- **L3 MIASHS/Ingémath**
- **Université Paris Cité**
- Année 2023-2024
- Course Homepage

- Moodle

⚠ Les trois exercices (modélisation, normalisation, requêtes) portent sur le schéma `nycflights` légèrement nettoyé.

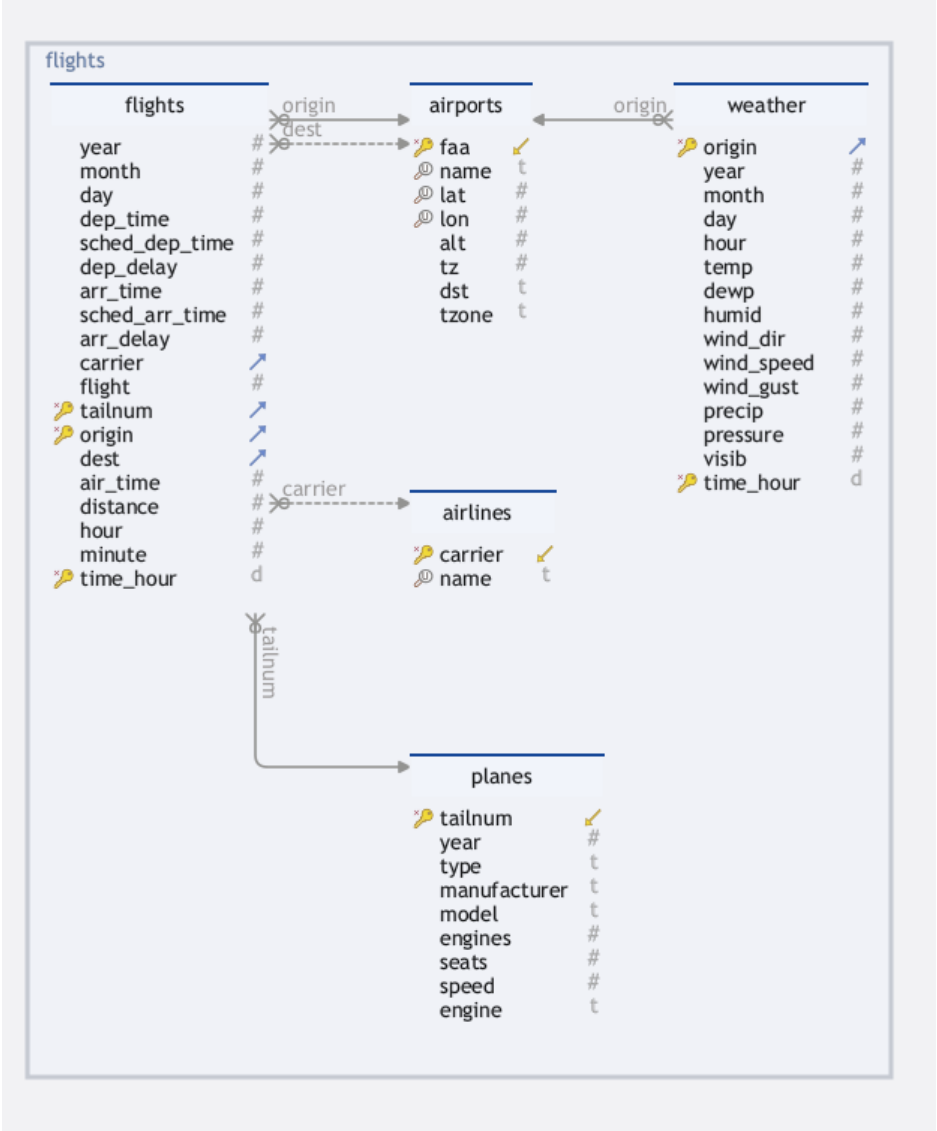


Fig. 1 : NYCFlights en relationel à pattes de corbeau

Définition du schéma en SQL

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

```sql
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)
);
```

```sql
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)
);
```

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

```sql
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`

```sql
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)
);
```

```sql
ALTER TABLE flights ADD
    CONSTRAINT flights_fk
    FOREIGN KEY (carrier)
    REFERENCES airlines(carrier)
    ON DELETE SET NULL
    ON UPDATE CASCADE;
```

