

Projet Final - Administration de Bases de Données Relationnelles



NOM : ADRAOUI

PRÉNOM : ALI

Classe : Cybersécurité

Année : B3

Date de rendu : 26/11/2025

1. Introduction

1.1. Présentation du projet Ynov-Air

Le projet Ynov-Air consiste en l'administration et l'amélioration d'un système de réservation de vols aériens. L'objectif est de mettre en pratique les compétences acquises en administration de bases de données relationnelles.

1.2. Fonctionnalité choisie et justification

Fonctionnalité choisie : Nous avons ajouté une fonctionnalité qui permet l'annulation des réservations demandé tout en justifiant la raison de l'annulation.

Justification : Dans certains cas, des clients sont obligés d'annuler leur réservation à cause d'un problème quelconque. On peut aussi appliquer le CRUD depuis l'ORM de l'application.

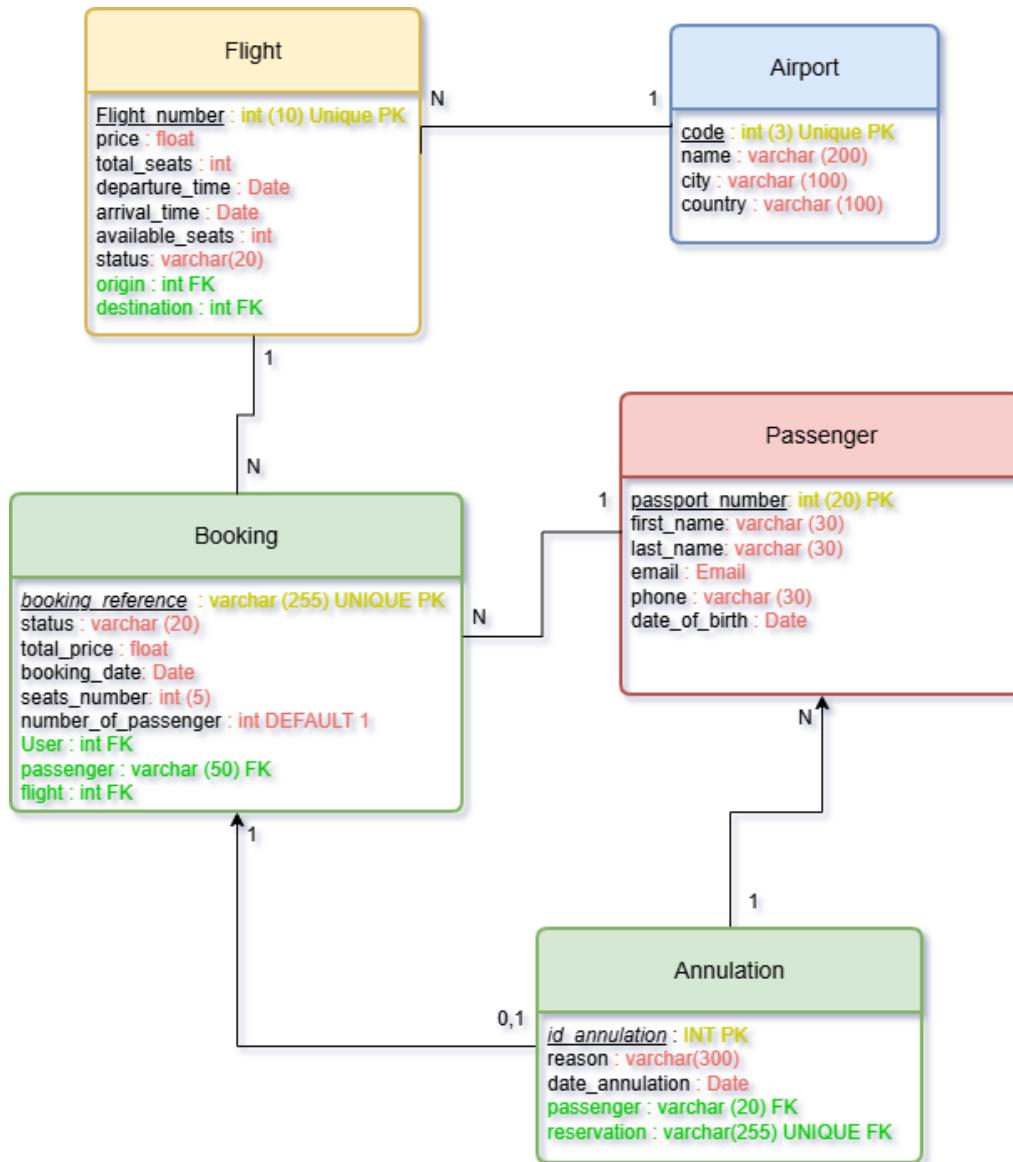
1.3. Objectifs personnels

Mon objectif principal dans ce projet est de développer une application web avec Django capable de gérer des données, afin de renforcer mes compétences en conception de modèles, en manipulation de bases de données et en mise en place d'un CRUD.

2. Modélisation de la Base de Données

2.1. Schéma relationnel mis à jour (MCD/MLD)

Voici le schéma relationnel qui définit chacune des entités ainsi que leurs relations



2.2. Explication des choix de conception

J'ai ajouté la table Annulation afin de permettre la gestion des réservations annulées par les clients

2.3. Dictionnaire de données

Table	Colonne	Type de Donnée	Contraintes
Annulation	Id_annulation	INT	PRIMARY KEY
	Date_annulation	DATE	-
	Reason	VARCHAR (20)	NOT NULL
	Passenger	VARCHAR (20)	FOREIGH KEY DEFAULT 1
	Booking	VARCHAR (255)	FOREIGH KEY

3. Scripts SQL de Création

3.1. Création des tables et des relation

```
class Annulation (models.Model):
    date_annulation = models.DateField(
        auto_now_add=True,
        null=True,
        verbose_name="Date d'annulation",
    )
    reason = models.CharField(
        max_length=400,
        blank=False,
