项目设计<*13*><42204341-张悦鑫>_<42212263-刘嘉译>

1.时间进度

时间	进度
15周	数据查询、设计数据库、编写查询分析程序
16周	设计可视化界面
17周	撰写设计报告-

2.组员分工

组员	分工
张悦鑫	设计数据库、编写查询分析程序、使用Qt设计可视化界面、撰写设计报告
刘嘉译	查询数据、使用ODBC连接数据库、撰写设计报告

3.数据库设计

从中经数据库中可以得到各个省份、城市的年度数据（社会消费品零售额、居民人均可支配收入、常住人口、GDP），下载后如图1所示。下载下来的数据导入数据库不怎么方便，需要对格式进行调整。由于查询涉及到了时间、区域、某某数值，所以将格式也调整为时间、区域、某某数值，调整后如图2所示。

数据来源：中经数据（CEIdata）		
序列ID	652552	652569
指标	消费品零售	消费品零售
地区	北京	天津
频度	年	年
单位	亿元	亿元
2010	7272.97	2860.2
2011	8334.79	3395.06
2012	9440.19	3921.43

数据来源：中经数据（CEIdata）		
序列ID	652552	652569
指标	消费品零售	消费品零售
地区	北京	天津
频度	年	年
单位	亿元	亿元
2016	13134.94	4188.1
2017	13933.74	4210.4
2018	14422.3	4231.2

数据来源：中经数据（CEIdata）		
序列ID	2877958	2877975
指标	人均可支配	人均可支配
地区	北京	天津
频度	年	年
单位	元/人	元/人
2020	69433.54	43854.09
2021	75002.2	47449.4
2022	77414.55	48976.12

数据来源：中经数据（CEIdata）		
序列ID	2003640	2003657
指标	常住人口数	常住人口数
地区	北京	天津
频度	年	年
单位	万人	万人
2014	2171.1	1429
2015	2188.3	1439
2016	2195.4	1443

图1 中经数据库下载的数据

1	北京	2010	7272.97
2	北京	2011	8334.79
3	北京	2012	9440.19
4	北京	2013	10382.47
5	北京	2014	11354.04
6	北京	2015	12271.87
7	北京	2016	13134.94
8	北京	2017	13933.74

1	北京	2016	13134.94	北京
2	北京	2017	13933.74	北京
3	北京	2018	14422.3	北京
4	北京	2019	15063.65	北京
5	北京	2020	13716.4	北京
6	北京	2021	14867.74	北京
7	北京	2022	13794.25	北京
8	北京	2023	14462.66	北京

1	北京	2014	44488.57	27554.91	1452.2	7000.9	8480.56
2	北京	2015	48457.99	30241	1421.26	7498.91	9297.4
3	北京	2016	52530.38	33114.15	1396.42	8229.63	9790.18
4	北京	2017	57229.83	35216.63	1408.26	9305.94	11299
5	北京	2018	62361.22	37686.8	1201.61	10611.76	12861.05
6	北京	2019	67755.91	41214.14	1200.59	11257.04	14084.15
7	天津	2014	28832.29	17162.98	2875.62	2781.69	6012
8	天津	2015	31291.36	19256.22	2906.08	2928	6201.06

1	北京	2014	2171.1
2	北京	2015	2188.3
3	北京	2016	2195.4
4	北京	2017	2194.4
5	北京	2018	2191.7
6	北京	2019	2190.1
7	北京	2020	2189.31
8	北京	2021	2188.6

图2 调整完格式的数据

进而数据库中表格的形式也随之确定了，如图3所示。

列名	数据类型
province	varchar(50)
year	int
sales_amount	decimal(10, 2)

列名	数据类型
city	varchar(50)
year	int
sales_amount	decimal(10, 2)
province	varchar(50)

列名	数据类型
province	varchar(50)
year	int
disposableIncome	float
disposableIncome_wa...	float
disposableIncome_op...	float
disposableIncome_net...	float
disposableIncome_tra...	float

列名	数据类型
province	varchar(50)
year	int
population	decimal(10, 2)

