

```
import os
import sqlite3
import pandas as pd

class HardwareStoreDB:
    PATH_DB      = os.path.join('data', 'HardwareStore.sqlite')
    PATH_PROD   = os.path.join('data', 'products.csv')
    PATH_STAT   = os.path.join('data', 'states.csv')
    PATH_ZIP    = os.path.join('data', 'zips.csv')
    PATH_SALE   = os.path.join('data', 'sales_sample.csv')

    def __init__(self,
                 create: bool = False
                 ):
        self._connected = False
        self._check_exists(create)

        if not self._existed:
            self._create_tables()
            self._load_data()
        return

    def _create_tables(self) -> None:
        sql = """
            CREATE TABLE tState (
                state_id TEXT PRIMARY KEY,
                state TEXT NOT NULL
            )
            ;"""
        self.run_action(sql)

        sql = """
            CREATE TABLE tZip (
                zip TEXT PRIMARY KEY,
                city TEXT NOT NULL,
                state_id TEXT NOT NULL,
                FOREIGN KEY (state_id)
                    REFERENCES tState (state_id)
                ON UPDATE CASCADE
                ON DELETE CASCADE
            )
            ;"""
        self.run_action(sql)

        sql = """
            CREATE TABLE tProd (
                prod_id INTEGER PRIMARY KEY,
                prod_desc TEXT NOT NULL,
                unit_price INTEGER NOT NULL
            )
            ;"""
        self.run_action(sql)

        sql = """
            CREATE TABLE tCust (
                cust_id INTEGER PRIMARY KEY,
                first TEXT NOT NULL,
                last TEXT NOT NULL,
                addr TEXT NOT NULL,
                zip TEXT NOT NULL,
                FOREIGN KEY (zip)
                    REFERENCES tZip (zip)
                ON UPDATE CASCADE
                ON DELETE CASCADE
            )
            ;"""
        self.run_action(sql)

        sql = """
            CREATE TABLE tInvoice (
                invoice_id INTEGER PRIMARY KEY,
                cust_id INTEGER NOT NULL,
                year INTEGER NOT NULL,
                month INTEGER NOT NULL,
                day INTEGER NOT NULL,
                time TEXT NOT NULL,
                FOREIGN KEY (cust_id)
                    REFERENCES tCust (cust_id)
                ON UPDATE CASCADE
                ON DELETE CASCADE
            )
            ;"""
        self.run_action(sql)


```

```

sql = """
CREATE TABLE tInvoiceDetail (
    invoice_id INTEGER NOT NULL,
    prod_id INTEGER NOT NULL,
    qty INTEGER NOT NULL,
    FOREIGN KEY (invoice_id)
        REFERENCES tInvoice (invoice_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (prod_id)
        REFERENCES tProd (prod_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    PRIMARY KEY (invoice_id, prod_id)
)
;"""
self.run_action(sql)
return

def _load_data(self) -> None:
    tState = pd.read_csv(self.PATH_STAT)
    tState.columns = ['state_id', 'state']
    sql = """
        INSERT INTO tState (state_id, state)
        VALUES (:state_id, :state)
    ;"""
    for row in tState.to_dict(orient='records'):
        self.run_action(sql, row, keep_open=True)

    tZip = pd.read_csv(self.PATH_ZIP)
    tZip.columns = ['zip', 'city', 'state_id']
    tZip['zip'] = tZip['zip'].astype(str)
    sql = """
        INSERT INTO tZip (zip, city, state_id)
        VALUES (:zip, :city, :state_id)
    ;"""
    for row in tZip.to_dict(orient='records'):
        self.run_action(sql, row, keep_open=True)

    tProd = pd.read_csv(self.PATH_PROD)
    tProd.columns = ['prod_id', 'prod_desc', 'unit_price']
    sql = """
        INSERT INTO tProd (prod_id, prod_desc, unit_price)
        VALUES (:prod_id, :prod_desc, :unit_price)
    ;"""
    for row in tProd.to_dict(orient='records'):
        self.run_action(sql, row, keep_open=True)

    tSale = pd.read_csv(self.PATH_SALE)
    tSale['zip'] = tSale['zip'].astype(str)

    max_cust_df = self.run_query(
        "SELECT COALESCE(MAX(cust_id), 0) AS max_id FROM tCust;")
    start_cust_id = int(max_cust_df['max_id'].iloc[0])

    cust_cols = ['first', 'last', 'addr', 'zip']
    tCust = (
        tSale[cust_cols]
            .drop_duplicates()
            .reset_index(drop=True)
    )
    tCust['cust_id'] = tCust.index + start_cust_id + 1

    sql = """
        INSERT INTO tCust (cust_id, first, last, addr, zip)
        VALUES (:cust_id, :first, :last, :addr, :zip)
    ;"""
    for row in tCust.to_dict(orient='records'):
        self.run_action(sql, row, keep_open=True)

    tSale = tSale.merge(tCust, on=customer_cols, how='left')

    dt = pd.to_datetime(tSale['date'])
    tSale['year'] = dt.dt.year.astype(int)
    tSale['month'] = dt.dt.month.astype(int)
    tSale['day'] = dt.dt.day.astype(int)
    tSale['time'] = dt.dt.strftime('%H:%M:%S')

    max_inv_df = self.run_query(
        "SELECT COALESCE(MAX(invoice_id), 0) AS max_id FROM tInvoice;")
    start_invoice_id = int(max_inv_df['max_id'].iloc[0])

    inv_cols = ['cust_id', 'year', 'month', 'day', 'time']
    tInvoice = (
        tSale[inv_cols]
            .drop_duplicates()
            .reset_index(drop=True)
    )
    tInvoice['invoice_id'] = tInvoice.index + start_invoice_id + 1

