utils.py

```python
from PyQt6.QtGui import QIcon
from PyQt6.QtCore import QSize, Qt
from PyQt6.QtWidgets import QTableWidgetItem, QPushButton, QHBoxLayout, QLabel

def icon_button(txt, icon_name, handler):
    btn = QPushButton(txt)
    btn.clicked.connect(handler)
    if icon_name == "":
        return btn

    icon = QIcon(f"icons/{icon_name}.png")
    btn.setIconSize(QSize(28, 28))
    btn.setIcon(icon)
    return btn

def table_cell(txt):
    item = QTableWidgetItem(txt)
    item.setFlags(item.flags() ^ Qt.ItemFlag.ItemIsEditable)
    return item

def hbox(items):
    box = QHBoxLayout()
    for i in items:
        box.addWidget(QLabel(i))
    return box
```

invoice_editor.py

```python
from PyQt6.QtWidgets import *
from PyQt6.QtCore import Qt,QSize

from decimal import Decimal

from pdf import PdfFile
from utils import table_cell, icon_button, hbox

class InvoiceEditor(QMainWindow):
    def __init__(self, db, update=False, update_id=13, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.db = db
        self.update = update
        self.update_id = update_id

        self.setWindowTitle('Create Invoice')
        self.setFixedSize(950, 500)

        records = []
        if self.update:
            self.setWindowTitle("Edit Invoice")
            rows = self.db.get_records(self.update_id)

            for r in rows:
                print(r)
                records.append({
                    'Item Name': r[2],
                    'Price': r[3],
                    'Qty': r[4],
                    'Discount': r[5]
                })

        self.table = QTableWidget(self)
        self.table.setColumnCount(5)

        for (i, w) in enumerate([200, 100, 100, 100, 30]):
            self.table.setColumnWidth(i, w)

        headers = ['Item Name', 'Price (Rs)', 'Qty', "Disc (%)", ""]
        self.table.setHorizontalHeaderLabels(headers)
        self.table.setRowCount(len(records))

        for (row, e) in enumerate(records):
            self.table.setItem(row, 0, QTableWidgetItem(e['Item Name']))
            self.table.setItem(row, 2, QTableWidgetItem(str(e['Qty'])))
            self.table.setItem(row, 1, QTableWidgetItem(str(e['Price'])))
            self.table.setItem(row, 3, QTableWidgetItem(str(e['Discount'])))
            self.table.setCellWidget(row, 4, icon_button('', 'delete', self.delete))

        self.description = QLineEdit()
        if self.update:
            self.description.setText(self.db.get_invoice_info(self.update_id)[0])

        main_layout = QVBoxLayout()
        main_layout.addWidget(QLabel("Description: "))
        main_layout.addWidget(self.description)
        main_layout.addWidget(self.table)

        gen_invoice = None
        if self.update:
            gen_invoice = icon_button('Update Invoice', 'save', self.update_invoice)
        else:
            gen_invoice = icon_button('Generate Invoice', '', self.generate_invoice)

        main_layout.addWidget(gen_invoice)
        main_layout.addWidget(icon_button('Go Back', 'back', self.close))

        widget = QWidget()
        widget.setLayout(main_layout)
        self.setCentralWidget(widget)

        dock = QDockWidget('New Record')
        dock.setFeatures(QDockWidget.DockWidgetFeature.NoDockWidgetFeatures)
        self.addDockWidget(Qt.DockWidgetArea.RightDockWidgetArea, dock)

        form = QWidget()
        form_layout = QFormLayout(form)
        form.setLayout(form_layout)

        self.item_name = QLineEdit(form)
        self.price = QLineEdit(form)
        self.qty = QSpinBox(form, minimum=1, maximum=1000)
        self.qty.clear()
        self.discount = QSpinBox(form, minimum=0, maximum=100)
        self.discount.setValue(0)

        form_layout.addRow('Item Name:', self.item_name)
        form_layout.addRow('Price(Rs):', self.price)
        form_layout.addRow('Qty:', self.qty)
        form_layout.addRow('Discount(%) :', self.discount)
        form_layout.addRow(icon_button('Add', '', self.add_record))

        dock.setWidget(form)

    def delete(self):
        current_row = self.table.currentRow()

        button = QMessageBox.question(
            self,
            'Confirmation',
            'Are you sure that you want to delete the selected row?',
            QMessageBox.StandardButton.Yes |
            QMessageBox.StandardButton.No
        )
        if button == QMessageBox.StandardButton.Yes:
            self.table.removeRow(current_row)

    def valid(self):
        item_name = self.item_name.text().strip()

        if not item_name:
            QMessageBox.critical(self, 'Error', 'Please enter the item name')
            self.first_name.setFocus()
            return False

        try:
            price = float(self.price.text().strip())
            qty = int(self.qty.text().strip())
            discount = int(self.discount.text().strip())

        except ValueError:
            QMessageBox.critical(self, 'Error', 'Please enter valid data')
            return False

        if qty <= 0 or discount >= 100 or discount < 0:
            QMessageBox.critical(
                self, 'Error', 'Please enter valid data')