图3 数据库表格结构

4.查询编写

4.1 社会消费变化及区域差异

（1）查询在指定时间段指定区域的社会消费品零售总额。

代码：

```
CREATE PROCEDURE GetSalesByProvince
    @province_name NVARCHAR(255),
    @start_year INT,
    @end_year INT
AS
BEGIN
    SELECT province, year, SUM(sales_amount) AS total_sales
    FROM salesAmount
    WHERE province = @province_name
      AND year BETWEEN @start_year AND @end_year
    GROUP BY province, year
    ORDER BY year;
END;
```

```
EXEC GetSalesByProvince @province_name = '北京', @start_year = 2015, @end_year =
2020;


EXEC GetSalesByProvince '北京', 2015, 2020;
```

分析：

1.输入参数：

- `@province_name`：省份名称，用于指定查询的区域。
- `@start_year` 和 `@end_year`：查询的起始年份和结束年份，用于限定时间范围。

2.查询逻辑：

- 从表 salesAmount中筛选出满足以下条件的记录：
  - `province` 等于 `@province_name`。
  - `year` 在 `@start_year` 和 `@end_year` 之间（包含边界）。
- 对筛选出的数据按 `province` 和 `year` 进行分组，并计算每年的总消费额。
- 按照 `year` 升序排列结果。

3.输出结果：

- 返回包含 `province`、`year` 和 `total_sales` 的结果集，展示指定省份在指定年份范围内的年度社会消费品零售总额。

（2-1）查询指定时间段社会消费品零售总额最高和最低区域。

代码：

```
CREATE PROCEDURE GetMaxMinSalesByYearRange
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    -- 使用 CTE 和查询逻辑
    WITH TotalSales AS (
        SELECT
            province,
            SUM(sales_amount) AS total_sales
        FROM salesAmount
        WHERE year BETWEEN @startYear AND @endYear -- 使用参数指定时间段
        GROUP BY province
    ),
    MaxMin AS (
        SELECT
            MAX(total_sales) AS max_sales,
            MIN(total_sales) AS min_sales
        FROM TotalSales
    )
    SELECT
        ts.province,
```

```
        ts.total_sales,
        CASE
            WHEN ts.total_sales = mm.max_sales THEN '最高'
            WHEN ts.total_sales = mm.min_sales THEN '最低'
            ELSE ''
        END AS mark
    FROM TotalSales ts
    CROSS JOIN MaxMin mm
    ORDER BY ts.total_sales DESC;
END;

-- 调用存储过程，指定年份范围
EXEC GetMaxMinSalesByYearRange @startYear = 2010, @endYear = 2023;
```

分析：

1.输入参数：

- `@startYear`：起始年份，限制时间段的下界。
- `@endYear`：结束年份，限制时间段的上界。

2.主要步骤：

- **第一步：构建 CTE（公共表表达式）TotalSales**
  - 汇总指定年份范围内各省份的总销售额。
  - 通过 `SUM(sales_amount)` 按 `province` 进行聚合，计算各省份的社会消费品零售总额。
  - 数据示例（假设查询 2010-2015 年）：

```
province     total_sales
北京          500000.00
上海          400000.00
天津          300000.00
```

- **第二步：构建 CTE MaxMin**
  - 计算出 TotalSales 的 `max_sales`（最大销售额）和 `min_sales`（最小销售额）。
  - 数据示例：

```
max_sales    min_sales
500000.00    300000.00
```

- **第三步：通过 CROSS JOIN 比较并标记**
  - 将每个省份的总销售额与最大值和最小值进行比较：
    - 如果等于 `max_sales`，标记为“最高”。
    - 如果等于 `min_sales`，标记为“最低”。
  - 最终结果：

```
province     total_sales    mark
北京          500000.00      最高
上海          400000.00
天津          300000.00      最低
```

3.排序和输出：

- 按 `total_sales` 降序排列结果，便于快速查看。

4.输出结果：

- 返回的列包括：
  - `province`：省份名称。
  - `total_sales`：该省份在指定时间段内的总社会消费品零售总额。
  - `mark`：标记“最高”或“最低”，未达到最大或最小值的省份为空。