        null=False,
        verbose_name="Raison"
    )
    booking = models.ForeignKey(
        Booking,
        default= None,
        on_delete=models.SET_NULL,
        null=True,
        verbose_name="Reservation",
    )

    passenger = models.ForeignKey(
        Passenger,
        on_delete=models.CASCADE,
        default=1,
        verbose_name="Passager")

    class Meta:
        verbose_name = "Annulation"
        verbose_name_plural = "Annulations"
```

5. Requêtes SQL Documentées

5.1. Catégorie 1 : Requêtes de Base

1) Liste des vols d'une compagnie spécifique pour une date donnée

Chaque vol opéré par une compagnie possède un flight_number qui sert à l'identifier de manière unique au sein de cette compagnie

Exemple :

	flight_number	id
1	YN100	1
2	YN101	2
3	YN102	3
4	YN103	4
5	YN104	5
6	YN105	6
7	YN106	7
8	YN107	8
9	YN108	9
10	YN109	10

A red arrow points from the 'flight_number' column to the word 'compagnie'.

Voilà le résultat :

```
-- Liste des vols d'une compagnie spécifique
-- pour une date donnée
▷Select (Shift + Enter)
Select F.flight_number, F.departure_time
from flights_flight AS F
Where F.flight_number LIKE '%compagnie%'
AND F.departure_time = 'temps_voulu'|
```

Exemple :

The screenshot shows two panes. The left pane displays a table of flight information from a database named 'db.sqlite3'. The right pane shows the SQL code used to generate the results.

flight_number	departure_time
YN104	2025-11-20 10:07:10...
YN105	2025-11-20 14:07:10...
YN106	2025-11-20 18:07:10...
YN107	2025-11-20 22:07:10...
YN132	2025-11-20 10:07:10...
YN133	2025-11-20 14:07:10...
YN134	2025-11-20 18:07:10...
YN135	2025-11-20 22:07:10...
YN160	2025-11-20 10:07:10...

```

1 -- database: c:\Users\redacted\Desktop\python\dj...
2 -- Liste des vols d'une compagnie spécifique
3 -- pour une date donnée
4 Select F.flight_number, F.departure_time
5 from flights_flight AS F
6 Where F.flight_number LIKE '%YN%'
7 AND F.departure_time LIKE '2025-11-20%'

```

2) Passagers ayant réservé un vol spécifique :

