PROJECT – STAGE 2 :  
Yatharth Shah

Database Creation Questions :  
  
CREATE TABLE StudentDemographics (

Student\_ID INT PRIMARY KEY,

First\_Name VARCHAR(50),

Last\_Name VARCHAR(50),

Age INT,

Gender VARCHAR(50)

);  
  
CREATE TABLE StudentAcademics (

Student\_ID INT PRIMARY KEY,

First\_Name VARCHAR(50),

Last\_Name VARCHAR(50),

Major VARCHAR(50),

Year\_In\_School VARCHAR(50),

FOREIGN KEY (Student\_ID) REFERENCES StudentDemographics(Student\_ID)

);  
  
CREATE TABLE StudentFinances (

Student\_ID INT PRIMARY KEY,

Monthly\_Income INT,

Financial\_Aid INT,

Tuition INT,

FOREIGN KEY (Student\_ID) REFERENCES StudentDemographics(Student\_ID)

);

CREATE TABLE StudentExpenses (

Student\_ID INT PRIMARY KEY,

Housing INT,

Food INT,

Transportation INT,

Books\_Supplies INT,

Entertainment INT,

Personal\_Care INT,

Technology INT,

Health\_Wellness INT,

Miscellaneous INT,

FOREIGN KEY (Student\_ID) REFERENCES StudentDemographics(Student\_ID)

);

CREATE TABLE StudentPaymentMethods (

Student\_ID INT PRIMARY KEY,

Preferred\_Payment\_Method VARCHAR(50),

FOREIGN KEY (Student\_ID) REFERENCES StudentDemographics(Student\_ID)

);  
  
  
Screenshots of the tables :

1) Student Demographics :

A screenshot of a computer screen

Description automatically generated

2) Student Academics :

A screenshot of a computer

Description automatically generated

3) Student Finances :

A screenshot of a computer screen

Description automatically generated

4) Student Expenses :

A screenshot of a computer screen

Description automatically generated

5) Student Payment :

A screen shot of a computer

Description automatically generated

For creating the records :   
  
INSERT INTO StudentDemographics (Student\_ID, First\_Name, Last\_Name, Age, Gender) VALUES

(0, 'John', 'Doe', 19, 'Non-binary'),

(1, 'Garry', 'Potter', 24, 'Male'),

(2, 'Cristiano', 'Ronaldo', 24, 'Male'),

(3, 'Lionel', 'Messi', 23, 'Male'),

(4, 'Anushka', 'Kohli', 20, 'Female'),

(5, 'Jude', 'Bellingham', 25, 'Non-binary'),

(6, 'Phil', 'Foden', 23, 'Female'),

(7, 'Jack', 'Grealish', 23, 'Female'),

(8, 'Lewis', 'Hamilton', 22, 'Non-binary'),

(9, 'Selina', 'Gomez', 18, 'Female'),

(10, 'Arjun', 'Kapoor', 23, 'Male'),

(11, 'Yatharth', 'Shah', 25, 'Male'),

(12, 'Bukayo', 'Saka', 21, 'Male'),

(13, 'Maria', 'Lopez', 19, 'Female'),

(14, 'Axel', 'Santino', 24, 'Non-binary'),

(15, 'Mario', 'Luiz', 22, 'Non-binary'),

(16, 'Lamin', 'Yamal', 23, 'Female'),

(17, 'Surya', 'Yadav', 21, 'Female'),

(18, 'Isha', 'Ambani', 19, 'Female'),

(19, 'Nita', 'Merchant', 20, 'Female'),

(20, 'Wayne', 'Rooney', 19, 'Male'),

(21, 'Suneel', 'Chettri', 24, 'Male');  
  
INSERT INTO StudentAcademics (Student\_ID, First\_Name, Last\_Name, Major, Year\_In\_School) VALUES