```

```

sql = """
    INSERT INTO tInvoice (invoice_id, cust_id, year, month, day, time)
    VALUES (:invoice_id, :cust_id, :year, :month, :day, :time)
"""
for row in tInvoice.to_dict(orient='records'):
    self.run_action(sql, row, keep_open=True)

tSale = tSale.merge(
    tInvoice,
    on=inv_cols,
    how='left')

tDetail = tSale[['invoice_id', 'prod_id', 'qty']].copy()

sql = """
    INSERT INTO tInvoiceDetail (invoice_id, prod_id, qty)
    VALUES (:invoice_id, :prod_id, :qty)
"""
for row in tDetail.to_dict(orient='records'):
    self.run_action(sql, row, keep_open=True)

self._commit_and_close()
return

def run_query(self,
              sql: str,
              params: dict = None,
              keep_open: bool = False
              ) -> pd.DataFrame:
    self._connect()
    try:
        results = pd.read_sql(sql, self._conn, params=params)
    except Exception as e:
        raise type(e)(f'sql: {sql}\nparams: {params}')
    finally:
        if not keep_open:
            self._close()
    return results

def run_action(self,
               sql: str,
               params: dict = None,
               commit: bool = False,
               keep_open: bool = False
               ) -> int:

    self._connect()
    try:
        if params is None:
            self._curs.execute(sql)
        else:
            self._curs.execute(sql, params)
        if commit:
            self._conn.commit()
    except Exception as e:
        self._conn.rollback()
        self._close()
        raise type(e)(f'sql: {sql}\nparams: {params}') from e

    if not keep_open:
        self._close()
    return self._curs.lastrowid

def _connect(self, foreign_keys: bool = True) -> None:
    if not self._connected:
        self._conn = sqlite3.connect(self.PATH_DB)
        self._curs = self._conn.cursor()
        if foreign_keys:
            self._curs.execute("PRAGMA foreign_keys=ON;")
        self._connected = True
    return

def _commit_and_close(self) -> None:
    self._conn.commit()
    self._close()
    return

def _close(self) -> None:
    self._conn.close()
    self._connected = False
    return

def _check_exists(self, create: bool) -> None:
    self._existed = True

```

```
path_parts = self.PATH_DB.split(os.sep)

n = len(path_parts)
for i in range(n):
    part = os.sep.join(path_parts[:i+1])
    if not os.path.exists(part):
        self._existed = False
        if not create:
            raise FileNotFoundError(f'{part} does not exist.')
        else:
            if i == n-1:
                print('Creating DB')
                self._connect()
                self._close()
            else:
                os.mkdir(part)

return
```