（2-2）分析近三年最高区域和最低区域是否有变化。

代码：

```
CREATE PROCEDURE AnalyzeMaxMinSales
AS
BEGIN
    SET NOCOUNT ON;

    -- 使用 CTE 按年计算总额
    WITH YearlyTotalSales AS (
        SELECT
            year,
            province,
            SUM(sales_amount) AS total_sales
        FROM salesAmount
        WHERE year BETWEEN 2021 AND 2023 -- 指定近三年的范围
        GROUP BY year, province
    ),
    MaxMinByYear AS (
        SELECT
            year,
            MAX(total_sales) AS max_sales,
            MIN(total_sales) AS min_sales
        FROM YearlyTotalSales
        GROUP BY year
    )
    SELECT
        yts.year,
        yts.province,
        yts.total_sales,
        CASE
            WHEN yts.total_sales = mmy.max_sales THEN '最高'
            WHEN yts.total_sales = mmy.min_sales THEN '最低'
            ELSE ''
        END AS mark
    FROM YearlyTotalSales yts
    INNER JOIN MaxMinByYear mmy
        ON yts.year = mmy.year
    WHERE yts.total_sales = mmy.max_sales OR yts.total_sales = mmy.min_sales --
仅显示最高和最低区域
    ORDER BY yts.year, mark DESC;
END;
```

```
EXEC AnalyzeMaxMinSales
```

分析：

1. 数据来源和逻辑清晰

- 使用了两层 CTE：
  - **YearlyTotalSales**: 按年和省份计算总的消费品销售额（`total_sales`）。
  - **MaxMinByYear**: 按年获取每年的最高总额（`max_sales`）和最低总额（`min_sales`）。
- 在最终查询中，通过 `INNER JOIN` 连接 **YearlyTotalSales** 和 **MaxMinByYear**，筛选出最高和最低的区域并标记，且结果按年份排序。这种设计清晰且易于理解。

2. 参数范围的固定

- 范围固定: 存储过程中直接写死了年份范围 2021 到 2023，这与题目"分析近三年"匹配，但可能会影响灵活性。因为时间过了 2023 年后，代码需手动修改年份范围。

3. 结果过滤

- **筛选最高和最低区域**: 使用 `WHERE yts.total_sales = mmy.max_sales OR yts.total_sales = mmy.min_sales` 只保留最高和最低区域，这一点精准符合题意。

4. 输出标记

- 通过 `CASE` 对最高区域标记为 **'最高'**，最低区域标记为 **'最低'**，标记逻辑非常清晰，且在结果中能很好地体现出显著性。

（3）分析在近三年(20220~2022)中，哪些区域的社会消费品零售总额在2020年同比下降，2021年同比增加，2022年同比下降。计算这些省份在所有省份和直辖市的占比。

代码：

```
CREATE PROCEDURE AnalyzeGrowthRates
AS
BEGIN
    SET NOCOUNT ON;

    -- 计算每年的增长率
    WITH SalesWithGrowth AS (
        SELECT
            province,
            year,
            sales_amount,
            LAG(sales_amount) OVER (PARTITION BY province ORDER BY year) AS
prev_year_sales
        FROM salesAmount
        WHERE year BETWEEN 2019 AND 2022
    ),
    GrowthRates AS (
        SELECT
            province,
            year,
            sales_amount,
            prev_year_sales,
            CASE
```

```
                WHEN prev_year_sales IS NOT NULL THEN (sales_amount -
prev_year_sales) / prev_year_sales * 100
                ELSE NULL
            END AS growth_rate
        FROM SalesWithGrowth
    ),
    FilteredProvinces AS (
        SELECT
            province
        FROM GrowthRates
        WHERE
            (year = 2020 AND growth_rate < 0) -- 2020年同比下降
            OR (year = 2021 AND growth_rate > 0) -- 2021年同比增加
            OR (year = 2022 AND growth_rate < 0) -- 2022年同比下降
        GROUP BY province
        HAVING COUNT(CASE WHEN year = 2020 AND growth_rate < 0 THEN 1 END) > 0
               AND COUNT(CASE WHEN year = 2021 AND growth_rate > 0 THEN 1 END) >
0
               AND COUNT(CASE WHEN year = 2022 AND growth_rate < 0 THEN 1 END) >
0
    ),
    TotalSales AS (
        SELECT
            province,
            SUM(sales_amount) AS total_sales
        FROM salesAmount
        WHERE year BETWEEN 2020 AND 2022
        GROUP BY province
    ),
    TotalAllProvinces AS (
        SELECT
            SUM(sales_amount) AS total_all_sales
        FROM salesAmount
        WHERE year BETWEEN 2020 AND 2022
    )
    SELECT
        ts.province,
        ts.total_sales,
        (ts.total_sales * 1.0 / tap.total_all_sales) * 100 AS percentage
    FROM TotalSales ts
    JOIN FilteredProvinces fp ON ts.province = fp.province
    CROSS JOIN TotalAllProvinces tap
    ORDER BY percentage DESC;
END;

EXEC AnalyzeGrowthRates;
```