(0, 'John', 'Doe', 'Psychology', 'Freshman'),

(1, 'Garry', 'Potter', 'Economics', 'Junior'),

(2, 'Cristiano', 'Ronaldo', 'Economics', 'Junior'),

(3, 'Lionel', 'Messi', 'Computer Science', 'Senior'),

(4, 'Anushka', 'Kohli', 'Computer Science', 'Senior'),

(5, 'Jude', 'Bellingham', 'Computer Science', 'Sophomore'),

(6, 'Phil', 'Foden', 'Engineering', 'Freshman'),

(7, 'Jack', 'Grealish', 'Economics', 'Junior'),

(8, 'Lewis', 'Hamilton', 'Computer Science', 'Senior'),

(9, 'Selina', 'Gomez', 'Computer Science', 'Junior'),

(10, 'Arjun', 'Kapoor', 'Biology', 'Junior'),

(11, 'Yatharth', 'Shah', 'Engineering', 'Freshman'),

(12, 'Bukayo', 'Saka', 'Economics', 'Sophomore'),

(13, 'Maria', 'Lopez', 'Biology', 'Junior'),

(14, 'Axel', 'Santino', 'Economics', 'Freshman'),

(15, 'Mario', 'Luiz', 'Computer Science', 'Sophomore'),

(16, 'Lamin', 'Yamal', 'Psychology', 'Freshman'),

(17, 'Surya', 'Yadav', 'Computer Science', 'Freshman'),

(18, 'Isha', 'Ambani', 'Psychology', 'Sophomore'),

(19, 'Nita', 'Merchant', 'Engineering', 'Senior'),

(20, 'Wayne', 'Rooney', 'Psychology', 'Junior'),

(21, 'Suneel', 'Chettri', 'Computer Science', 'Sophomore');  
  
  
INSERT INTO StudentFinances (Student\_ID, Monthly\_Income, Financial\_Aid, Tuition) VALUES

(0, 958, 270, 5939),

(1, 1006, 875, 4908),

(2, 734, 928, 3051),

(3, 617, 265, 4935),

(4, 810, 522, 3887),

(5, 523, 790, 3151),

(6, 1354, 69, 4973),

(7, 631, 748, 3966),

(8, 1402, 248, 5638),

(9, 1423, 74, 3977),

(10, 762, 615, 4093),

(11, 1068, 19, 5138),

(12, 719, 540, 4863),

(13, 1176, 859, 5201),

(14, 1496, 75, 5301),

(15, 1227, 26, 4175),

(16, 1419, 237, 5450),

(17, 1454, 851, 3538),

(18, 1487, 311, 4401),

(19, 1157, 401, 3131),

(20, 1180, 732, 4206),

(21, 516, 549, 4878);  
  
  
INSERT INTO StudentExpenses (Student\_ID, Housing, Food, Transportation, Books\_Supplies, Entertainment, Personal\_Care, Technology, Health\_Wellness, Miscellaneous) VALUES

(0, 709, 296, 123, 188, 41, 78, 134, 127, 72),

(1, 557, 365, 85, 252, 74, 92, 226, 129, 68),

(2, 666, 220, 137, 99, 130, 23, 239, 112, 133),

(3, 652, 289, 114, 223, 99, 30, 163, 105, 55),

(4, 825, 372, 168, 194, 71, 88, 71, 104, 104),

(5, 413, 386, 122, 131, 73, 38, 234, 108, 99),

(6, 812, 398, 101, 213, 21, 38, 157, 117, 48),

(7, 571, 269, 92, 251, 90, 152, 56, 62, 62),

(8, 599, 354, 82, 155, 123, 41, 162, 172, 194),

(9, 626, 249, 117, 123, 74, 243, 34, 196, 196),

(10, 660, 262, 58, 183, 98, 21, 274, 66, 50),

(11, 734, 243, 200, 228, 57, 57, 209, 193, 146),

(12, 894, 280, 120, 126, 41, 50, 78, 113, 114),

(13, 626, 234, 50, 274, 77, 125, 199, 103, 103),

(14, 657, 151, 63, 92, 124, 64, 196, 50, 117),

(15, 884, 328, 167, 175, 111, 96, 73, 154, 109),

(16, 883, 361, 200, 141, 76, 22, 285, 185, 76),

(17, 754, 357, 110, 188, 33, 24, 50, 74, 46),

(18, 883, 292, 69, 51, 75, 60, 105, 181, 164),

(19, 886, 196, 131, 216, 57, 39, 264, 39, 174),

(20, 597, 254, 195, 67, 51, 21, 97, 185, 46),

(21, 814, 286, 180, 282, 55, 25, 77, 73, 184);  
  
  
INSERT INTO StudentPaymentMethods (Student\_ID, Preferred\_Payment\_Method) VALUES

(0, 'Credit/Debit Card'),

(1, 'Credit/Debit Card'),

(2, 'Cash'),

(3, 'Mobile Payment App'),

(4, 'Credit/Debit Card'),

(5, 'Mobile Payment App'),

(6, 'Credit/Debit Card'),

(7, 'Mobile Payment App'),

(8, 'Credit/Debit Card'),

(9, 'Mobile Payment App'),

(10, 'Credit/Debit Card'),

(11, 'Credit/Debit Card'),

(12, 'Credit/Debit Card'),

(13, 'Mobile Payment App'),

(14, 'Cash'),

(15, 'Cash'),

(16, 'Cash'),

(17, 'Mobile Payment App'),

(18, 'Credit/Debit Card'),

(19, 'Mobile Payment App'),

(20, 'Credit/Debit Card'),

(21, 'Mobile Payment App');  
  
  
All of your tables are in Boyce-Codd Normal Form (BCNF) because the primary key is the only determinant that has total control over all other features. There are no partial or transitive dependencies, therefore any non-trivial functional dependency has a super key as a determinant.  
  