```sql
ALTER TABLE flights ADD
    CONSTRAINT flights_fk_dest
    FOREIGN KEY (dest)
    REFERENCES airports(faa)
    ON DELETE SET NULL
    ON UPDATE CASCADE;
```

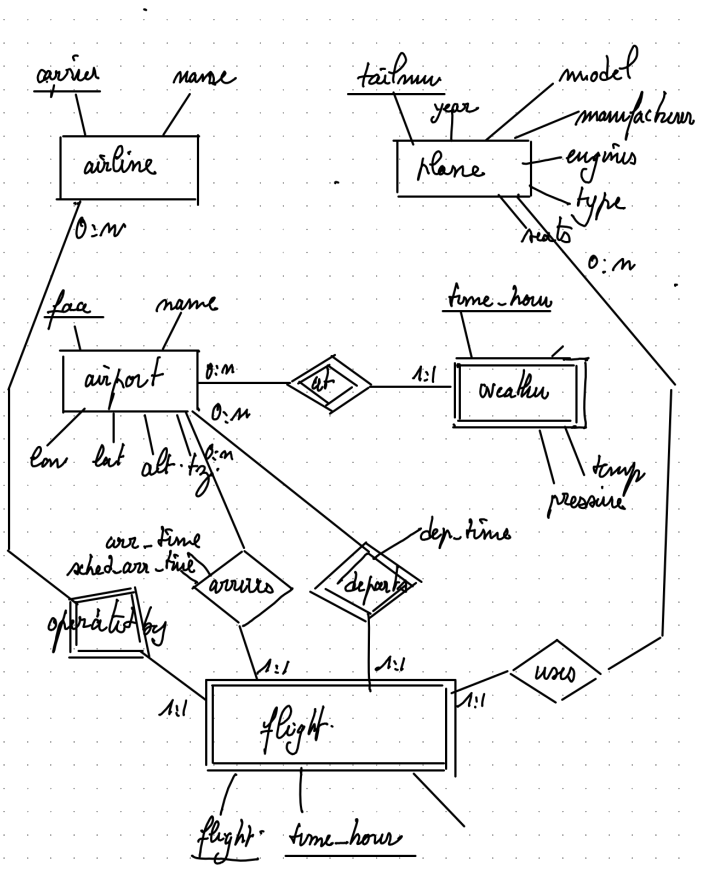
```sql
ALTER TABLE flights ADD
    CONSTRAINT flights_fk_origin
    FOREIGN KEY (origin)
    REFERENCES airports(faa)
    ON DELETE SET NULL
    ON UPDATE CASCADE;
```

```sql
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

Exercice : Modélisation

Le schéma entité-association est une tentative de rétro-ingénieurie du schéma relationnel `nycflights`.



Question

Proposez une variation de la représentation de l'entité `plane` où on définit une entité `model` (dont les instances seraient par exemple `Airbus A350`, `Boeing 777`, ...), et où chaque avion/aéronef serait considéré comme une réalisation d'une instance de `model`).
Préciser la nature de l'association entre `plane` et `model` et les cardinalités.
Précisez la répartition des attributs entre `plane` et `model`.

Solution

Association `est-un` (`is-a`), avec cardinalité `1:1` côté `plane` et `0:n` côté `model`
`model` a les attributs `model` (à renommer), `engines`, `type`, `manufacturer`, `engine`.
`plane` garde les attributs `year`, `seats`, `tailnum` (identifiant)
Si les instances de `plane` sont identifiées par `tailnum`, l'association `est-un` n'est pas identifiante.

Exercice : Normalisation

Question 1

Pour chaque table, préciser si elle est en FNBC ou non.

Solution

- ☑ `airlines` : en FNBC car deux colonnes
- ☑ `airports` : en FNBC
- ✗ `planes` : viole la condition FNBC par toutes les DF `model → ...`
- ✗ `weather` : deux clés `(origin, time_hour)` et `(origin, year, month, day, hour)`, mais les membres gauches des DF `time_hour → year, month, day, hour` et `year, month, day, hour → time_hour` violent la condition FNBC.
- ✗ `flights` viole la condition FNBC de plusieurs façons notamment via la DF `origin, dest → distance`.

Question 2

Si certaines tables ne sont pas en FNBC, proposer une décomposition en FNBC sans perte d'information.

Solution

- Pour `planes`, la décomposition `(model, manufacturer, engines, engine, type)`, `(tailnum, year, model, seats, speed)` est en FNBC.
- Pour `weather`, supprimer les colonnes `year, month, day, hour` qui peuvent être calculées à partir de `time_hour`.
- Pour `flights`, décomposition
  - `(origin, dest, distance)`
  - `(flight, time_hour)`
  - `(flight, dest, origin, time_hour, dep_time, sched_arr_time, arr_time, tailnum)`
  - `(tailnum, time_hour, carrier)`

Exercice : Requêtes (schéma `nycflights`)

Requête 1

For each airport of departure (denoted by `origin`), for each day of the year, list the codes (`carrier`) of the airlines that have one or more planes taking off from that airport on that day.

