

- **L3 MIAHS/Ingémath**
- **Université Paris Cité**
- Année 2023-2024
- [Course Homepage](#)
- [Moodle](#)



⚠ Les requêtes portent sur le schéma `nycflights` légèrement nettoyé.

This is a layout. You can create multiple layouts with the same or different tables.  
Double-click the table headers to edit.

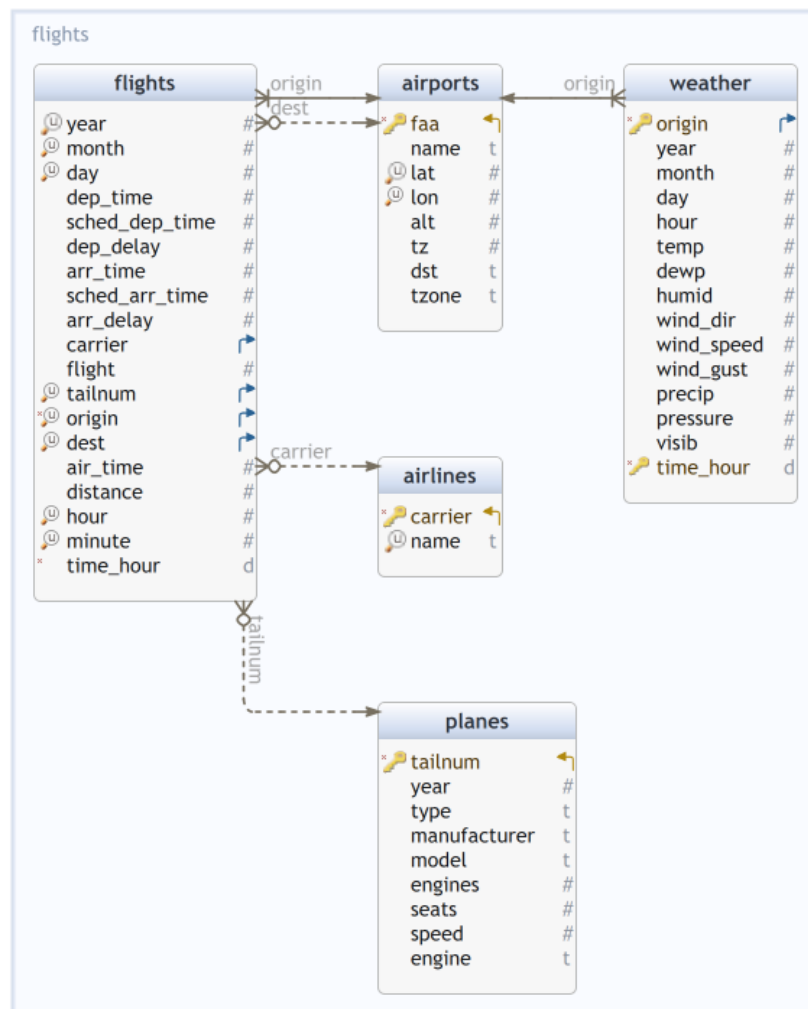


FIG. 1 : NYCFlights en relationnel à pattes de corbeau

## Définition du schéma en SQL

```
CREATE TABLE airlines (  
  carrier text NOT NULL,  
  "name" text NULL,  
  CONSTRAINT airlines_pk  
    PRIMARY KEY (carrier),  
  CONSTRAINT airlines_un  
    UNIQUE (name)  
);
```

```
CREATE TABLE airports (  
  faa text NOT NULL,  
  "name" text NULL,  
  lat float8 NULL,  
  lon float8 NULL,  
  alt float8 NULL,  
  tz float8 NULL,  
  dst text NULL,  
  tzone text NULL,  
  CONSTRAINT airports_pk  
    PRIMARY KEY (faa),  
  CONSTRAINT airports_un  
    UNIQUE (name),  
  CONSTRAINT airports_un_ll  
    UNIQUE (lat, lon)  
);
```

```
CREATE TABLE weather (  
  origin text NOT NULL,  
  "year" int4 NULL,  
  "month" int4 NULL,  
  "day" int4 NULL,  
  "hour" int4 NULL,  
  "temp" float8 NULL,  
  dewp float8 NULL,  
  humid float8 NULL,  
  wind_dir float8 NULL,  
  wind_speed float8 NULL,  
  wind_gust float8 NULL,  
  precip float8 NULL,  
  pressure float8 NULL,  
  visib float8 NULL,  
  time_hour timestamptz NOT NULL,  
  CONSTRAINT weather_pk  
    PRIMARY KEY (origin, time_hour)  
);
```

```
ALTER TABLE weather ADD  
  CONSTRAINT weather_fk  
  FOREIGN KEY (origin)  
  REFERENCES airports(faa)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE;
```

```
CREATE TABLE planes (  
  tailnum text NOT NULL,  
  "year" int4 NULL,  
  "type" text NULL,  
  manufacturer text NULL,  
  model text NULL,  
  engines int4 NULL,  
  seats int4 NULL,  
  speed int4 NULL,  
  engine text NULL,  
  CONSTRAINT planes_pk PRIMARY KEY (tailnum)  
);
```