CLI question :  
  
SQL Queries for

Inserting Data :   
  
INSERT INTO studentdemographics (student\_id, first\_name, last\_name, age, gender) VALUES (%s, %s, %s, %s, %s)

INSERT INTO studentacademics (student\_id, first\_name, last\_name, major, year\_in\_school) VALUES (%s, %s, %s, %s, %s)

INSERT INTO studentexpenses (student\_id, housing, food, transportation, books\_supplies, entertainment, personal\_care, technology, health\_wellness, miscellaneous) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

INSERT INTO studentfinances (student\_id, monthly\_income, financial\_aid, tuition) VALUES (%s, %s, %s, %s)

INSERT INTO studentpaymentmethods (student\_id, preferred\_payment\_method) VALUES (%s, %s)

Updating Data :  
  
UPDATE studentdemographics SET first\_name = %s, last\_name = %s, age = %s, gender = %s WHERE student\_id = %s

UPDATE studentacademics SET major = %s, year\_in\_school = %s WHERE student\_id = %s

UPDATE studentexpenses SET housing = %s, food = %s WHERE student\_id = %s

UPDATE studentfinances SET monthly\_income = %s, financial\_aid = %s WHERE student\_id = %s

UPDATE studentpaymentmethods SET preferred\_payment\_method = %s WHERE student\_id = %s

Deleting Data :

SELECT d.student\_id, d.first\_name, d.last\_name, d.age, d.gender, a.major, a.year\_in\_school, e.housing, e.food, e.transportation, e.books\_supplies, e.entertainment, e.personal\_care, e.technology, e.health\_wellness, e.miscellaneous, f.monthly\_income, f.financial\_aid, f.tuition, p.preferred\_payment\_method FROM studentdemographics d LEFT JOIN studentacademics a ON d.student\_id = a.student\_id LEFT JOIN studentexpenses e ON d.student\_id = e.student\_id LEFT JOIN studentfinances f ON d.student\_id = f.student\_id LEFT JOIN studentpaymentmethods p ON d.student\_id = p.student\_id ORDER BY d.student\_id;

Sorting Data :   
  
SELECT \* FROM studentdemographics ORDER BY %s %s

SELECT \* FROM studentacademics ORDER BY %s %s

SELECT \* FROM studentexpenses ORDER BY %s %s

SELECT \* FROM studentfinances ORDER BY %s %s

SELECT \* FROM studentpaymentmethods ORDER BY %s %s  
  
  
Grouping Data :   
  
SELECT gender, COUNT(\*) FROM studentdemographics GROUP BY gender

SELECT major, COUNT(\*) FROM studentacademics GROUP BY major

SELECT 'Housing', SUM(housing) FROM studentexpenses UNION ALL SELECT 'Food', SUM(food) FROM studentexpenses UNION ALL SELECT 'Transportation', SUM(transportation) FROM studentexpenses

SELECT CASE WHEN monthly\_income <= 500 THEN '0-500' WHEN monthly\_income > 500 AND monthly\_income <= 1000 THEN '501-1000' ELSE '1001+' END AS income\_range, COUNT(\*) FROM studentfinances GROUP BY income\_range

SELECT preferred\_payment\_method, COUNT(\*) FROM studentpaymentmethods GROUP BY preferred\_payment\_method

Finding Students Above Average Expenses :   
  
SELECT student\_id, housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous AS total\_expenses FROM studentexpenses WHERE (housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous) > (SELECT AVG(housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous) FROM studentexpenses)

Aggregating student Finances :   
  
SELECT AVG(monthly\_income) AS average\_income, SUM(financial\_aid) AS total\_aid FROM studentfinances  
  
  
Screenshots :   
  
Insert :  
  
A black rectangle with white text

Description automatically generated

Update :

A black background with white text

Description automatically generated

Delete Data :  
  
A black screen with white text

Description automatically generated

List All Data :

A screenshot of a computer

Description automatically generated

Search Data :

A screen shot of a computer

Description automatically generated

Sort Data :