            return False
        return True

    def reset_form(self):
        self.item_name.clear()
        self.price.clear()
        self.qty.clear()
        self.discount.setValue(0)

    def add_record(self):
        if not self.valid():
            return

        table = self.table
        row = table.rowCount()
        table.insertRow(row)
        table.setItem(row, 0, QTableWidgetItem(self.item_name.text().strip()))
        table.setItem(row, 1, QTableWidgetItem(self.price.text().strip()))
        table.setItem(row, 2, QTableWidgetItem(self.qty.text()))
        table.setItem(row, 3, QTableWidgetItem(self.discount.text()))
        table.setCellWidget(row, 4, icon_button('', 'delete', self.delete))
        self.reset_form()

    def generate_invoice(self):
        rows = self.table.rowCount()
        desc = self.description.text().strip()
        if desc == "":
            QMessageBox.critical(self, 'Error', 'Description cannot be empty')
            return

        records = []
        for i in range(rows):
            item_name = self.table.item(i, 0).text().strip()

            try:
                price = float(self.table.item(i, 1).text().strip())
                qty = int(self.table.item(i, 2).text().strip())
                discount = int(self.table.item(i, 3).text().strip())

                records.append({
                    'item_name': item_name, 'price': price,
                    'qty': qty, 'discount': discount
                })
            except ValueError:
                QMessageBox.critical(self, 'Error', 'Please enter valid data')
                return

        i = self.update_id
        if not self.update:
            i = self.db.insert_invoice({'description': desc})

        total_amt = 0
        for r in records:
            r['invoice_id'] = i
            self.db.insert_record(r)

            amt = price * qty
            amt = amt - (amt * (discount / 100))
            total_amt += amt
        self.db.set_amount(i, total_amt)

        self.receipt = InvoiceReceipt(self.db)
        self.receipt.prepare(i)
        self.receipt.show()
        self.close()

    def update_invoice(self):
        self.db.delete_records(self.update_id)
        self.generate_invoice()

    def close(self):
        self.hide()

class InvoiceReceipt(QMainWindow):
    def __init__(self, db):
        super().__init__()
        self.setWindowTitle("Invoice Receipt")
        self.setFixedSize(890, 600)
        self.db = db

    def prepare(self, i):
        self.i = i
        rows = self.db.get_records(i)

        grid = QTableWidget()
        grid.setColumnCount(6)
        headers = ['S.No', 'Item Name', 'Price(Rs)', 'Qty', "Disc(%)", "Amount(Rs)"]
        grid.setHorizontalHeaderLabels(headers)
        grid.setRowCount(len(rows))

        col_widths = [50, 300, 150, 60, 100, 150]
        for (i, w) in enumerate(col_widths):
            grid.setColumnWidth(i, w)

        self.data = []
        for (i, r) in enumerate(rows):
            tmp = float(r[3]) * r[4]
            tmp -= tmp * (r[5] / 100)
            self.data.append({
                'Item Name': r[2],
                'Price': r[3],
                'Qty': r[4],
                'Discount': r[5],
                'Amount': tmp,
            })
            grid.setItem(i, 0, table_cell(str(i+1)))
            grid.setItem(i, 1, table_cell(r[2]))
            grid.setItem(i, 2, table_cell(f'{r[3]:,.2f}'))
            grid.setItem(i, 3, table_cell(str(r[4])))
            grid.setItem(i, 4, table_cell(str(r[5])))

            r_amt = float(r[3]) * r[4]
            r_amt -= r_amt * (r[5]/100)
            grid.setItem(i, 5, table_cell(f'{r_amt:,.2f}'))

        self.info = self.db.get_invoice_info(self.i)

        main_layout = QVBoxLayout()
        main_layout.addLayout(hbox(['Invoice Id:', str(self.i)]))
        main_layout.addLayout(hbox(['Date:', self.info[1].strftime('%d %B %Y')]))
        main_layout.addLayout(hbox(['Description:', self.info[0]]))

        amt = float(self.info[2])
        total_amt = amt + (amt * 0.18)

        main_layout.addWidget(grid)
        main_layout.addLayout(hbox(['Subtotal:', f'{amt:,.2f}']))
        main_layout.addLayout(hbox(['Tax rate:', '18%']))
        main_layout.addLayout(hbox(['Tax:', 'Rs ' + f'{total_amt*0.18:,.2f}']))
        main_layout.addLayout(hbox(['Total:', 'Rs ' + f'{total_amt:,.2f}']))
        main_layout.addWidget(icon_button('Save as PDF', 'pdf', self.save_pdf))
        main_layout.addWidget(icon_button('Go Back', 'back', self.close))
        widget = QWidget()
        widget.setLayout(main_layout)
        self.setCentralWidget(widget)

    def close(self):
        self.hide()

    def save_pdf(self):
        fileName, _ = QFileDialog.getSaveFileName(self,
            "Save File", f"invoice-{self.i}", "PDF Files(*.pdf)")
        if fileName == "":
            return

        try:
            print("Saving", fileName)
            f = PdfFile()
            f.generate(fileName, self.data, self.info)
            self.hide()
        except Exception:
            QMessageBox.critical(self, 'Error', 'Failed to save pdf')
```

