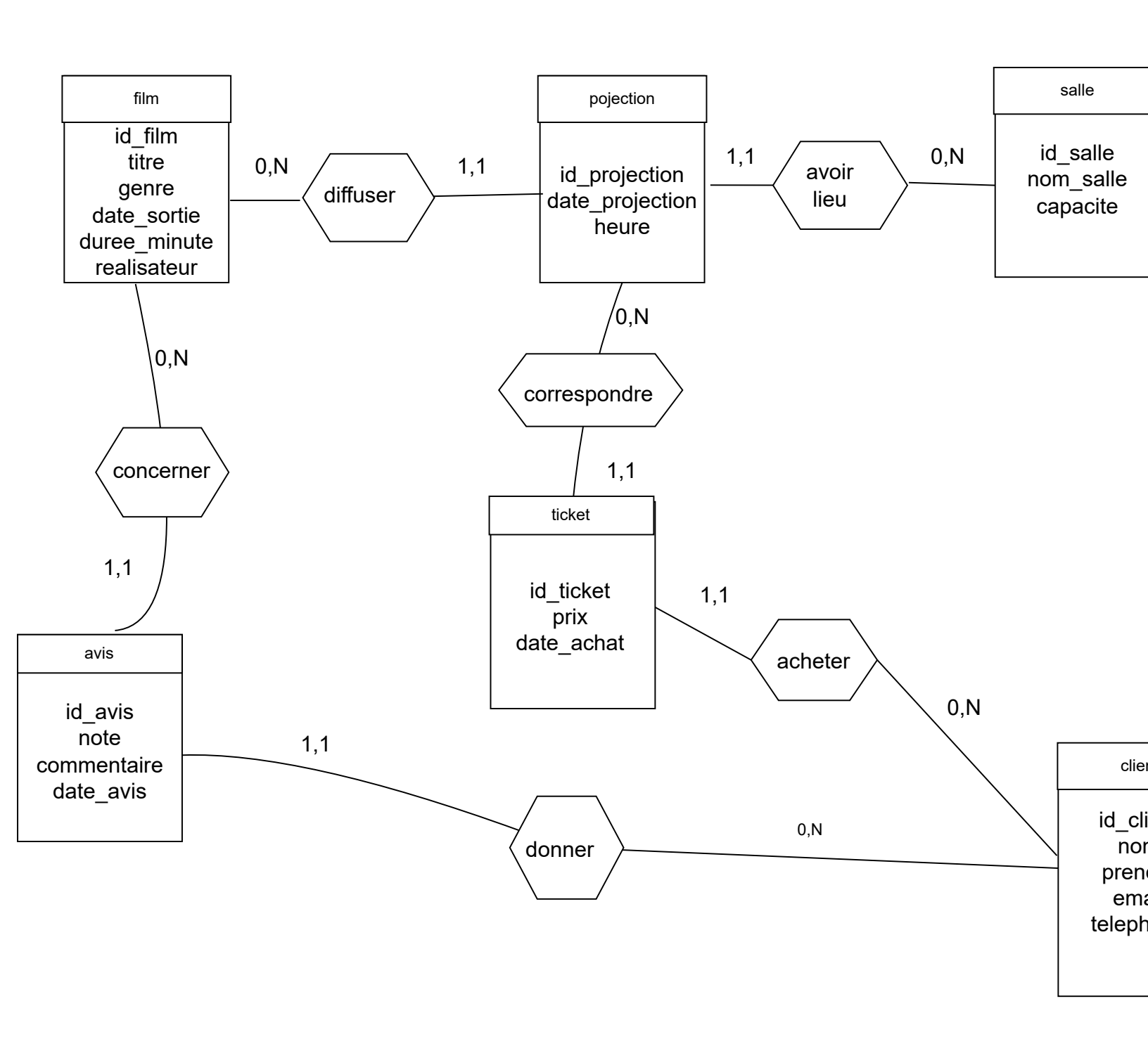
1) MODEL CONCEPTUEL DE TIDIANE_FLIX :



2. **Modélisation logique :**

🎬 Film

Attribut	Taper	Clé	Contraintes
id_film	INT	PK	AUTO_INCREMENT, NON NULL
titre	VARCHAR(255)		NON NUL
genre	VARCHAR(100)		
date_sortie	DATE		
durée_minutes	INT		
réalisateur	VARCHAR(255)		

🏛 Salle

Attribut	Taper	Clé	Contraintes
id_salle	INT	PK	AUTO_INCREMENT, NON NULL
nom_salle	VARCHAR(255)		NON NUL
capacité	INT		NON NUL

🎥 Projection

Attribut	Taper	Clé	Contraintes
id_projection	INT	PK	AUTO_INCREMENT, NON NULL
date_projection	DATE		NON NUL
heure	TEMPS		NON NUL
id_film	INT	FK	RÉFÉRENCES Film(id_film)
id_salle	INT	FK	RÉFÉRENCES Salle(id_salle)