The screenshot shows two panes. The left pane displays a table of passenger booking information from a database named 'db.sqlite3'. The right pane shows the SQL code used to generate the results.

booking_reference	passenger_id	REFERENCES
2FI16DF0	flights_passenger(id)	4 first_name: 'Vido...

```

2 -- Liste des vols d'une compagnie spécifique
3 -- pour une date donnée
4 Select booking_reference , passenger_id
5 from flights_booking where flight_id = 5
6
7

```

3) Prix total des réservations par mois :

The screenshot shows two panes. The left pane displays a table of monthly revenue from a database named 'db.sqlite3'. The right pane shows the SQL code used to generate the results.

mois	revenu_total
2025-05-28 18:16:19	1066.49
2025-05-29 16:34:24	162.78
2025-05-30 22:04:16	208.77

```

1 -- database: c:\Users\Lamia\Desktop\pyth Untitled-1
2 -- Prix total des réservations par mois
3 Select b.booking_date AS mois,
4 SUM(b.total_price) AS revenu_total
5 FROM flights_booking AS b
6 GROUP BY mois
7 ORDER BY mois

```

4) Passagers ayant réservé un vol spécifique :

The screenshot shows two panes. The left pane displays a table of airport movement counts from a database named 'db.sqlite3'. The right pane shows the SQL code used to generate the results.

aeroport	mouvements
Aéroport Charles de...	392
Aéroport Lyon-Saint...	224
Aéroport de Barcelo...	112
Aéroport Toulouse-B...	112
Aéroport Marseille ...	112
Aéroport de Londres...	56
Aéroport Nice Côte ...	56
Aéroport Madrid-Bar...	56

```

3 SELECT a.name AS aeroport,
4 COUNT(f.id) AS mouvements
5 FROM flights_flight AS f
6 inner JOIN flights_airport AS a
7 ON f.origin_id = a.id
8 OR f.destination_id = a.id
9 GROUP BY a.name
10 ORDER BY mouvements DESC
11

```

5) Compagnies avec le plus de vols opérés

```

SELECT f.flight_number ,count(f.id) as nb_vol FROM flights_flight f
GROUP BY f.flight_number
ORDER BY nb_vol DESC

```

The screenshot shows a SQLite database interface with two panes. The left pane is a Query Editor displaying the results of a SELECT query. The right pane is a code editor with the same query written in SQL.

5.2. Catégorie 2 : Requêtes avec Jointures Complexes

1) Revenus par compagnie et par route

```

SELECT flight.flight_number AS compagnie,
       origin.code AS origin_code,
       dest.code AS destination_code,
       SUM(booking.total_price) AS revenus
  FROM flights_booking AS booking
   JOIN flights_flight AS flight ON flight.id = booking.flight_id
   JOIN flights_airport AS origin ON origin.id = flight.origin_id
   JOIN flights_airport AS dest ON dest.id = flight.destination_id
 WHERE booking.status IN ('CONFIRMED', 'COMPLETED')
 GROUP BY flight.flight_number, origin.code, dest.code
 ORDER BY revenus DESC;

```

The screenshot shows a SQLite database interface with two panes. The left pane is a Query Editor displaying the results of a complex query involving multiple tables (flights_booking, flights_flight, flights_airport). The right pane is a code editor with the same query written in SQL.

2) Occupation moyenne des vols par destination

```

SELECT d.code AS destination,
       AVG((f.total_seats - f.available_seats) * 1.0 / f.total_seats) * 100 AS taux
  FROM flights_flight f
   JOIN flights_airport d ON d.id = f.destination_id
 WHERE f.total_seats > 0
 GROUP BY d.code;

```

The screenshot shows a SQLite database interface with two panes. The left pane is a Query Editor displaying the results of a query to calculate the average occupancy rate for each destination. The right pane is a code editor with the same query written in SQL.