A screenshot of a computer

Description automatically generated

Group Data :   
  
A black rectangle with white text

Description automatically generated

Students Above Average Expenses :

A black rectangle with white text

Description automatically generated

Aggregate Student Finances :

A black and blue rectangle

Description automatically generated

Code :

import psycopg2

def connect\_to\_db():

"""Connect to the PostgreSQL database server."""

conn = None

try:

# Set your connection parameters here

conn = psycopg2.connect(

dbname="postgres",

user="yatharthshah",

password="Y27",

host="localhost"

)

return conn

except Exception as error:

print(f"Error: {error}")

return None

# INSERT DATA :

def insert\_student\_demographics(conn):

cursor = conn.cursor()

print("Inserting into student demographics...")

student\_id = input("Enter student ID: ")

first\_name = input("Enter first name: ")

last\_name = input("Enter last name: ")

age = input("Enter age: ")

gender = input("Enter gender: ")

try:

cursor.execute("""

INSERT INTO studentdemographics (student\_id, first\_name, last\_name, age, gender)

VALUES (%s, %s, %s, %s, %s)

""", (student\_id, first\_name, last\_name, age, gender))

conn.commit()

print("Student demographic data inserted successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def insert\_student\_academics(conn):

cursor = conn.cursor()

print("Inserting into student academics...")

student\_id = input("Enter student ID: ")

first\_name = input("Enter first name: ")

last\_name = input("Enter last name: ")

major = input("Enter major: ")

year\_in\_school = input("Enter year in school: ")

try:

cursor.execute("""

INSERT INTO studentacademics (student\_id, first\_name, last\_name, major, year\_in\_school)

VALUES (%s, %s, %s, %s, %s)

""", (student\_id, first\_name, last\_name, major, year\_in\_school))

conn.commit()

print("Student academic data inserted successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def insert\_student\_expenses(conn):

cursor = conn.cursor()

print("Inserting into student expenses...")

student\_id = input("Enter student ID: ")

housing = input("Enter housing expenses: ")

food = input("Enter food expenses: ")

transportation = input("Enter transportation expenses: ")

books\_supplies = input("Enter books and supplies expenses: ")

entertainment = input("Enter entertainment expenses: ")

personal\_care = input("Enter personal care expenses: ")

technology = input("Enter technology expenses: ")

health\_wellness = input("Enter health and wellness expenses: ")

miscellaneous = input("Enter miscellaneous expenses: ")

try:

cursor.execute("""

INSERT INTO studentexpenses (student\_id, housing, food, transportation, books\_supplies, entertainment, personal\_care, technology, health\_wellness, miscellaneous)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

""", (student\_id, housing, food, transportation, books\_supplies, entertainment, personal\_care, technology, health\_wellness, miscellaneous))

conn.commit()

print("Student expense data inserted successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def insert\_student\_finances(conn):

cursor = conn.cursor()

print("Inserting into student finances...")

student\_id = input("Enter student ID: ")

monthly\_income = input("Enter monthly income: ")

financial\_aid = input("Enter financial aid amount: ")

tuition = input("Enter tuition amount: ")

try:

cursor.execute("""

INSERT INTO studentfinances (student\_id, monthly\_income, financial\_aid, tuition)

VALUES (%s, %s, %s, %s)

""", (student\_id, monthly\_income, financial\_aid, tuition))

conn.commit()

print("Student finance data inserted successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def insert\_student\_payment\_methods(conn):

cursor = conn.cursor()

print("Inserting into student payment methods...")

student\_id = input("Enter student ID: ")

preferred\_payment\_method = input("Enter preferred payment method: ")

try:

cursor.execute("""

INSERT INTO studentpaymentmethods (student\_id, preferred\_payment\_method)

VALUES (%s, %s)

""", (student\_id, preferred\_payment\_method))

conn.commit()

print("Student payment method data inserted successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def insert\_data\_menu(conn):

while True:

print("\nInsert Data Menu:")

print("1. Insert Student Demographics")

print("2. Insert Student Academics")

print("3. Insert Student Expenses")

print("4. Insert Student Finances")

print("5. Insert Student Payment Methods")

print("0. Go Back")

choice = input("Enter choice: ")

if choice == '1':

insert\_student\_demographics(conn)

elif choice == '2':

insert\_student\_academics(conn)

elif choice == '3':

insert\_student\_expenses(conn)

elif choice == '4':

insert\_student\_finances(conn)

elif choice == '5':

insert\_student\_payment\_methods(conn)

elif choice == '0':

break

else:

print("Invalid choice. Please try again.")