list_invoice.py

```python
from PyQt6.QtWidgets import *
from PyQt6.QtCore import Qt
from decimal import Decimal

from pdf import PdfFile
from utils import table_cell, icon_button, hbox
from invoice_editor import InvoiceReceipt, InvoiceEditor

class ListInvoice(QMainWindow):
    def __init__(self, db):
        super().__init__()

        self.setWindowTitle("All Invoices")
        self.setFixedSize(800, 500)
        self.db = db

        self.table = QTableWidget(self)
        self.table.setColumnCount(7)
        for (i, w) in enumerate([50, 320, 130, 120, 40, 40, 40]):
            self.table.setColumnWidth(i, w)

        records = self.db.get_invoices()
        rows = []

        for r in records:
            rows.append({
                'Id': r[0],
                'Description': r[2],
                'Date': r[1],
                'Amount': r[3]
            })

        self.table.setHorizontalHeaderLabels([
            'Id', 'Description', 'Date', 'Amount(Rs)', '', '', ''
        ])

        self.table.setRowCount(len(rows))

        for (row, e) in enumerate(rows):
            self.table.setItem(row, 0, table_cell(str(e['Id'])))
            self.table.setItem(row, 1, table_cell(str(e['Description'])))
            self.table.setItem(row, 2, table_cell(e['Date'].strftime('%d/%m/%y')))
            self.table.setItem(row, 3, table_cell(f'{e['Amount']:,.2f}'))

            self.table.setCellWidget(row, 4, icon_button('', 'delete', self.delete))
            self.table.setCellWidget(row, 5, icon_button('', 'edit', self.edit))
            self.table.setCellWidget(row, 6, icon_button('', 'view', self.view))

        main_layout = QVBoxLayout()
        main_layout.addWidget(self.table)
        main_layout.addWidget(icon_button('Go Back', 'back', self.close))

        widget = QWidget()
        widget.setLayout(main_layout)
        self.setCentralWidget(widget)

    def view(self):
        self.receipt = InvoiceReceipt(self.db)
        current_row = self.table.currentRow()
        i = int(self.table.item(current_row, 0).text().strip())
        self.receipt.prepare(i)
        self.receipt.show()

    def delete(self):
        current_row = self.table.currentRow()
        i = int(self.table.item(current_row, 0).text().strip())

        button = QMessageBox.question(
            self,
            'Confirmation',
            'Are you sure that you want to delete the selected invoice?',
            QMessageBox.StandardButton.Yes |
            QMessageBox.StandardButton.No
        )
        if button == QMessageBox.StandardButton.Yes:
            self.table.removeRow(current_row)
            self.db.delete_invoice(i)

    def edit(self):
        current_row = self.table.currentRow()
        i = int(self.table.item(current_row, 0).text().strip())

        self.editor = InvoiceEditor(self.db, update=True, update_id=i)
        self.editor.show()
        self.hide()

    def close(self):
        self.hide()
```

