m3_min double precision,
  exh_flow_corr_m3_s double precision,
  exh_Temp_degC double precision,
  exh_press_kPa double precision,
  amb_temp_degC double precision,
  amb_press_kPa double precision,
  amb_humid_RH double precision,
  altitude_m double precision,
  velocity_km_h double precision,
  battery_V double precision,
  co_mass_g_s double precision,
  co2_mass_g_s double precision,
  nox_mass_g_s double precision,
  fuel_g_s double precision,
```

```
  power_kW double precision,
  speed_km_h double precision,
  voltage_V double precision,
  engine_coolant_temperature_by_ecu_degC double precision,
  fuel_pressure_by_ecu_kPa double precision,
  engine_RPM_by_ecu_rpm double precision,
  vehicle_speed_by_ecu_km_h double precision,
  vehicle_speed_by_ecu_m_s double precision,
  distance_m double precision,
  intake_air_temperature_by_ecu_degC double precision,
  maf_air_flow_rate_by_ecu_grams_sec double precision,
  maf_air_flow_rate_by_ecu_m3_s double precision,
  fuel_rail_pressure_by_ecu_kPa double precision,
  fuel_rail_pressure_diesel_by_ecu_kPa double precision,
  commande_EGR_by_ecu_perc double precision,
  barometric_pressure_by_ecu_kPa double precision,
  ambient_air_temperature_by_ecu_degC double precision,
  fuel_rail_pressure_Time_by_ecu_kPa double precision,
  engine_oil_temperature_by_ecu_degC double precision,
  engine_fuel_rate_by_ecu_L_h double precision,
  actual_engine_by_ecu_perc double precision,
  engine_reference_torque_by_ecu_Nm double precision
);

--Season ENUM--
DROP TYPE if exists seasons;
CREATE TYPE seasons as ENUM('autumn','winter','spring','summer');

--Date Hierarchy Table--
CREATE TABLE if not exists date_hierarchy(
  date_key Date UNIQUE ,
  month smallint,
  season seasons,
  year smallint
);

--Time Hierarchy Table--
CREATE TABLE if not exists time_hierarchy(
  step_key int,
  minute smallint,
```

```
  hour smallint
);

--Distance Hierarchy Table--
CREATE TABLE if not exists distance_hierarchy(
  km_key int UNIQUE,
  km_5 int,
  km_25 int,
  km_50 int
);

--Acquisition Fact General Table--
CREATE TABLE if not exists acquisition_fact_general(
  row_id bigserial,
  test_id int,
  date_key date,
  step_key int,
  km_key int,
  amb_temp_degC double precision,
  amb_press_kPa double precision,
  amb_humid_RH double precision,
  altitude_m double precision,
  velocity_km_h double precision,
  battery_V double precision,
  fuel_g_s double precision,
  power_kW double precision,
  speed_km_h double precision,
  voltage_V double precision,
  distance_m double precision
);

--Acquisition Fact Flow Table--
CREATE TABLE if not exists acquisition_fact_flow(
  row_id bigserial,
  test_id int,
  date_key date,
  step_key int,
  km_key int,
  a_f double precision,
  exh_flow_corr_m3_min double precision,
```

```
  exh_flow_corr_m3_s double precision,
  exh_Temp_degC double precision,
  exh_press_kPa double precision
);

--Acquisition Fact Particles Table--
CREATE TABLE if not exists acquisition_fact_particles(
  row_id bigserial,
  test_id int,
  date_key date,
  step_key int,
  km_key int,
  co_concentration_vol_per double precision,
  co2_concentration_vol_perc double precision,
  nox_concentration_ppm double precision,
  h2O_conc_vol_perc double precision,
  co_mass_g_s double precision,
  co2_mass_g_s double precision,
  nox_mass_g_s double precision
);


--Acquisition Fact ECU Table--
CREATE TABLE if not exists acquisition_fact_ecu(
  row_id bigserial,
  test_id int,
  date_key date,
  step_key int,
  km_key int,
  engine_coolant_temperature_by_ecu_degC double precision,
  fuel_pressure_by_ecu_kPa double precision,
  engine_RPM_by_ecu_rpm double precision,
  vehicle_speed_by_ecu_km_h double precision,
  vehicle_speed_by_ecu_m_s double precision,
  intake_air_temperature_by_ecu_degC double precision,
  maf_air_flow_rate_by_ecu_grams_sec double precision,
  maf_air_flow_rate_by_ecu_m3_s double precision,
  fuel_rail_pressure_by_ecu_kPa double precision,
  fuel_rail_pressure_diesel_by_ecu_kPa double precision,
  commande_EGR_by_ecu_perc double precision,
```

```
  barometric_pressure_by_ecu_kPa double precision,
  ambient_air_temperature_by_ecu_degC double precision,
  fuel_rail_pressure_Time_by_ecu_kPa double precision,
  engine_oil_temperature_by_ecu_degC double precision,
  engine_fuel_rate_by_ecu_L_h double precision,
  actual_engine_by_ecu_perc double precision,
  engine_reference_torque_by_ecu_Nm double precision
);

--Index for fact General table--
CREATE INDEX if not exists acquisition_fact_general_date_index on
    acquisition_fact_general using hash(date_key);

CREATE INDEX if not exists acquisition_fact_general_step_index on
    acquisition_fact_general using hash(step_key);

CREATE INDEX if not exists acquisition_fact_general_km_index on
    acquisition_fact_general using hash(km_key);


--Index for fact Flow table--
CREATE INDEX if not exists acquisition_fact_flow_date_index on
    acquisition_fact_flow using hash(date_key);

CREATE INDEX if not exists acquisition_fact_flow_step_index on
    acquisition_fact_flow using hash(step_key);

CREATE INDEX if not exists acquisition_fact_flow_km_index on
    acquisition_fact_flow using hash(km_key);


--Index for fact Particles table--
CREATE INDEX if not exists acquisition_fact_particles_date_index on
    acquisition_fact_particles using hash(date_key);

CREATE INDEX if not exists acquisition_fact_particles_step_index on
    acquisition_fact_particles using hash(step_key);

CREATE INDEX if not exists acquisition_fact_particles_km_index on
    acquisition_fact_particles using hash(km_key);
```

```
--Index for fact Ecu table--
CREATE INDEX if not exists acquisition_fact_ecu_date_index on
    acquisition_fact_ecu using hash(date_key);

CREATE INDEX if not exists acquisition_fact_ecu_step_index on
    acquisition_fact_ecu using hash(step_key);

CREATE INDEX if not exists acquisition_fact_ecu_km_index on
    acquisition_fact_ecu using hash(km_key);
```