Réorganiser la base de données, en « nyc_tab_organised » ; que j'ai partitionné en plusieurs petites tables selon les mois. Ensuite insertion des données dans «nyc_tab_organised ». C'est elle qui me sert de table de départ pour le modèle en flocon.

```
CREATE TABLE nyc_tab_organised (
   vendorid INTEGER,
   tpep_pickup_datetime TIMESTAMP WITHOUT TIME ZONE,
   tpep_dropoff_datetime TIMESTAMP WITHOUT TIME ZONE,
   passenger_count INTEGER,
   trip_distance DOUBLE PRECISION,
   ratecodeid INTEGER,
   store_and_fwd_flag TEXT,
   pulocationid INTEGER,
   dolocationid INTEGER,
   payment_type INTEGER,
   fare_amount DOUBLE PRECISION,
   extra DOUBLE PRECISION,
   mta_tax DOUBLE PRECISION,
   tip_amount DOUBLE PRECISION,
   tolls_amount DOUBLE PRECISION,
   improvement_surcharge DOUBLE PRECISION,
   total_amount DOUBLE PRECISION,
   congestion_surcharge DOUBLE PRECISION,
   airport_fee DOUBLE PRECISION
) PARTITION BY RANGE (tpep_pickup_datetime);
```

Les tables de partition

```
CREATE TABLE nyc_tab_jan2024 PARTITION OF nyc_tab_ organised
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
CREATE TABLE nyc_tab_feb2024 PARTITION OF nyc_tab_ organised
FOR VALUES FROM ('2024-02-01') TO ('2024-10-01');
```

Jusqu'au dernier mois.

);

Une table pour tous les mois avant janvier 2024

```
CREATE TABLE nyc_tab_before2023 PARTITION OF nyc_tab_ organised

FOR VALUES FROM ('2000-01-01') TO ('2023-12-01');

CREATE TABLE nyc_tab_default PARTITION OF nyc_tab_ organised DEFAULT;


SELECT inhrelid::regclass AS partition

FROM pg_inherits

WHERE inhparent = 'nyc_tab_ organised '::regclass;
```

Ensuite, déplacez les données dans les tables appropriées :

```
INSERT INTO nyc_tab_ organised

SELECT * FROM nyc_tab;
```

Le modèle en flocon

Table de faits : C'est la table principale qui contient les mesures (ici nyc_tab_organised)

Tables de dimensions : Ce sont les tables qui contiennent des informations supplémentaires qui décrivent les faits (infos sur les lieux, les prix, les paiements, etc)

Tables de dimension

```
CREATE TABLE vendor (

   vendorid INTEGER PRIMARY KEY,

);

CREATE TABLE location (

   locationid INTEGER PRIMARY KEY,

   location_name TEXT,

   city TEXT,

   state TEXT,

   country TEXT

);
```

```
CREATE TABLE payment_type (
   payment_type_id INTEGER PRIMARY KEY,
   payment_description TEXT
);

sql
CREATE TABLE date_dim (
   date_key DATE PRIMARY KEY,
   year INTEGER,
   month INTEGER,
   day INTEGER,
   quarter INTEGER
);
```

Relier les tables de dimension et la table de fait

Mettre à jour la table de fait pour utiliser les clés étrangères et les relier aux tables de dimensions

```
ALTER TABLE nyc_tab_organised
   ADD COLUMN vendor_id INTEGER REFERENCES vendor(vendorid),
   ADD COLUMN location_id INTEGER REFERENCES location(locationid),
   ADD COLUMN payment_type_id INTEGER REFERENCES payment_type(payment_type_id),
   ADD COLUMN date_key DATE REFERENCES date_dim(date_key);
```

Insertion les données

```
INSERT INTO vendor (vendorid)
SELECT DISTINCT vendorid
FROM nyc_tab_ organised;

INSERT INTO location (locationid)
SELECT DISTINCT locationid
FROM nyc_tab_ organised;
SELECT * FROM nyc_tab_ organised WHERE locationid IS NULL;
```

```
INSERT INTO location (locationid)

SELECT DISTINCT pulocationid FROM nyc_tab_ organised WHERE pulocationid IS NOT NULL;


INSERT INTO payment_type (payment_type_id)

SELECT DISTINCT payment_type_id

FROM nyc_tab_ organised;


INSERT INTO payment_type (payment_type_id, payment_description)

VALUES (0, 'Unknown');


SELECT * FROM nyc_tab_ organised WHERE payment_type_id IS NULL;


UPDATE nyc_tab_ organised

SET payment_type_id = 0

WHERE payment_type_id IS NULL;


INSERT INTO payment_type (payment_type_id)

SELECT DISTINCT payment_type_id FROM nyc_tab_ organised WHERE payment_type_id IS NOT
NULL;


INSERT INTO date_dim (date_key)

SELECT DISTINCT date_key FROM nyc_tab_ organised WHERE date_key IS NOT NULL;


INSERT INTO date_dim (date_key)

SELECT DISTINCT date_key

FROM nyc_tab_ organised;


INSERT INTO location (locationid)

SELECT DISTINCT locationid

FROM nyc_tab_ organised;
```

Refaire une mise à jour de la table de fait et des clés étrangères.

```
UPDATE nyc_tab_ organised

SET vendor_id = (SELECT vendorid FROM vendor WHERE vendorid = nyc_tab_organised.vendorid),

   location_id = (SELECT locationid FROM location WHERE locationid =
nyc_tab_organised.pulocationid),

   payment_type_id = (SELECT payment_type_id FROM payment_type WHERE payment_type_id =
nyc_tab_organised.payment_type),

   date_key = (SELECT date_key FROM date_dim WHERE date_key =
nyc_tab_organised.tpep_pickup_datetime::date);
```

Pour vérifier si tout est correct, interrogation des données du nouveau modèle.

```
SELECT

   f.vendor_id,

   f.location_id,

   f.payment_type_id,

   f.date_key,

   f.total_amount

FROM nyc_tab_organised f

JOIN vendor v ON f.vendor_id = v.vendorid

JOIN location l ON f.location_id = l.locationid

JOIN payment_type p ON f.payment_type_id = p.payment_type_id

JOIN date_dim d ON f.date_key = d.date_key

LIMIT 20;
```

Optimisation et indexation, pour rendre les requêtes plus rapides et plus efficaces.

```
CREATE INDEX idx_vendor_id ON nyc_tab_organised (vendor_id);

CREATE INDEX idx_location_id ON nyc_tab_organised (location_id);

CREATE INDEX idx_payment_type_id ON nyc_tab_organised (payment_type_id);

CREATE INDEX idx_date_key ON nyc_tab_organised (date_key);
```