Dans le schéma nycflights, on a aussi les dépendances fonctionnelles suivantes :

Table airports

- faa, name, et (lon, lat) sont des clés.

Table airlines

- carrier et name sont des clés

Table weather

- origin, time\_hour est une clé
- time\_hour → year, month, day, hour
- year, month, day, hour → time\_hour

Table planes

- tailnum est une clé
- model → manufacturer, engines, engine, type

Table flights

- tailnum, time\_hour → carrier

```
CREATE TABLE flights (  
    "year" int4 NULL,  
    "month" int4 NULL,  
    "day" int4 NULL,  
    dep_time int4 NULL,  
    sched_dep_time int4 NULL,  
    dep_delay float8 NULL,  
    arr_time int4 NULL,  
    sched_arr_time int4 NULL,  
    arr_delay float8 NULL,  
    carrier text NULL,  
    flight int4 NULL,  
    tailnum text NOT NULL,  
    origin text NOT NULL,  
    dest text NULL,  
    air_time float8 NULL,  
    distance float8 NULL,  
    "hour" float8 NULL,  
    "minute" float8 NULL,  
    time_hour timestamptz NOT NULL,  
    CONSTRAINT flights_pk  
        PRIMARY KEY (  
            tailnum, origin, time_hour  
        )  
);  
  
ALTER TABLE flights ADD  
    CONSTRAINT flights_fk  
    FOREIGN KEY (carrier)  
    REFERENCES airlines(carrier)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE;  
  
ALTER TABLE flights ADD  
    CONSTRAINT flights_fk_dest  
    FOREIGN KEY (dest)  
    REFERENCES airports(faa)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE;  
  
ALTER TABLE flights ADD  
    CONSTRAINT flights_fk_origin  
    FOREIGN KEY (origin)  
    REFERENCES airports(faa)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE;  
  
ALTER TABLE flights ADD  
    CONSTRAINT flights_fk_planes  
    FOREIGN KEY (tailnum)  
    REFERENCES planes(tailnum)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE;
```

- time\_hour → sched\_dep\_time
- sched\_dep\_time, dep\_time → dep\_delay
- sched\_arr\_time, arr\_time → arr\_delay
- origin, dest, dep\_time, arr\_time → airtime
- time\_hour → year, month, day, hour, minute
- year, month, day, hour, minute → time\_hour
- origin, dest → distance
- (tailnum, origin, time\_hour) est une clé
- (flight, dest, origin, year, month, day) est une clé

💡 Solution

💡 Solution

💡 Solution

Exercice : Requêtes (schéma nycflights)

### i Requête 1

Pour chaque couple origine/destination, lister les caractéristiques de l'avion le plus rapide sur la liaison.

### Solution

```
WITH R As (  
  SELECT f.origin, f.dest, f.tailnum, RANK() OVER w AS rnk  
  FROM flights AS f  
  WHERE f.airtime IS NOT NULL  
  WINDOW w AS (PARTITION BY f.origin, f.dest ORDER by f.airtime DESC)  
)  
  
SELECT R.origin, R.dest, p.*  
FROM (SELECT * FROM R WHERE R.rnk=1) AS S  
JOIN planes as p  
ON (R.tailnum=p.tailnum) ;
```

```
WITH R As (  
  SELECT f.origin, f.dest, min(f.airtime) as min_time  
  FROM flights AS f  
  WHERE f.airtime IS NOT NULL  
  GROUP BY f.origin, f.dest  
) , S AS (  
  SELECT f.origin, f.dest, f.tailnum  
  FROM flights AS f NATURAL JOIN R  
  WHERE f.airtime = R.min_time  
)  
  
SELECT S.origin, S.dest, p.*  
FROM S  
JOIN planes as p  
ON (S.tailnum=p.tailnum) ;
```

### Requête 2

Pour chaque aéroport d'origine, déterminer pour chaque heure de la journée, les températures maximales et minimales

### Solution

```
WITH o AS (  
  SELECT DISTINCT f.origin  
  FROM flights as f  
)  
  
SELECT w.origin, w.hour, MAX(w.temp), MIN(w.temp)  
FROM o NATURAL JOIN weather w  
GROUP BY w.hour ;
```

### Requête 3

Pour chaque aéroport d'origine, pour chaque température enregistrée en début d'heure, arrondie à l'entier le plus proche, indiquer la proportion de vols avec un retard supérieur à 30 mn au décollage.