main.py

```python
from db import DBConnection
from main_window import MainWindow
from PyQt6.QtWidgets import QApplication

try:
    db = DBConnection()
    app = QApplication([])

    font = app.font()
    font.setPointSize(14)
    app.setFont(font)

    window = MainWindow(db)
    window.show()

    app.exec()
except Exception as e:
    print("Unexpected Error Occurred")
    print(str(e))
```

main_window.py

```python
from PyQt6.QtWidgets import *
from PyQt6.QtGui import QPixmap
from PyQt6.QtCore import Qt
from invoice_editor import InvoiceEditor
from list_invoice import ListInvoice
from utils import icon_button

class MainWindow(QMainWindow):
    def __init__(self, db):
        super().__init__()

        self.db = db
        self.setWindowTitle("Invoice System")
        self.setFixedSize(900, 400)

        title = QLabel("Invoice Management System")
        font = title.font()
        font.setPointSize(40)
        font.setBold(True)
        title.setFont(font)
        title.setAlignment(Qt.AlignmentFlag.AlignCenter)

        layout = QVBoxLayout()
        layout.addWidget(title)
        layout.addWidget(icon_button('Create New Invoice', '', self.new_invoice))
        layout.addWidget(icon_button('Search By Id', '', self.modify_invoice))
        layout.addWidget(icon_button('List All Invoice', '', self.list_invoice))

        layout.setSpacing(15)

        widget = QWidget()
        widget.setLayout(layout)
        widget.setContentsMargins(50, 0, 50, 100)
        self.setCentralWidget(widget)

    def new_invoice(self):
        self.editor = InvoiceEditor(self.db)
        self.editor.show()

    def modify_invoice(self):
        inv_id, ok = QInputDialog.getInt(self, "Moidfy Invoice", "Enter the Invoice 
Id", min=1)
        if not ok:
            return

        if not self.db.invoice_exists(inv_id):
            QMessageBox.critical(self, 'Error', 'Invoice does not exist')
            return

        self.editor = InvoiceEditor(self.db, update=True, update_id=inv_id)
        self.editor.show()

    def list_invoice(self):
        self.viewer = ListInvoice(self.db)
        self.viewer.show()
```