👤 Client

Attribut	Taper	Clé	Contraintes
id_client	INT	PK	AUTO_INCREMENT, NON NULL
nom	VARCHAR(255)		NON NUL
prénom	VARCHAR(255)		
e-mail	VARCHAR(255)		UNIQUE
téléphone	VARCHAR(20)		

🎫 Billet

Attribut	Taper	Clé	Contraintes
id_ticket	INT	PK	AUTO_INCREMENT, NON NULL
prix	DÉCIMAL(10,2)		NON NUL
date_achat	DATE		NON NUL
id_client	INT	FK	RÉFÉRENCES Client(id_client)
id_projection	INT	FK	RÉFÉRENCES Projection(id_projection)

📝 Avis

Attribut	Taper	Clé	Contraintes
id_avis	INT	PK	AUTO_INCREMENT, NON NULL
note	INT		VÉRIFIER (note ENTRE 1 ET 5)
commentaire	TEXTE		
date_avis	DATE		
id_client	INT	FK	RÉFÉRENCES Client(id_client)
id_film	INT	FK	RÉFÉRENCES Film(id_film)

💡 Projet : TidianeFlix - Gestion de projection de films

---

1. 🎯 Création de la base de données

SQL

```
CREATE DATABASE TIDIANE_FLIX;
```

---

2. Création des tables

SQL

```
-- Table : film

CREATE TABLE film (
id_film SERIAL PRIMARY KEY,
titre VARCHAR(100),
genre VARCHAR(50),
date_sortie DATE,
duree INT,
realisateur VARCHAR(100)
);

-- Table : salle

CREATE TABLE salle (
id_salle SERIAL PRIMARY KEY,
nom_salle VARCHAR(50),
capacite INT
);

-- Table : projection

CREATE TABLE projection (
id_projection SERIAL PRIMARY KEY,
date_projection DATE,
heure_debut TIME,
heure_fin TIME,
id_film INT REFERENCES film(id_film),
id_salle INT REFERENCES salle(id_salle)
);

-- Table : client

CREATE TABLE client (
id_client SERIAL PRIMARY KEY,
nom VARCHAR(50),
prenom VARCHAR(50),
email VARCHAR(100),
telephone VARCHAR(20)
);

-- Table : ticket

CREATE TABLE ticket (
id_ticket SERIAL PRIMARY KEY,
prix DECIMAL(6,2),
date_achat DATE,
id_projection INT REFERENCES projection(id_projection),
id_client INT REFERENCES client(id_client)
);

-- Table : avis

CREATE TABLE avis (
id_avis SERIAL PRIMARY KEY,
note INT,
commentaire TEXT,
date_avis DATE,
id_film INT REFERENCES film(id_film),
id_client INT REFERENCES client(id_client)
);
```

---

3. 🧪 Exemples d'insertion de données fictives (à adaptateur)

SQL

```
-- Insertion de films

INSERT INTO film (titre, genre, date_sortie, duree, realisateur)
VALUES
('Wakanda Forever', 'Action', '2022-11-11', 161, 'Ryan Coogler'),
('Titanic', 'Romance', '1997-12-19', 195, 'James Cameron'),
('Inception', 'Science-fiction', '2010-07-16', 148, 'Christopher Nolan');
-- (Ajoute 12 autres films…)

-- Insertion de salles

INSERT INTO salle (nom_salle, capacite)
VALUES
('Salle A', 150), ('Salle B', 100), ('Salle C', 200);

-- Insertion de clients

INSERT INTO client (nom, prenom, email, telephone)

VALUES
('Niang', 'Cheikh', 'cheikh@gmail.com', '770000000'),
('Diallo', 'Aminata', 'ami@gmail.com', '780000001');
-- (Ajoute jusqu'à 20 clients…)

-- Insertion de projections

INSERT INTO projection (date_projection, heure_debut, heure_fin, id_film, id_salle)
VALUES
('2025-05-01', '18:00', '20:41', 1, 1),
('2025-05-02', '20:00', '22:30', 2, 2);
-- (Ajoute jusqu'à 20 projections…)

-- Insertion de tickets

INSERT INTO ticket (prix, date_achat, id_projection, id_client)
VALUES
(3000, '2025-05-01', 1, 1),
(3500, '2025-05-02', 2, 2);
-- (Ajoute jusqu'à 30 tickets…)

-- Insertion d'avis

INSERT INTO avis (note, commentaire, date_avis, id_film, id_client)
VALUES
(5, 'Excellent film !', '2025-05-01', 1, 1),
(4, 'Très bon mais un peu long', '2025-05-02', 2, 2);
-- (Ajoute jusqu'à 10 avis…)
```

---

4. 🔍 Requêtes SQL simples

a) Lister tous les films avec leur durée et leur genre :

SQL

```
SELECT titre, duree, genre
FROM film;
```

b) Afficher les projections d'un film donné (ex. : *Wakanda Forever*) :

Méthode sans jointure :

