MISSION 1 — Création de la base et des tables

```
CREATE DATABASE educore;

USE educore;

CREATE TABLE users (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL,

   email VARCHAR(150) NOT NULL UNIQUE,

   created_at DATETIME DEFAULT CURRENT_TIMESTAMP

);


CREATE TABLE courses (

   id INT AUTO_INCREMENT PRIMARY KEY,

   title VARCHAR(150) NOT NULL,

   price DECIMAL(10,2) NOT NULL CHECK (price > 0)

);


CREATE TABLE enrollments (

   id INT AUTO_INCREMENT PRIMARY KEY,

   user_id INT,

   course_id INT,

   progress INT DEFAULT 0 CHECK (progress BETWEEN 0 AND 100),

   FOREIGN KEY (user_id) REFERENCES users(id),

   FOREIGN KEY (course_id) REFERENCES courses(id)

);


CREATE TABLE payments (

   id INT AUTO_INCREMENT PRIMARY KEY,

   user_id INT,

   amount DECIMAL(10,2) NOT NULL,

   paid_at DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (user_id) REFERENCES users(id)

);
```

MISSION 2 — Insertion des données

```
INSERT INTO users (name, email) VALUES

('Aboudi', 'aboudi12@gmail.com'),

('Abid', 'abid@gmail.com'),

('Hassan', 'hassan1@gmail.com'),

('Frank', 'frank123@gmail.com'),

('Cherif', 'cherif@gmail.com'),

('frederick', 'frederick@gmail.com'),

('Brahim', 'brahim02@gmail.com'),

('Saleh', 'saleh002@gmail.com'),

('Armend', 'armend@gmail.com'),

('Babs', 'babs0012@gmail.com');

INSERT INTO courses (title, price) VALUES

('SQL Avancé', 100.00),

('Maths Avancé', 29.99),

('C++ Avancé', 19.99),

('Linux Avancé', 49.99),

('Algorithme Avancé', 30.99),

('Francais Avancé', 90.99);
```

MISSION 3 — Analyses marketing

```
SELECT c.title, COUNT(e.id) AS nb_inscrits

FROM courses c
```

```
JOIN enrollments e ON c.id = e.course_id

GROUP BY c.id;


SELECT u.name, COUNT(p.id) AS nb_paiements

FROM users u

JOIN payments p ON u.id = p.user_id

GROUP BY u.id

HAVING COUNT(p.id) >= 2;


SELECT u.name

FROM users u

LEFT JOIN payments p ON u.id = p.user_id

WHERE p.id IS NULL;
```

MISSION 4 — Analyse pédagogique

```
SELECT c.title, AVG(e.progress) AS progression_moyenne

FROM courses c

JOIN enrollments e ON c.id = e.course_id

GROUP BY c.id;


SELECT u.name, c.title, e.progress

FROM enrollments e

JOIN users u ON e.user_id = u.id

JOIN courses c ON e.course_id = c.id

WHERE e.progress < 25;
```

MISSION 5 — Sous-requêtes

```
SELECT title, price
FROM courses
WHERE price > (SELECT AVG(price) FROM courses);

SELECT u.name, COUNT(e.course_id) AS nb_cours
FROM users u
JOIN enrollments e ON u.id = e.user_id
GROUP BY u.id
HAVING COUNT(e.course_id) >= 2;
```

MISSION 6 — Vues

```
CREATE VIEW v_active_users AS
SELECT DISTINCT u.id, u.name, u.email
FROM users u
JOIN enrollments e ON u.id = e.user_id;

CREATE VIEW v_monthly_revenue AS
SELECT DATE_FORMAT(paid_at,'%Y-%m') AS mois,
SUM(amount) AS revenu
FROM payments
GROUP BY mois;
```

MISSION 7 — Index et performance

```
EXPLAIN SELECT * FROM users WHERE email = 'abid@gmail.com';

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_payments_paid_at ON payments(paid_at);
CREATE INDEX idx_enrollments_course_user
ON enrollments(course_id, user_id);
```

MISSION 8 — Transactions

SECURITER DES OPERATIONS TELLES QUE LES PAIEMENTS

```
START TRANSACTION;
INSERT INTO payments (user_id, amount) VALUES (1, 99.99);
COMMIT;
```

PERMISSION D’ANNULATION EN CAS D’ERREUR

```
START TRANSACTION;
INSERT INTO payments (user_id, amount) VALUES (1, -99.99);
ROLLBACK;
```