pdf.py

```python
from fpdf import FPDF

class PdfFile(FPDF):
    def footer(self):
        self.set_y(-15)
        self.set_font("helvetica", style="I", size=8)
        self.cell(0, 10, f"Page {self.page_no()}/{{nb}}", align="C")

    def draw_banner(self, info):
        self.set_font("helvetica", size=40, style="B")
        self.ln(10)
        self.cell(80)
        self.cell(text="Invoice", align="C")

    def generate(self, file_name, data, info):
        self.add_page()
        self.image("icons/school.png", 10, 8, 33)

        self.draw_banner(info)
        self.ln(50)

        self.set_font("helvetica", size=12)
        self.cell(w=30, text="Date: ")
        self.cell(w=0, text=info[1].strftime('%d %B %Y'), align='L')
        self.ln(6)

        self.cell(w=30, text="Description: ")
        self.cell(w=0, text=info[0], align='L')
        self.ln(10)

        self.set_font("helvetica", size=14)
        self.set_y(90)
        with self.table(
            borders_layout="NO_HORIZONTAL_LINES",
            col_widths=(25, 80, 40, 20, 35, 50),
            text_align=("RIGHT", "LEFT", "RIGHT", "RIGHT", "RIGHT", "RIGHT"),
        ) as table:

            h_txt = [
                'S. No.', 'Item Name', 'Price (Rs)',
                'Qty', 'Disc (%)', 'Amount (Rs)'
            ]

            h_row = table.row()
            for s in h_txt:
                h_row.cell(s)

            for (i, r) in enumerate(data):
                row = table.row()
                row.cell(str(i+1))
                row.cell(r['Item Name'])
                row.cell(f'{ r['Price']:,.2f}')
                row.cell(str(r['Qty']))
                row.cell(f'{ r['Discount']}%')
                row.cell(f'{r['Amount']:,.2f}')
        self.ln(10)

        amt = float(info[2])
        self.bottom_text(f"{"Subtotal":11}: Rs {amt:10,.2f}")
        self.bottom_text(f"{"Tax Rate":10}:   {"18%":10}")
        self.bottom_text(f"{"Tax":14}: Rs {amt*0.18:10,.2f}")

        self.ln(5)
        self.set_font("helvetica", size=20, style="B")
        self.bottom_text(f"Total: Rs {amt+amt*0.18:,.2f}", x=-95)

        self.output(file_name)

    def bottom_text(self, txt, x=-90):
        self.set_x(x)
        self.cell(text=txt)
        self.ln(7)
```

1. To Create a students Table

```
CREATE TABLE students (     adm_no INT
 NOT NULL AUTO_INCREMENT,
    student_name VARCHAR(255) NOT NULL,
    mother_name VARCHAR(255) NOT NULL,
    father_name VARCHAR(255) NOT NULL,
    class INT NOT NULL,
    section CHAR(1) NOT NULL,
    PRIMARY KEY(adm_no)
);
```

2. To insert a row in students table

```
INSERT INTO students(
    adm_no, student_name, mother_name, father_name,
    class, section
) VALUES (
    13726, 'Ashish Anand', 'Kamini Kumari', 'Rohan Kumar',
    11, 'A'
);
```

3. To get name of student with admission number 13726

```
SELECT student_name FROM students WHERE adm_no = 13726;
```

4. To change class of student with admission number 13726

```
UPDATE students SET class = 12 WHERE adm_no = 13726;
```

5. To remove student whose admission number is 13726

```
DELETE FROM students WHERE adm_no = 13726;
```

1. **Program to print between 1 and 100 pythagorean triplet**

```
for i in  range(1, 101):
   for j in range(1, 101):
        for k in range(1, 101):
            if (i*i + j*j) == (k*k):
                print(i, j, k)
```

2. **Program to input numbers and calculate their sum**

```
try:
    num = int(input("Enter number of inputs: "))
    nums = []

    for i in range(num):
        nums.append(int(input(f"Enter {i+1} number: ")))

    total = sum(nums)
    print(f"Total sum is: {total}")
except Exception as err:
    print(str(err))
```