SQL

```
SELECT id_film
FROM film
WHERE titre = 'Wakanda Forever';

-- Ensuite, avec l'ID trouvé (ex. 1)

SELECT *
FROM projection
WHERE id_film = 1;
```

c) Lister les clients triés par nom :

SQL

```
SELECT *
FROM client
ORDER BY nom;
```

d) Compter le nombre total de films projetés dans la semaine :

SQL

```
SELECT COUNT(DISTINCT id_film) AS nombre_films
FROM projection
WHERE date_projection BETWEEN CURRENT_DATE AND CURRENT_DATE + INTERVAL '7 days';
```

🔗 Requêtes avec jointures

✅ 1. Détail des tickets achetés

(*film, date de projection, client, prix du ticket*)

sql

```
SELECT
t.id_ticket,
f.titre AS film,
p.date_projection AS date,
c.nom || ' ' || c.prenom AS client,
t.prix
FROM ticket t
JOIN projection p ON t.id_projection = p.id_projection
JOIN film f ON p.id_film = f.id_film
JOIN client c ON t.id_client = c.id_client;
```

✅ 2. Liste des projections avec le titre du film et le nom de la salle

sql

```
SELECT
p.id_projection,
f.titre AS film,
s.nom_salle AS salle,
p.date_projection,
p.heure_debut,
p.heure_fin
FROM projection p
JOIN film f ON p.id_film = f.id_film
JOIN salle s ON p.id_salle = s.id_salle;
```

✅ 3. Afficher les notes et commentaires donnés pour chaque film

sql

```
SELECT
f.titre AS film,
a.note,
a.commentaire,
c.nom || ' ' || c.prenom AS client,
a.date_avis
FROM avis a
JOIN film f ON a.id_film = f.id_film
JOIN client c ON a.id_client = c.id_client;
```

✅ 4. Clients ayant assisté à plus de 3 projections

sql

```
SELECT
c.nom || ' ' || c.prenom AS client,
COUNT(t.id_ticket) AS nombre_projections
FROM ticket t
JOIN client c ON t.id_client = c.id_client
GROUP BY c.id_client
HAVING COUNT(t.id_ticket) > 3;
```

🔍 Requêtes SQL Avancées

✅ 1. Film le plus projeté ce mois-ci

sql

```
SELECT f.titre, COUNT(*) AS nb_projections
FROM projection p
JOIN film f ON p.id_film = f.id_film
WHERE strftime('%m', p.date_projection) = strftime('%m', 'now')
AND strftime('%Y', p.date_projection) = strftime('%Y', 'now')
GROUP BY f.id_film
ORDER BY nb_projections DESC
LIMIT 1;
```

✅ 2. Moyenne des notes par film

sql

```
SELECT f.titre, AVG(a.note) AS moyenne_note
FROM avis a
JOIN film f ON a.id_film = f.id_film
GROUP BY f.id_film;
```

✅ 3. Salles les plus utilisées

sql

```
SELECT s.nom_salle, COUNT(*) AS nb_projections
FROM projection p
JOIN salle s ON p.id_salle = s.id_salle
GROUP BY s.id_salle
ORDER BY nb_projections DESC;
```

✅ 4. Clients ayant donné un avis sans avoir acheté de ticket

sql

```
SELECT DISTINCT c.nom, c.prenom
FROM client c
JOIN avis a ON c.id_client = a.id_client
WHERE c.id_client NOT IN (
SELECT t.id_client FROM ticket t
);
```

✅ 5. Total des revenus générés par jour

sql

```
SELECT date_achat, SUM(prix) AS revenu_total
FROM ticket
GROUP BY date_achat;
```

✅ Par semaine (en utilisant strftime('%W', date) pour la semaine de l'année)

sql

```
SELECT strftime('%Y-%W', date_achat) AS semaine, SUM(prix) AS revenu_total
FROM ticket
GROUP BY semaine;
```

✅ 6. Vue films_populaires (titre, nombre de tickets vendus, note moyenne)

sql

```
CREATE VIEW films_populaires AS
SELECT
f.titre,
COUNT(t.id_ticket) AS nb_tickets,
ROUND(AVG(a.note), 2) AS note_moyenne
FROM film f
LEFT JOIN projection p ON f.id_film = p.id_film
LEFT JOIN ticket t ON p.id_projection = t.id_projection
LEFT JOIN avis a ON f.id_film = a.id_film
GROUP BY f.id_film;
```

✅ 7. Fonction revenu_par_film(id_film) – total des gains générés par un film

PostgreSQL (exemple fonctionnelle, ajustable selon le SGBD utilisé)

sql

```
CREATE OR REPLACE FUNCTION revenu_par_film(id_f INT)
RETURNS NUMERIC AS $$
DECLARE
total NUMERIC;
BEGIN
SELECT SUM(t.prix) INTO total
FROM ticket t
JOIN projection p ON t.id_projection = p.id_projection
WHERE p.id_film = id_f;

RETURN COALESCE(total, 0);
END;
$$ LANGUAGE plpgsql;
```