# UPDATE DATA :

def update\_student\_demographics(conn):

cursor = conn.cursor()

print("Updating student demographics...")

student\_id = input("Enter student ID to update: ")

first\_name = input("Enter new first name (leave blank if no change): ")

last\_name = input("Enter new last name (leave blank if no change): ")

age = input("Enter new age (leave blank if no change): ")

gender = input("Enter new gender (leave blank if no change): ")

updates = []

params = []

if first\_name:

updates.append("first\_name = %s")

params.append(first\_name)

if last\_name:

updates.append("last\_name = %s")

params.append(last\_name)

if age:

updates.append("age = %s")

params.append(age)

if gender:

updates.append("gender = %s")

params.append(gender)

if not updates:

print("No updates made.")

return

query = "UPDATE studentdemographics SET " + ", ".join(updates) + " WHERE student\_id = %s;"

params.append(student\_id)

try:

cursor.execute(query, params)

conn.commit()

print("Student demographic data updated successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def update\_student\_academics(conn):

cursor = conn.cursor()

print("Updating student academics...")

student\_id = input("Enter student ID to update: ")

major = input("Enter new major (leave blank if no change): ")

year\_in\_school = input("Enter new year in school (leave blank if no change): ")

updates = []

params = []

if major:

updates.append("major = %s")

params.append(major)

if year\_in\_school:

updates.append("year\_in\_school = %s")

params.append(year\_in\_school)

if not updates:

print("No updates made.")

return

query = "UPDATE studentacademics SET " + ", ".join(updates) + " WHERE student\_id = %s;"

params.append(student\_id)

try:

cursor.execute(query, params)

conn.commit()

print("Student academics data updated successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def update\_student\_expenses(conn):

cursor = conn.cursor()

print("Updating student expenses...")

student\_id = input("Enter student ID to update: ")

housing = input("Enter new housing expense (leave blank if no change): ")

food = input("Enter new food expense (leave blank if no change): ")

updates = []

params = []

if housing:

updates.append("housing = %s")

params.append(housing)

if food:

updates.append("food = %s")

params.append(food)

if not updates:

print("No updates made.")

return

query = "UPDATE studentexpenses SET " + ", ".join(updates) + " WHERE student\_id = %s;"

params.append(student\_id)

try:

cursor.execute(query, params)

conn.commit()

print("Student expenses data updated successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def update\_student\_finances(conn):

cursor = conn.cursor()

print("Updating student finances...")

student\_id = input("Enter student ID to update: ")

monthly\_income = input("Enter new monthly income (leave blank if no change): ")

financial\_aid = input("Enter new financial aid amount (leave blank if no change): ")

updates = []

params = []

if monthly\_income:

updates.append("monthly\_income = %s")

params.append(monthly\_income)

if financial\_aid:

updates.append("financial\_aid = %s")

params.append(financial\_aid)

if not updates:

print("No updates made.")

return

query = "UPDATE studentfinances SET " + ", ".join(updates) + " WHERE student\_id = %s;"

params.append(student\_id)

try:

cursor.execute(query, params)

conn.commit()

print("Student finances data updated successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def update\_student\_payment\_methods(conn):

cursor = conn.cursor()

print("Updating student payment methods...")

student\_id = input("Enter student ID to update: ")

preferred\_payment\_method = input("Enter new preferred payment method (leave blank if no change): ")

if not preferred\_payment\_method:

print("No updates made.")

return

query = "UPDATE studentpaymentmethods SET preferred\_payment\_method = %s WHERE student\_id = %s;"

params = (preferred\_payment\_method, student\_id)

try:

cursor.execute(query, params)

conn.commit()

print("Student payment method updated successfully.")

except Exception as e:

print("Error:", e)

conn.rollback()

finally:

cursor.close()

def data\_menu(conn, action):

while True:

print(f"\n{action} Data Menu:")

print("1. Student Demographics")

print("2. Student Academics")

print("3. Student Expenses")

print("4. Student Finances")

print("5. Student Payment Methods")

print("0. Go Back")

choice = input("Enter choice: ")

if choice == '1':

if action == "Insert":

insert\_student\_demographics(conn)

elif action == "Update":

update\_student\_demographics(conn)

elif choice == '2':

if action == "Insert":

insert\_student\_academics(conn)

elif action == "Update":

update\_student\_academics(conn)

elif choice == '3':

if action == "Insert":

insert\_student\_expenses(conn)

elif action == "Update":

update\_student\_expenses(conn)

elif choice == '4':

if action == "Insert":

insert\_student\_finances(conn)

elif action == "Update":

update\_student\_finances(conn)

elif choice == '5':

if action == "Insert":

insert\_student\_payment\_methods(conn)

elif action == "Update":

update\_student\_payment\_methods(conn)

elif choice == '0':

break

else:

print("Invalid choice. Please try again.")