分析：

1.计算每年的增长率：

- 使用 `LAG` 函数计算每个省份在近三年（2019-2022）的社会消费品零售总额的同比增长率。
- 增长率计算公式为：

$$\text{增长率} = \frac{\text{本年销售额} - \text{上年销售额}}{\text{上年销售额}} \times 100$$

2.筛选符合条件的省份：

- 根据题目要求，筛选出满足以下条件的省份：
  - **2020年同比下降**（增长率 < 0）
  - **2021年同比增加**（增长率 > 0）
  - **2022年同比下降**（增长率 < 0）
- 使用 `HAVING` 和 `COUNT` 聚合函数确保每个省份同时满足这三个条件。

3.计算这些省份的社会消费品零售总额总和：

- 筛选出的省份，在 2020-2022 年期间的社会消费品零售总额进行总和统计。

4.计算筛选省份总额在所有省份中的占比：

- 使用 `CROSS JOIN` 获取所有省份 2020-2022 年的零售总额总和。
- 计算筛选省份的总额占所有省份总额的百分比。

5.排序输出：

- 按照占比（`percentage`）从高到低排序输出结果。

（4）分析省会城市的消费对其所在省份的贡献，即查询在指定时间段省会城市社会消费品零售总额在其所在的省份的占比。

代码：

```
CREATE PROCEDURE AnalyzeCapitalCityContribution
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH ProvinceTotal AS (
        -- 计算每个省份在指定时间段的总额
        SELECT
            province,
            SUM(sales_amount) AS province_total
        FROM salesAmount
        WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY province
    ),
    CapitalCityTotal AS (
        -- 计算每个省会城市在指定时间段的总额
        SELECT
            city,
            province,
            SUM(sales_amount) AS city_total
        FROM capital_salesAmount -- 假设表中包含省会城市数据
        WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY city, province
```

```
    )
    SELECT
        cct.city AS capital_city,
        cct.province,
        cct.city_total,
        pt.province_total,
        (cct.city_total * 1.0 / pt.province_total) * 100 AS
contribution_percentage
    FROM CapitalCityTotal cct
    JOIN ProvinceTotal pt
        ON cct.province = pt.province
    ORDER BY contribution_percentage DESC;
END;


EXEC AnalyzeCapitalCityContribution @startYear = 2020, @endYear = 2022;
```

分析：

1.输入参数：

- `@startYear`：指定分析的起始年份。
- `@endYear`：指定分析的结束年份。

2.主要步骤：

- **Step 1:** 使用 `ProvinceTotal` CTE 计算每个省份在指定时间段内的社会消费品零售总额（`province_total`）。
  - 数据来源：`salesAmount` 表，按 `province` 分组。
  - 条件：仅包含 `year` 在指定时间范围内的数据。
- **Step 2:** 使用 `CapitalCityTotal` CTE 计算每个省会城市在指定时间段的社会消费品零售总额（`city_total`）。
  - 数据来源：假设存在 `capital_salesAmount` 表，记录省会城市的消费额，按 `city` 和 `province` 分组。
  - 条件：同样仅包含 `year` 在指定时间范围内的数据。
- **Step 3:** 将 `CapitalCityTotal` 和 `ProvinceTotal` 通过 `province` 进行关联。
  - 目标：计算每个省会城市的消费额占其所在省份总额的百分比 `contribution_percentage`。

3.输出：

- `capital_city`（省会城市名）
- `province`（所在省份名）
- `city_total`（省会城市消费总额）
- `province_total`（省份消费总额）
- `contribution_percentage`（省会城市消费额占比，按百分比表示，降序排列）

4.排序规则：

- 结果按 `contribution_percentage` 降序排列，优先显示贡献率高的省会城市。

（5）分析比较指定时间段东部、西部、中部、东北部地区的社会消费品零售总额，并进行排序。

代码：

```
CREATE PROCEDURE AnalyzeRegionalSales
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH RegionSales AS (
        SELECT
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知区域'
            END AS region,
            SUM(sales_amount) AS total_sales
        FROM salesAmount
        WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知区域'
            END
    )
    SELECT
        region,
        total_sales
    FROM RegionSales
    ORDER BY total_sales DESC; -- 按总额降序排序
END;

EXEC AnalyzeRegionalSales @startYear = 2020, @endYear = 2022;
```