5.3. Catégorie 3 : Requêtes Analytiques

1) Top 10 des routes les plus rentables

	origin_code	destination_code	revenus	
1	BCN	LYS	8091.7	
2	CDG	NCE	6588.62	
3	LHR	CDG	6439.14	
4	LYS	CDG	5920.01	
5	NCE	CDG	5382.13	
6	LYS	BCN	4812.87	
7	CDG	LHR	4609.13	
8	MAD	CDG	4577.389999999999	
9	LYS	MRS	4160.46	
10	CDG	LYS	4015.16	
+				

```

    ▷Select (Shift + Enter)
5   SELECT origin.code AS origin_code,
6   | dest.code AS destination_code,
7   | SUM(booking.total_price) AS revenus
8   FROM flights_booking AS booking
9   JOIN flights_flight AS flight ON flight.id = booking.flight_id
10  JOIN flights_airport AS origin ON origin.id = flight.origin_id
11  JOIN flights_airport AS dest ON dest.id = flight.destination_id
12  WHERE booking.status IN ('CONFIRMED', 'COMPLETED')
13  GROUP BY origin_code, destination_code
14  ORDER BY revenus DESC
15  LIMIT 10

```

2) Analyse de la saisonnalité des réservations

	date	nb_reservations	revenus	
1	2025-05-28 18:16:19	1	1066.49	
2	2025-05-31 21:10:30	1	521.95	
3	2025-06-03 09:53:37	1	1574.32	
4	2025-06-03 16:09:56	1	2447.68	
5	2025-06-12 23:28:20	1	813.53	
6	2025-06-13 02:49:53	1	412.1	
7	2025-06-13 16:45:52	1	217.08	
8	2025-06-15 04:45:53	1	1223.11	
9	2025-06-15 09:56:06	1	333.33	
10	2025-06-20 18:39:55	1	193.91	
11	2025-06-23 04:07:21	1	595.45	
12	2025-06-23 04:52:49	1	579.74	
13	2025-06-23 09:37:32	1	414.81	
14	2025-06-24 16:43:51	1	470.43	
15	2025-06-27 19:54:29	1	1673.07	
16	2025-06-27 23:41:10	1	1245.43	
17	2025-06-30 05:47:37	1	456.14	
18	2025-06-30 18:41:46	1	1023.96	

```

    ▷Select (Shift + Enter)
5   SELECT booking.booking_date AS date,
6   | COUNT(*) AS nb_reservations,
7   | SUM(booking.total_price) AS revenus
8   FROM flights_booking AS booking
9   WHERE booking.status IN ('CONFIRMED', 'COMPLETED')
10  GROUP BY date
11  ORDER BY date;

```

3) Taux de remplissage moyen par type d'avion

	type_avion	taux_remplissage_moyen	
1	200	0.36360139860139856	
2	180	0.3577690972222222	
3	250	0.3491891891891892	
4	150	0.34761229314420805	
+			

```

    ▷Select (Shift + Enter)
5   SELECT flight.total_seats AS type_avion,
6   | AVG( (flight.total_seats - flight.available_seats) * 1.0 / flight.total_seats)
7   | AS taux_remplissage_moyen
8   | FROM flights_flight AS flight
9   | GROUP BY type_avion
10  | ORDER BY taux_remplissage_moyen DESC

```

5.4. Catégorie 4 : Requêtes Liées à la nouvelle Fonctionnalité

1) Liste des annulations (triées par date)

SQLITE 3.50.4

```

2   SELECT id, date_annulation, reason
3   FROM flights_annulation
4   ORDER BY date_annulation DESC;

```

	id	date_annulation	reason
1	19	2025-11-24	Changement de plan ...
2	51	2025-11-24	j'ai plus envie de ...
3	32	2025-11-23	Maladie soudaine
4	11	2025-11-22	Double réservation
5	23	2025-11-22	Raison professionne...
6	29	2025-11-22	Problème de visa
7	41	2025-11-22	Double réservation
8	44	2025-11-22	Double réservation
9	47	2025-11-22	Maladie soudaine
10	3	2025-11-19	Urgence familiale
11	39	2025-11-19	Retard de connexion...
12	17	2025-11-17	Retard de connexion...
13	36	2025-11-16	Annulation du vol p...
14	14	2025-11-14	Erreur de date lors...
15	22	2025-11-14	Changement de plan ...
16	16	2025-11-13	Maladie soudaine
17	26	2025-11-13	Maladie soudaine
18	28	2025-11-13	Annulation du vol p...
19	46	2025-11-11	Problème de visa
20	13	2025-11-10	Problème de visa
21	42	2025-11-10	Annulation du vol p...
22	5	2025-11-07	Conditions météorologiques
23	4	2025-11-05	Conditions météorologiques
24	6	2025-11-05	Annulation du vol p...
25	37	2025-11-04	Maladie soudaine
26	9	2025-10-29	Urgence familiale