# DELETE DATA :

def delete\_student\_data(conn):

student\_id = input("Enter student ID to delete all related records, or type 'cancel' to go back: ")

# Provide an option to cancel the operation

if student\_id.lower() == 'cancel':

print("Delete operation cancelled.")

return

cursor = conn.cursor()

try:

# Proceed with the deletion if not cancelled

print("Proceeding with deletion...")

# Delete from child tables first to respect foreign key constraints

cursor.execute("DELETE FROM studentexpenses WHERE student\_id = %s", (student\_id,))

cursor.execute("DELETE FROM studentfinances WHERE student\_id = %s", (student\_id,))

cursor.execute("DELETE FROM studentpaymentmethods WHERE student\_id = %s", (student\_id,))

cursor.execute("DELETE FROM studentacademics WHERE student\_id = %s", (student\_id,))

cursor.execute("DELETE FROM studentdemographics WHERE student\_id = %s", (student\_id,))

conn.commit()

print("All data related to student ID", student\_id, "has been deleted successfully.")

except Exception as e:

print("Failed to delete data:", e)

conn.rollback()

finally:

cursor.close()

# LIST ALL DATA :

def list\_all\_student\_data(conn):

print("Listing all student data in a unified view...")

cursor = conn.cursor()

try:

# Execute a SQL query that joins all relevant student tables

query = """

SELECT d.student\_id, d.first\_name, d.last\_name, d.age, d.gender,

a.major, a.year\_in\_school,

e.housing, e.food, e.transportation, e.books\_supplies, e.entertainment, e.personal\_care, e.technology, e.health\_wellness, e.miscellaneous,

f.monthly\_income, f.financial\_aid, f.tuition,

p.preferred\_payment\_method

FROM studentdemographics d

LEFT JOIN studentacademics a ON d.student\_id = a.student\_id

LEFT JOIN studentexpenses e ON d.student\_id = e.student\_id

LEFT JOIN studentfinances f ON d.student\_id = f.student\_id

LEFT JOIN studentpaymentmethods p ON d.student\_id = p.student\_id

ORDER BY d.student\_id;

"""

cursor.execute(query)

records = cursor.fetchall()

# Print column headers

print("\nUnified Student Data:")

headers = ["Student ID", "First Name", "Last Name", "Age", "Gender",

"Major", "Year in School",

"Housing", "Food", "Transportation", "Books Supplies", "Entertainment", "Personal Care", "Technology", "Health Wellness", "Miscellaneous",

"Monthly Income", "Financial Aid", "Tuition",

"Preferred Payment Method"]

print("\t".join(headers))

# Print each row of the data

for row in records:

print("\t".join(str(x) for x in row))

except Exception as e:

print("Failed to retrieve data:", e)

finally:

cursor.close()

# SEARCH STUDENT DATA :

def search\_student\_data(conn):

student\_id = input("Enter student ID to search for: ")

cursor = conn.cursor()

try:

# Execute a SQL query that joins all relevant student tables

query = """

SELECT d.student\_id, d.first\_name, d.last\_name, d.age, d.gender,

a.major, a.year\_in\_school,

e.housing, e.food, e.transportation, e.books\_supplies, e.entertainment, e.personal\_care, e.technology, e.health\_wellness, e.miscellaneous,

f.monthly\_income, f.financial\_aid, f.tuition,

p.preferred\_payment\_method

FROM studentdemographics d

LEFT JOIN studentacademics a ON d.student\_id = a.student\_id

LEFT JOIN studentexpenses e ON d.student\_id = e.student\_id

LEFT JOIN studentfinances f ON d.student\_id = f.student\_id

LEFT JOIN studentpaymentmethods p ON d.student\_id = p.student\_id

WHERE d.student\_id = %s;

"""

cursor.execute(query, (student\_id,))

record = cursor.fetchone()

# Print the results

if record:

headers = ["Student ID", "First Name", "Last Name", "Age", "Gender",

"Major", "Year in School",

"Housing", "Food", "Transportation", "Books Supplies", "Entertainment", "Personal Care", "Technology", "Health Wellness", "Miscellaneous",

"Monthly Income", "Financial Aid", "Tuition",

"Preferred Payment Method"]

print("\nDetailed Information for Student ID:", student\_id)

print("\t".join(headers))

print("\t".join(str(x) for x in record))

else:

print("No data found for Student ID:", student\_id)

except Exception as e:

print("Failed to retrieve data:", e)

finally:

cursor.close()