分析：

1.功能目标：

- 依据 `salesAmount` 表中存储的销售数据，按照省份归属划分为四个区域（东部、中部、西部、东北）。
- 动态过滤指定的时间段（`@startYear` 到 `@endYear`）。
- 统计各区域的社会消费品零售总额（`SUM(sales_amount)`），并按照销售额从高到低排序。

2.实现亮点：

- **使用动态时间过滤：**
  - 利用输入参数 `@startYear` 和 `@endYear`，实现了灵活筛选时间段的能力，适用于多种分析场景。
- **区域划分逻辑清晰：**
  - 通过 `CASE` 语句对省份进行分类，按区域（东部、中部、西部、东北）进行划分，逻辑简单且直观。
- **聚合与排序：**
  - 使用了 `SUM(sales_amount)` 对指定区域的销售额进行聚合，`ORDER BY total_sales DESC` 实现了结果的降序排列，便于对比。
- **采用 CTE（公用表表达式）：**
  - 利用 `WITH` 和 CTE 的方式，将区域划分和销售额计算的逻辑清晰地分离，提高了代码可读性和维护性。

4.2 居民收入变化及区域差异

（1）查询在分析在指定时间段指定区域居民收入环比增长率。

代码：

```
CREATE PROCEDURE AnalyzeIncomeGrowthRate
    @startYear INT, -- 起始年份
    @endYear INT,   -- 结束年份
    @province NVARCHAR(50) -- 指定区域
AS
BEGIN
    SET NOCOUNT ON;

    WITH IncomeWithLag AS (
        -- 计算指定区域在指定时间段的收入及上一年的收入
        SELECT
            province,
            year,
            disposableincome, -- 当前居民收入
            LAG(disposableincome) OVER (PARTITION BY province ORDER BY year) AS
prev_year_income -- 上一年居民收入
        FROM disposableincome
        WHERE year BETWEEN @startYear AND @endYear
          AND province = @province -- 指定区域
    )
    SELECT
        province,
        year,
        disposableincome AS current_income,
        prev_year_income AS previous_income,
        CASE
            WHEN prev_year_income IS NOT NULL THEN
                ((disposableincome - prev_year_income) / prev_year_income) * 100
            ELSE
                NULL
        END AS growth_rate
    FROM IncomeWithLag
    ORDER BY year;
```

```
END;

EXEC AnalyzeIncomeGrowthRate @startYear = 2020, @endYear = 2022, @province = '北京';
```

分析：

1.功能目标：

- 查询指定区域（`@province`）在指定时间段（`@startYear` 到 `@endYear`）的居民可支配收（`disposableincome`）。
- 计算环比增长率：根据当前年份的居民收入与上一年的收入进行比较，得出增长率。
- 输出按年份排序的结果。

2.实现亮点：

- **利用窗口函数 `LAG` 计算上一年收入：**
  - 使用 `LAG(disposableincome)` 提取上一年的收入值，不需要自连接表，避免了复杂的子查询，提高了效率和代码简洁性。
- **支持动态筛选时间段和区域：**
  - 通过输入参数 `@startYear`、`@endYear` 和 `@province`，实现了时间段和区域的灵活筛选。
- **自动处理缺失数据：**
  - 当上一年收入为 `NULL`（例如第一个年份的数据时），使用 `CASE` 语句将增长率设为 `NULL`，避免了除零或错误的计算。
- **结果排序：**
  - 按年份排序，保证了输出数据的时序性，便于直观分析年度变化。