Solution

```
SELECT DISTINCT f.origin, f.year, f.month, f.day, f.carrier
FROM nycflights.flights f
ORDER BY f.origin, f.year, f.month, f.day;
```

👉 Il n'est pas nécessaire, ni même utile de procéder à une aggrégation (`GROUP BY`), il suffit de projeter sur les attributs qui identifient le jour de l'année `f.year, f.month, f.day`, l'aéroport de décollage (`origin`), et l'identifiant des compagnies aériennes `carrier`, et, bien sûr d'éliminer les doublons avec `DISTINCT`.
👉 Si on veut récupérer les noms complets des compagnies aériennes, plutôt que les codes, on peut effectuer une jointure naturelle avec `airlines`.
👉 Si on veut (finalement) une ligne par aéroport de départ et jour de l'année, on doit alors effectuer une aggrégation et utiliser une fonction d'aggrégation de chaine de caractères comme `concat()`

```
WITH R AS (
  SELECT DISTINCT f.origin, f.year, f.month, f.day, f.carrier
  FROM nycflights.flights f
)

SELECT R.origin, R.year, R.month, R.day, concat(R.carrier) AS carriers
FROM R
GROUP BY R.origin, R.year, R.month, R.day
ORDER BY R.origin, R.year, R.month, R.day ;
```

Requête 2

Lister pour chaque aéroport d'origine, chaque jour de l'année, pour chaque compagnie aérienne, le nombre d'avions exploités par la compagnie aérienne qui décollent de cet aéroport, ce jour là.

Solution

```
SELECT f.origin, f.year, f.month, f.day, f.carrier, COUNT(DISTINCT tailnum)
FROM nycflights.flights f
GROUP BY f.origin, f.year, f.month, f.day, f.carrier
ORDER BY f.origin, f.year, f.month, f.day, f.carrier;
```

👉 Il faut bien garder en tête que la clause `GROUP BY` (la clause de partitionnement) est définie par une liste de colonnes (ou plus généralement d'expressions) séparées par des virgules, ici `f.origin, f.year, f.month, f.day, f.carrier`. Ces colonnes sont évoquées dans *pour chaque aéroport d'origine, chaque jour de l'année, pour chaque compagnie aérienne*. Ces colonnes doivent aussi apparaître dans la clause `SELECT` (la clause de projection finale).
👉 Dans la clause de projection `SELECT ...` ne peuvent figurer que - les expressions qui apparaissent dans la clause `GROUP BY ...` - les expressions d'aggrégation comme `COUNT(...)`.

Requête 3

Lister pour chaque vol exploité par la compagnie (`carrier` nommé dans `airlines`) *Delta Air Lines Inc.* : les conditions météorologiques (`weather`) à l'heure prévue du décollage (`sched_dep_time`).

Solution

```
WITH delta AS (
    SELECT al.carrier
    FROM nycflights.airlines al
    WHERE al."name" = 'Delta Air Lines Inc.'
),
delta_f AS (
    SELECT f.origin, f.flight, f.year, f.month, f.day, f.hour
    FROM nycflights.flights f
    WHERE f.carrier IN (SELECT * FROM delta)
)

SELECT f.flight, w.*
FROM  nycflights.weather w NATURAL JOIN delta_f f;
```

Requête 4

Nombre de vols au décollage par aéoroport d'origine et par compagnie aérienne (`carrier`).

Solution

```
SELECT f.origin, f.carrier, COUNT(*) AS n
FROM nycflights.flights f
GROUP BY f.origin, f.carrier
ORDER BY f.carrier, n DESC;
```

Requête 5

Lister les caractéristiques des avions (`planes`) exploités par au moins deux compagnies aériennes (`carrier`) différentes dans la base de données.

Solution

```
WITH for_hire AS (
    SELECT f.tailnum, COUNT(DISTINCT f.carrier) AS n_carrier
    FROM nycflights.flights f
    GROUP BY f.tailnum
    HAVING COUNT(DISTINCT f.carrier) >=2
)

SELECT p.*
FROM nycflights.planes p NATURAL JOIN for_hire ;
```

Requête 6

Lister pour chaque jour et chaque aéoroport d'origine les dix avions les plus en retard au décollage (`dep_delay`). Ne pas prendre en compte les vols annulés (`dep_time IS NULL`).

Solution

```sql
WITH f_delayed AS (
  SELECT f.*, RANK() OVER w AS rnk
  FROM nycflights.flights f
  WHERE f.dep_time IS NOT NULL
  WINDOW w AS (PARTITION BY f.origin, f.year, f.month, f.day ORDER BY f.dep_delay DESC)
)

SELECT fd.origin, fd.year, fd.month, fd.day, fd.tailnum
FROM f_delayed fd
WHERE fd.rnk <= 10;
```

Requête 7

Lister pour chaque modèle d'avion (`model`) le nombre de jours où un avion de ce modèle a subi le plus grand retard au décollage (`dep_delay`) parmi les avions qui ont décollé ce jour là du même aéroport (`origin`).