# AGGREGATE STUDENT FINANCES :

def aggregate\_student\_finances(conn):

cursor = conn.cursor()

try:

cursor.execute("SELECT AVG(monthly\_income) AS average\_income, SUM(financial\_aid) AS total\_aid FROM studentfinances")

result = cursor.fetchone()

print(f"Average Monthly Income: {result[0]:.2f}")

print(f"Total Financial Aid: {result[1]:.2f}")

except Exception as e:

print("Failed to compute finance aggregates:", e)

finally:

cursor.close()

# SORT

def sort\_student\_demographics(conn):

cursor = conn.cursor()

print("Sorting Student Demographics...")

column = input("Enter column to sort by (e.g., age, gender): ")

order = input("Choose order (ASC for ascending, DESC for descending): ")

try:

cursor.execute(f"SELECT \* FROM studentdemographics ORDER BY {column} {order}")

records = cursor.fetchall()

for record in records:

print(record)

except Exception as e:

print("Error sorting student demographics:", e)

finally:

cursor.close()

def sort\_student\_academics(conn):

cursor = conn.cursor()

print("Sorting Student Academics...")

column = input("Enter column to sort by (e.g., major, year\_in\_school): ")

order = input("Choose order (ASC for ascending, DESC for descending): ")

try:

cursor.execute(f"SELECT \* FROM studentacademics ORDER BY {column} {order}")

records = cursor.fetchall()

for record in records:

print(record)

except Exception as e:

print("Error sorting student academics:", e)

finally:

cursor.close()

def sort\_student\_expenses(conn):

cursor = conn.cursor()

print("Sorting Student Expenses...")

column = input("Enter column to sort by (e.g., housing, food): ")

order = input("Choose order (ASC for ascending, DESC for descending): ")

try:

cursor.execute(f"SELECT \* FROM studentexpenses ORDER BY {column} {order}")

records = cursor.fetchall()

for record in records:

print(record)

except Exception as e:

print("Error sorting student expenses:", e)

finally:

cursor.close()

def sort\_student\_finances(conn):

cursor = conn.cursor()

print("Sorting Student Finances...")

column = input("Enter column to sort by (e.g., monthly\_income, financial\_aid): ")

order = input("Choose order (ASC for ascending, DESC for descending): ")

try:

cursor.execute(f"SELECT \* FROM studentfinances ORDER BY {column} {order}")

records = cursor.fetchall()

for record in records:

print(record)

except Exception as e:

print("Error sorting student finances:", e)

finally:

cursor.close()

def sort\_student\_payment\_methods(conn):

cursor = conn.cursor()

print("Sorting Student Payment Methods...")

column = input("Enter column to sort by (e.g., preferred\_payment\_method): ")

order = input("Choose order (ASC for ascending, DESC for descending): ")

try:

cursor.execute(f"SELECT \* FROM studentpaymentmethods ORDER BY {column} {order}")

records = cursor.fetchall()

for record in records:

print(record)

except Exception as e:

print("Error sorting student payment methods:", e)

finally:

cursor.close()

def sort\_data\_menu(conn):

while True:

print("\nSort Data Menu:")

print("1. Sort Student Demographics")

print("2. Sort Student Academics")

print("3. Sort Student Expenses")

print("4. Sort Student Finances")

print("5. Sort Student Payment Methods")

print("0. Go Back")

choice = input("Enter choice: ")

if choice == '1':

sort\_student\_demographics(conn)

elif choice == '2':

sort\_student\_academics(conn)

elif choice == '3':

sort\_student\_expenses(conn)

elif choice == '4':

sort\_student\_finances(conn)

elif choice == '5':

sort\_student\_payment\_methods(conn)

elif choice == '0':

break

else:

print("Invalid choice. Please try again.")