（2）查询在指定时间段区域居民收入最高和最低区域，以及它们与全国平均值的差异。

代码：

```
CREATE PROCEDURE AnalyzeRegionIncomeDifference
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH RegionIncome AS (
        -- 计算每个区域在指定时间段的总收入和平均收入
        SELECT
            province,
            AVG(disposableincome) AS avg_income -- 计算指定时间段的平均收入
        FROM disposableincome
        WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY province
    ),
    NationalAverage AS (
        -- 计算全国平均收入
        SELECT
            AVG(disposableincome) AS national_avg_income
        FROM disposableincome
```

```
        WHERE year BETWEEN @startYear AND @endYear
    ),
    MaxMinRegions AS (
        -- 找到最高收入和最低收入的区域
        SELECT
            province AS max_province,
            avg_income AS max_income,
            NULL AS min_province,
            NULL AS min_income
        FROM RegionIncome
        WHERE avg_income = (SELECT MAX(avg_income) FROM RegionIncome)
        UNION ALL
        SELECT
            NULL AS max_province,
            NULL AS max_income,
            province AS min_province,
            avg_income AS min_income
        FROM RegionIncome
        WHERE avg_income = (SELECT MIN(avg_income) FROM RegionIncome)
    )
    SELECT
        r.province,
        r.avg_income,
        n.national_avg_income,
        r.avg_income - n.national_avg_income AS difference
    FROM RegionIncome r
    CROSS JOIN NationalAverage n
    WHERE r.province IN (
        (SELECT max_province FROM MaxMinRegions WHERE max_province IS NOT NULL),
        (SELECT min_province FROM MaxMinRegions WHERE min_province IS NOT NULL)
    )
    ORDER BY r.avg_income DESC;
END;


EXEC AnalyzeRegionIncomeDifference @startYear = 2020, @endYear = 2022;
```

分析:

1.输入参数:

- `@startYear`: 分析的起始年份。
- `@endYear`: 分析的结束年份。

2.主要计算目标:

- 每个区域在指定时间段内的平均可支配收入。
- 全国平均收入。
- 收入最高和最低的区域。
- 最高和最低区域的平均收入与全国平均收入之间的差异。

3.优点:

- **结构清晰**:

    - 存储过程将计算分为三个 CTE，分别处理区域平均收入、全国平均收入，以及收入最高和最低的区域。
    - 逻辑分明，便于阅读和维护。
- **灵活性高**:
    - 使用动态的 `@startYear` 和 `@endYear` 参数，可以灵活调整分析的时间段。

（3）分析在指定时间段某个区域居民的各项收入。

代码：

```
CREATE PROCEDURE AnalyzeIndividualIncome
    @province NVARCHAR(50), -- 区域
    @startYear INT,         -- 起始年份
    @endYear INT            -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        province,
        year,
        SUM(disposableincome_wageincome) AS total_wage_income,           -- 工资性收入总和
        SUM(disposableincome_operatingincome) AS total_operating_income, -- 经营性收入总和
        SUM(disposableincome_netpropertyincome) AS total_property_income, -- 财产性收入总和
        SUM(disposableIncome_transfernetincome) AS total_transfer_income    -- 转移性收入总和
    FROM disposableincome
    WHERE province = @province -- 动态指定区域
      AND year BETWEEN @startYear AND @endYear -- 动态指定时间段
    GROUP BY province, year
    ORDER BY year;
END;

EXEC AnalyzeIndividualIncome @province = '北京', @startYear = 2020, @endYear = 2022;
```

分析：

1.功能分析

该存储过程接收三个参数：

- `@province NVARCHAR(50)`：指定分析的区域。
- `@startYear INT`：指定分析的起始年份。
- `@endYear INT`：指定分析的结束年份。

存储过程的输出：

- 通过查询 `disposableincome` 表，获取该区域在指定年份范围内的各项居民收入来源的年度汇总数据。
- 输出内容按年份升序排列，便于观察收入变化趋势。

输出字段包括：

- `province`：区域名称。
- `year`：年份。
- `total_wage_income`：工资性收入总和。
- `total_operating_income`：经营性收入总和。
- `total_property_income`：财产性收入总和。
- `total_transfer_income`：转移性收入总和。

2.优点

- 灵活性：
  - 参数化存储过程允许动态指定区域、起始年份和结束年份，满足多种查询需求。
- 清晰的分组和汇总逻辑：
  - 按年份分组汇总后，能够清晰显示每年各收入来源的具体数据。
- 易于扩展：
  - 如果需要添加其他收入类型的统计（如其他收入来源），可以直接在查询中加入新的字段和汇总逻辑。