Solution

```sql
WITH delayed_flight AS (
  SELECT f.origin, f.year, f.month, f.day, f.tailnum,
         RANK() OVER w AS rnk
  FROM nycflights.flights f
  WINDOW w AS (PARTITION BY f.origin, f.year, f.month, f.day
               ORDER BY f.dep_delay DESC)
), plane_of_day  AS (
  SELECT df.origin, df.year, df.month, df.day, df.tailnum
  FROM delayed_flight df
  WHERE df.rnk = 1
)

SELECT p.model, COUNT(DISTINCT(df.year, df.month, df.day)) AS nb_bad_days
FROM plane_of_day df  JOIN nycflights.planes p ON
  (df.tailnum=p.tailnum)
GROUP BY p.model
ORDER BY nb_bad_days DESC;
```

👉 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 ;

```sql
...

SELECT p.model
FROM plane_of_day df  NATURAL JOIN nycflights.planes p
GROUP BY p.model ;
```

Requête 8

Lister les aéroports de destination (`dest`) qui sont desservis au moins une fois à partir de chaque aéroport de départ (`origin`).

Solution

```sql
WITH origins AS (
  SELECT DISTINCT f.origin  -- les origines
  FROM nycflights.flights f
), dests AS (
  SELECT DISTINCT f.dest  -- les destinations
  FROM nycflights.flights f
),
origin_dest AS (
  SELECT DISTINCT f.dest, f.origin
  FROM nycflights.flights f   -- les couples realises
  ),
origin_cross_dest AS (
  SELECT d.dest, o.origin  -- les couples possibles
  FROM dests d, origins o
),
whitness_orphans AS (
  SELECT *
  FROM origin_cross_dest

  EXCEPT

  SELECT *
  FROM origin_dest  -- couples pas realises
)

SELECT dest
FROM dests

EXCEPT

SELECT w.dest
FROM whitness_orphans w ;
```

👉 Dans cette réponse, nous n'avons pas utilisé l'agrégation (`GROUP BY ...`). On aurait pu écrire cette requête en algèbre relationnelle. Avec l'agrégation c'est plus simple.

```sql
WITH R AS (
  SELECT COUNT(DISTINCT f.origin) as nb_origins
  FROM nycflights.flights f
)

SELECT f.dest
FROM nycflights.flights f
GROUP BY f.dest
HAVING COUNT(DISTINCT f.origin) >= ALL (SELECT * FROM R) ;
```

Requête 9

Lister les compagnies aériennes (`carrier`) pour lesquelles, chaque jour, au moins un avion figure parmi les 10 avions les plus en retard au décollage (`dep_delay`) de son aéroport de départ (`origin`).

Solution

```
WITH delayed_flight AS (
  SELECT f.origin, f.year, f.month, f.day, f.tailnum, f.carrier,
         RANK() OVER w AS rnk
  FROM nycflights.flights f
  WINDOW w AS (PARTITION BY f.origin, f.year, f.month, f.day
               ORDER BY f.dep_delay DESC)
), carriers_of_day  AS (
  SELECT DISTINCT df.origin, df.year, df.month, df.day, df.carrier
  FROM delayed_flight df
  WHERE df.rnk <= 10
), nb_bad_days_per_carrier AS (
  SELECT df.origin,
         df.carrier,
         COUNT(DISTINCT (df.year, df.month, df.day)) as nb
  FROM carriers_of_day df  GROUP BY df.origin, df.carrier
), nb_days AS (
  SELECT COUNT(DISTINCT (df.year, df.month, df.day)) AS nb
  FROM carriers_of_day df
)

SELECT a.origin, a.carrier
FROM nb_bad_days_per_carrier a
WHERE a.nb >= ALL (SELECT nb FROM nb_days) ;
```

☛ La complexité de la réponse tient à la complexité de la question : s'il fallait écrire une formule du calcul relationnel des tuples, il faudrait introduire des quantifications alternées. Les requêtes de la clause `WITH` correspondent à ses alternances de quantification $\forall, \exists$. `carriers_of_day` liste pour chaque journée répertoriée dans la base, les transporteurs dont un avion figure parmi les retardataires du jour pour un aéroport donné. On détermine ensuite le nombre total de journées de la honte pour chaque transporteur, et le nombre total de journées, on sélectionne enfin les transporteurs dont le nombre de journées de la honte coïncide avec le nombre total de journées.

Requête 10

Pour chaque couple `(origin, dest)`, lister les dix vols les plus rapides (`airtime` donne le temps de vol, `distance` la distance entre `dest` et `origin`).

Solution

```
WITH R AS (
  SELECT f.origin, f.dest, f.tailnum, f.flight, RANK() OVER w AS rnk
  FROM nycflights.flights f
  WHERE f.airtime IS NOT NULL
  WINDOW w AS (PARTITION BY f.origin, f.dest ORDER BY f.distance/f.airtime DESC)
)

SELECT R.*
FROM R
WHERE R.rnk <= 10 ;
```

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(...)`