3. **Dice : Program to print a random number between 1 and 6**

```
import random

num = random.randrange(1, 7) #upper bound is not inclusive
print(num)
```

4. **Program that accepts radius and print info about circle**

```
import math
try:
    radius = float(input("Enter the radius: "))

    area = math.pi * (radius)**2
    circumfrence = 2 * math.pi * radius

    print(f"Area: {area:.2f}")
    print(f"Circumfrence: {circumfrence:.2f}")

except Exception as err:
    print(str(err))
```

5. **Program to calculate accept marks and calculate percentage**

```
def input_marks(subject):
    marks = int(input(f"Enter {subject} marks: "))
    if marks < 0 or marks > 100:
        raise Exception(f"Invalid marks given in {subject}")
    return marks

try:
    maths = input_marks('maths')
```

```python
    eng = input_marks('english')
    phy = input_marks('physics')
    chem = input_marks('chemistry')
    cs = input_marks('computer science')

    total = maths + eng + phy + chem + cs
    percentage = (total / 500) * 100
    print(f"Percentage is: {percentage}")
except Exception as err:
    print(str(err))
```

6. Program to check if the entered word is palindrome

```python
txt = input("Enter a word: ").strip()

rev = ""
for i in range(len(txt)-1, -1, -1): #Start from last char
    rev += txt[i]

if rev == txt:
    print("Word is palindrome")
else:
    print("Word is not palindrome")
```

7. Program to check if a number is prime number or not

```python
try:
    num = int(input("Enter a number: "))

    for i in range(2, num):
        if num % i == 0:
            print("Number is not prime")
            quit()
    print("Number is prime")
except Exception as err:
    print(str(err))
```

8: Program to solve quadratic equation

```python
import math

try:
    print("Enter details of quadratic of form ax^2 + bx + c")

    a = float(input("Enter value of a: "))
    b = float(input("Enter value of b: "))
    c = float(input("Enter value of c: "))

    d = (b**2 - 4*a*c)
    if d < 0:
        print("No roots")
    elif d == 0:
        r = -b/(2 * a)
        print(f"Only one root: {r}")
    else:
        d = math.sqrt(d)
        r1 = ((-b) - d) / (2 * a)
        r2 = ((-b) + d) / (2 * a)
        print(f"Roots are {r1} and {r2}")
except Exception as err:
    print(str(err))
```

9. Program to find sum of given series
S = (1) + (1+2) + (1+2+3) + ... + (1+2+3+...+n)

```
try:
    n = int(input("Enter value of n:"))
    assert(n > 0)

    s = 0
    for i in range(1,n+1):
        for j in range(1, i+1):
            s += j
    print(f"Sum is: {s}")
except Exception as err:
    print(str(err))
```

10. Program to perform linear search on list of 10 numbers

```
try:
    nums = []
    for i in range(10):
        nums.append(int(input(f"Enter {i+1} number: ")))

    target = int(input("Enter number to search: "))

    for (i, j) in enumerate(nums):
        if target == j:
            print(f"{target} is present at index {i}")
            quit()
    print(f"Number not found in list")

except Exception as err:
    print(str(err))
```

11. Program that reads a line and counts number of uppercase lowercase letters and digits

```
txt = input("Enter a line: ").strip()

upper , lower, dig = 0, 0, 0
for c in txt:
    if c.isupper():
        upper += 1
    elif c.islower():
        lower += 1
    elif c.isdigit():
        dig += 1

print(f"Uppercase letters: {upper}")
print(f"Lowercase letters: {lower}")
print(f"Digits: {dig}")
```

12. Program to print the following pattern

```
    1
   1 1
  1   1
 1     1
111111111
```

```python
for i in range(5):
    for j in range(1, 10):
        if i+j == 5 or j-i == 5 or i == 4:
            print('1', end='')
        else:
            print(' ', end='')
    print()
```