2) Motifs d'annulation les plus fréquents

▷Select (Shift + Enter)

```

5  SELECT annulation.reason,
6   COUNT(*) AS nb
7   FROM flights_annulation AS annulation
8   GROUP BY annulation.reason
9   ORDER BY nb DESC
10  LIMIT 10;

```

	reason	nb
1	Maladie soudaine	8
2	Urgence familiale	6
3	Problème de visa	6
4	Conditions météorologiques	6
5	Changement de plan ...	6
6	Double réservation	5
7	Annulation du vol p...	5
8	Retard de connexion...	4
9	Raison professionnelle	2
10	Erreur de date lors...	2

3) Revenus perdus par destination (sur réservations annulées)

▷Select (Shift + Enter)

```

5  SELECT d.code AS destination_code,
6   SUM(b.total_price) AS revenus_perdus,
7   COUNT(*) AS nb_annulations
8   FROM flights_annulation AS a
9   JOIN flights_booking AS b ON b.id = a.booking_id
10  JOIN flights_flight AS f ON f.id = b.flight_id
11  JOIN flights_airport AS d ON d.id = f.destination_id
12  WHERE b.status = 'CANCELLED'
13  GROUP BY destination_code
14  ORDER BY revenus_perdus DESC;

```

	destination_code	revenus_perdus	nb_annulations
1	LYS	4851.67	3
2	CDG	1623.139999999999	5
3	MRS	642.91	1
4	TLS	490.57	1

6. Documentation de la Nouvelle Fonctionnalité

6.1. Description détaillée

La table ***Annulation*** permet de conserver une trace précise de chaque réservation annulée. Elle indique quel passager a annulé, quelle réservation est concernée, à quelle date l'annulation a eu lieu, ainsi que la raison fournie par le passager.

6.2. Cas d'usage

Il peut être utilisé pour avoir un historique précis des annulations pour faire des statistiques, comprendre les motifs les plus fréquents et vérifier facilement ce qui s'est passé en cas de contrôle ou de demande d'information.

6.3. Règles de gestion

- Chaque annulation doit être associée à un passager
- Une annulation peut être liée à une réservation, mais ce n'est pas obligatoire
- Le motif d'annulation est obligatoire
- La date d'annulation est fixée automatiquement
- Créer une annulation entraîne un changement d'état de la réservation
- Le système ne libère pas automatiquement les sièges lors d'une annulation via la vue
- Une réservation ne doit normalement pas être annulée plusieurs fois
- Une annulation doit toujours contenir des informations fiables pour l'audit

6.4. Contraintes d'intégrité

- **reason** est obligatoire et doit contenir un texte non vide (max 400 caractères)
- **date_annulation** est automatiquement renseignée à la création et ne doit pas être modifiée ensuite
- **id** est une clé primaire auto-incrémentée garantissant l'unicité de chaque annulation

Conclusion

La mise en place de la table **Annulation** a permis d'ajouter au projet Ynov-Air une traçabilité complète des réservations annulées, en enregistrant pour chaque action qui a annulé, quelle réservation est concernée, à quelle date et pour quel motif. Cette fonctionnalité répond pleinement aux besoins de suivi, de reporting et d'audit, tout en assurant l'intégrité des données grâce à des règles cohérentes : motif obligatoire, date générée automatiquement, lien obligatoire avec le passager et conservation de l'historique même si la réservation est supprimée. Le développement a tout de même présenté quelques difficultés, notamment la gestion des relations entre Booking et Annulation, l'intégrité référentielle (SET NULL / CASCADE) et l'absence initiale de libération automatique des sièges, ce qui a nécessité une clarification de la logique métier. Parmi les améliorations possibles, il serait pertinent d'appeler systématiquement la méthode booking.cancel() pour centraliser le comportement d'annulation, d'empêcher plusieurs annulations sur une même réservation. cette table apporte un véritable plus au fonctionnement du système en garantissant une gestion fiable, propre et historique des annulations.