### Solution

```
WITH R AS (  
  SELECT f.origin, f.year, f.month, f.day, f.hour, f.flight, f.dep_delay, ROUND(w.temp, 0) as temp  
  FROM flights f JOIN weater w ON  
    (f.origin=w.origin AND  
     f.year=w.year AND  
     f.month=w.month AND  
     f.day=w.day AND  
     f.hour=w.hour)  
)  
  
SELECT f.origin, f.t, SUM(f.dep_delay > 30)/COUNT(*) AS p  
FROM R AS f  
GROUP BY f.origin, f.t ;
```

### Requête 4

Pour chaque aéroport de destination, lister les modèles d'avion qui ont atterri au moins une fois dans cet aéroport.

### Solution

```
SELECT DISTINCT(f.dest, p.models)  
FROM flights f JOIN planes p ON (f.tailnum=p.tailnum) ;
```

### Requête 5

Pour chaque modèle d'avion, lister pour chaque semaine, le nombre de vols effectivement réalisés.

### Solution

```
SELECT p.model, DATE_PART('week', f.time_hour) AS semaine, COUNT(*) AS n  
FROM flights f JOIN planes p ON (f.tailnum=p.tailnum)  
WHERE f.dep_time IS NOT NULL  
GROUP BY p.model, DATE_PART('week', f.time_hour) AS semaine ;
```

### Requête 6

Quelles sont les destinations qui ne sont pas desservies le jeudi ?

### Solution

```
WITH R AS (  
    SELECT DISTINCT f.dest  
    FROM flights f  
    WHERE DATE_PART('week', f.time_hour) = 'Thursday'  
), S AS (  
    SELECT DISTINCT f.dest  
    FROM flights f  
)  
  
SELECT * FROM S  
  
EXCEPT  
  
SELECT * FROM R;
```

### Requête 7

Quelles sont les villes desservies par une seule compagnie le dimanche ?

### Solution

```
SELECT f.dest  
FROM flights f  
WHERE DATE_PART('week', f.time_hour) = 'Sunday'  
GROUP BY f.dest  
HAVING COUNT(DISTINCT f.carrier) = 1 ;
```

👉 L'utilisation d'une clause **WITH** (Common Table Expression) plutôt que d'une requête imbriquée rend le code plus lisible.

👉 Dans la réponse, nous donnons plus que ce qui était demandé. On aurait pu se contenter de ;

...

### Requête 8

Quelles sont les compagnies pour lesquelles le retard médian au décollage est supérieur à 15 minutes ?

### Solution

```
SELECT f.carrier  
FROM flights f  
WHERE f.dep_delay IS NOT NULL  
GROUP BY f.carrier  
HAVING MEDIAN(f.dep_delay) > 15 ;
```

### Requête 9

Quelles sont les destinations qui sont desservies quotidiennement par une compagnie ?

### Solution

...

### Requête 10

Quelles sont les compagnies qui exploitent des avions de tous les constructeurs ?

### Solution

### Quelques conseils

- Préférez les clauses `WITH` et les jointures aux requêtes imbriquées sauf si la requête imbriquée est très simple. C'est une question de lisibilité et donc souvent de correction.
- Ne mélangez pas les fonctions fenêtres et les clauses `GROUP BY ...`.

```
SELECT ..., FOO() OVER w
FROM R
WINDOW w AS (PARTITION BY ... ORDER BY ...)
GROUP BY ... ;
```

est tout simplement incorrect.

- Lorsque vous effectuez un partitionnement par `GROUP BY ...`, la clause `SELECT ...` est sévèrement contrainte, vous n'y trouverez que
  - les colonnes qui ont servi dans la clause `GROUP BY ...`, normalement elles devraient toutes y figurer
  - des fonctions d'aggrégation, comme `COUNT(...)`, `SUM(...)`, `VAR(...)`