13. Program to print the given pattern

```
1
1 3
1 3 5
1 3 5 7
1 3 5 7 9
```

```python
def nth_odd(n):
    return (2 * n + 1)

for i in range(1, 6):
    for j in range(i):
        print(nth_odd(j), end=' ')
    print()
```

14. Program to calculate factorial using recursion

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

try:
    n = int(input("Enter a number: "))
    assert(n >= 0)
    f = factorial(n)
    print(f"Factorial is {f}")

except Exception as err:
    print(str(err))
```

15. Program to perform insertion sort on list of 10 numbers

```python
try:
    nums = []
    for i in range(10):
        nums.append(int(input(f"Enter {i+1} number: ")))

    nums2 = nums.copy()
    for i in range(1, len(nums2)):
        key = nums2[i]
        j = i-1
        while j >= 0 and key < nums2[j]:
            nums2[j+1] = nums2[j]
            j -= 1
        nums2[j+1] = key

    print("Sorted Array: ")
    print(nums2)

except Exception as err:
    print(str(err))
```

1. Program to print names of all students present in a given class

```python
from mysql.connector import connect

try:
    cnx = connect(user="admin",password="admin@12345",,database="defaultdb")

    c = int(input('Enter class: '))
    assert(c > 0 & c <= 12)

    with cnx.cursor() as cur:
        cur.execute(f'SELECT student_name FROM students WHERE class = {c}')
        res = cur.fetchall()

        print("All students in given class: ")
        for r in res:
            print(r[0])

except Exception as err:
    print(str(err))
```

2. Program to change section of a certain student

```python
from mysql.connector import connect

try:
    cnx = connect(user="admin",password="admin@12345",,database="defaultdb")

    adm = int(print("Enter adm number of student: "))
    sec = print("Enter new section: ").strip()[0] # To get only one character

    with cnx.cursor() as cur:
        cursor.execute(f'UPDATE students SET section = {sec} WHERE adm_no = {adm}'
)
        cnx.commit()

except Exception as err:
    print(str(err))
```

3. Program to print details of a particular student

```python
from mysql.connector import connect

try:
    cnx = connect(user="admin",password="admin@12345",,database="defaultdb")

    adm = int(input("Enter admission number: "))

    with cnx.cursor() as cur:
        query = (
            'SELECT student_name, mother_name, father_name '
            f'class, section FROM students WHERE adm_no = {adm}'
        )
        cur.execute(query)
        res = cur.fetchall()[0]

        print(f'Student Name: {res[0]}')
        print(f'Mother Name: {res[1]}')
        print(f'Father Name: {res[2]}')
        print(f'Class: {res[3]}')
        print(f'Section: {res[4]}')

except Exception as err:
    print(str(err))
```

4. Program to remove a specific student

```
from mysql.connector import connect

try:
    cnx = connect(user="admin",password="admin@12345",,database="defaultdb")

    adm = int(input("Enter admission number: "))

    with cnx.cursor() as cur:
        cur.execute(f'DELETE FROM students WHERE adm_no = {adm}')
        cnx.commit()

except Exception as err:
    print(str(err))
```

5. Program to create a new student

```
from mysql .connector import connect
try:
    cnx = connect(user="admin",password="admin@12345",,database="defaultdb")

    name = input('Enter name: ').strip()
    mother_name = input('Enter mother name: ').strip()
    father_name = input('Enter father name: ').strip()
    c = int(input('Enter Class: '))
    s = input('Enter Section:').strip()[0]

    assert(c > 0 && c <= 12)

    with cnx.cursor() as cur:
        query = (
            'INSERT INTO students(student_name, mother_name, father_name,'
            f' class, section) VALUES ({name}, {mother_name}, {father_name}, '
            f' {c}, {s})'
        )
        cur.execute(query)
        cnx.commit()

except Exception as err:
    print(str(err))
```