# GROUP BY :

def group\_student\_demographics\_by\_gender(conn):

cursor = conn.cursor()

print("Grouping Student Demographics by Gender...")

try:

cursor.execute("SELECT gender, COUNT(\*) FROM studentdemographics GROUP BY gender")

records = cursor.fetchall()

for record in records:

print(f"Gender: {record[0]}, Count: {record[1]}")

except Exception as e:

print("Error grouping student demographics by gender:", e)

finally:

cursor.close()

def group\_student\_academics\_by\_major(conn):

cursor = conn.cursor()

print("Grouping Student Academics by Major...")

try:

cursor.execute("SELECT major, COUNT(\*) FROM studentacademics GROUP BY major")

records = cursor.fetchall()

for record in records:

print(f"Major: {record[0]}, Count: {record[1]}")

except Exception as e:

print("Error grouping student academics by major:", e)

finally:

cursor.close()

def group\_student\_expenses\_by\_category(conn):

cursor = conn.cursor()

print("Grouping Student Expenses by Category Total...")

try:

cursor.execute("""

SELECT 'Housing' AS category, SUM(housing) FROM studentexpenses

UNION ALL

SELECT 'Food', SUM(food) FROM studentexpenses

UNION ALL

SELECT 'Transportation', SUM(transportation) FROM studentexpenses

""")

records = cursor.fetchall()

for record in records:

print(f"Category: {record[0]}, Total: {record[1]}")

except Exception as e:

print("Error grouping student expenses by category:", e)

finally:

cursor.close()

def group\_student\_finances\_by\_income(conn):

cursor = conn.cursor()

print("Grouping Student Finances by Income Range...")

try:

cursor.execute("""

SELECT CASE

WHEN monthly\_income <= 500 THEN '0-500'

WHEN monthly\_income > 500 AND monthly\_income <= 1000 THEN '501-1000'

ELSE '1001+' END AS income\_range,

COUNT(\*) FROM studentfinances

GROUP BY income\_range

""")

records = cursor.fetchall()

for record in records:

print(f"Income Range: {record[0]}, Count: {record[1]}")

except Exception as e:

print("Error grouping student finances by income range:", e)

finally:

cursor.close()

def group\_student\_payment\_methods\_by\_type(conn):

cursor = conn.cursor()

print("Grouping Student Payment Methods by Type...")

try:

cursor.execute("SELECT preferred\_payment\_method, COUNT(\*) FROM studentpaymentmethods GROUP BY preferred\_payment\_method")

records = cursor.fetchall()

for record in records:

print(f"Payment Method: {record[0]}, Count: {record[1]}")

except Exception as e:

print("Error grouping student payment methods by type:", e)

finally:

cursor.close()

def group\_data\_menu(conn):

while True:

print("\nGroup Data Menu:")

print("1. Group Student Demographics by Gender")

print("2. Group Student Academics by Major")

print("3. Group Student Expenses by Category")

print("4. Group Student Finances by Income Range")

print("5. Group Student Payment Methods by Type")

print("0. Go Back")

choice = input("Enter choice: ")

if choice == '1':

group\_student\_demographics\_by\_gender(conn)

elif choice == '2':

group\_student\_academics\_by\_major(conn)

elif choice == '3':

group\_student\_expenses\_by\_category(conn)

elif choice == '4':

group\_student\_finances\_by\_income(conn)

elif choice == '5':

group\_student\_payment\_methods\_by\_type(conn)

elif choice == '0':

break

else:

print("Invalid choice. Please try again.")

# STUDENTS ABOVE AVERAGE :

def find\_students\_above\_average\_expenses(conn):

cursor = conn.cursor()

try:

cursor.execute("""

SELECT student\_id, housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous AS total\_expenses

FROM studentexpenses

WHERE (housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous) >

(SELECT AVG(housing + food + transportation + books\_supplies + entertainment + personal\_care + technology + health\_wellness + miscellaneous) FROM studentexpenses)

""")

records = cursor.fetchall()

for record in records:

print(f"Student ID: {record[0]}, Total Expenses: {record[1]}")

except Exception as e:

print("Error finding students with above-average expenses:", e)

finally:

cursor.close()

def main():

conn = connect\_to\_db()

if conn is not None:

while True:

print("\nMain Menu:")

print("1. Insert Data")

print("2. Update Data")

print("3. Delete Data")

print("4. List All Data")

print("5. Search Data")

print("6. Sort Data")

print("7. Group Data")

print("8. Students Above Average Expenses")

print("9. Aggregate Student Finances")

print("0. Exit")

choice = input("Enter choice: ")

if choice == '1':

data\_menu(conn, "Insert")

elif choice == '2':

data\_menu(conn, "Update")

elif choice == '3':

delete\_student\_data(conn)

elif choice == '4':

list\_all\_student\_data(conn)

elif choice == '5':

search\_student\_data(conn)

elif choice == '6':

sort\_data\_menu(conn)

elif choice == '7':

group\_data\_menu(conn)

elif choice == '8':

find\_students\_above\_average\_expenses(conn)

elif choice == '9':

aggregate\_student\_finances(conn)

elif choice == '0':

print("Exiting program.")

break

else:

print("Invalid choice. Please try again.")

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()