（4）分析比较指定区域各项收入占比的变化。

代码：

```
CREATE PROCEDURE AnalyzeIncomePercentageChange
    @province NVARCHAR(50), -- 区域
    @startYear INT,         -- 起始年份
    @endYear INT            -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH IncomePercentage AS (
        SELECT
            province,
            year,
            SUM(disposableincome_wageincome) AS total_wage_income,           --
工资性收入总和
            SUM(disposableincome_operatingincome) AS total_operating_income, --
经营性收入总和
            SUM(disposableincome_netpropertyincome) AS total_property_income, --
财产性收入总和
            SUM(disposableIncome_transfernetincome) AS total_transfer_income,
 -- 转移性收入总和
            SUM(disposableincome) AS total_income                             --
总收入
        FROM disposableincome
        WHERE province = @province -- 动态指定区域
          AND year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY province, year
    )
    SELECT
        province,
```

```
        year,
        total_wage_income,
        (total_wage_income * 1.0 / total_income) * 100 AS wage_income_percentage,
-- 工资性收入占比
        total_operating_income,
        (total_operating_income * 1.0 / total_income) * 100 AS
operating_income_percentage, -- 经营性收入占比
        total_property_income,
        (total_property_income * 1.0 / total_income) * 100 AS
property_income_percentage, -- 财产性收入占比
        total_transfer_income,
        (total_transfer_income * 1.0 / total_income) * 100 AS
transfer_income_percentage -- 转移性收入占比
    FROM IncomePercentage
    ORDER BY year;
END;

EXEC AnalyzeIncomePercentageChange @province = '北京', @startYear = 2020, @endYear
= 2022;
```

分析：

1.输入参数

- `@province`：指定查询的区域。
- `@startYear`：指定查询的起始年份。
- `@endYear`：指定查询的结束年份。

2.CTE（公共表表达式）- `IncomePercentage`

CTE的目的是计算每个年份的各类收入总和，并在每个区域内按照年份进行分组。

这一部分会从 `disposableincome` 表中选取符合指定区域和年份范围的所有数据，并对不同类型的收入字段求和。数据的字段包括：

- 工资性收入总和（`total_wage_income`）
- 经营性收入总和（`total_operating_income`）
- 财产性收入总和（`total_property_income`）
- 转移性收入总和（`total_transfer_income`）
- 总收入（`total_income`）

3.查询结果

接下来，查询使用了 `IncomePercentage` CTE 中的数据来计算各类收入占总收入的百分比。

这一部分使用了上面计算的各项收入的总和，并通过除以 `total_income` 来计算各收入类别占总收入的百分比。为了确保计算结果是浮动的（避免整数除法），`total_wage_income * 1.0` 是为了强制将其转换为浮动类型。

4.排序

最后，使用 `ORDER BY year` 对结果按年份进行排序，方便观察收入变化的趋势。

5.执行存储过程

执行时，指定了`@province = '北京'`，`@startYear = 2020`，`@endYear = 2022`，因此查询会输出2020年到2022年间北京地区各项收入占比的变化情况。

（5）查询分析在指定时间段区域居民收入最高的3个区域和最低的3个区域的以及它们的人口特征，包括人口数量、人口密度，等等。人口密度用人口数量/区域面积得到，可不用另行采集存人口密度数据。

代码：

```
ALTER PROCEDURE [dbo].[GetTopAndBottomRegions]
    @Year INT -- 输入参数，指定年份
AS
BEGIN
    -- 开始事务
    BEGIN TRANSACTION;

    -- 防止错误影响后续操作
    BEGIN TRY
        -- 使用CTE计算人口密度
        WITH PopulationDensity AS (
            SELECT
                p.province,
                p.year,
                p.population,
                a.area,
                (p.population / a.area) AS population_density
            FROM population p
            JOIN area a ON p.province = a.province
            WHERE p.year = @Year -- 使用动态年份参数
        ),
        IncomeRanked AS (
            SELECT
                d.province,
                d.year,
                d.disposableincome,
                pd.population,
                pd.population_density
            FROM disposableincome d
            JOIN PopulationDensity pd
                ON d.province = pd.province
                AND d.year = pd.year
            WHERE d.year = @Year -- 使用动态年份参数
        ),
        TopAndBottomRegions AS (
            -- 获取收入最高的3个区域
            SELECT
                *,
                '最高' AS category
            FROM (
                SELECT TOP 3 *
                FROM IncomeRanked
                ORDER BY disposableincome DESC
            ) AS TopRegions

            UNION ALL

            -- 获取收入最低的3个区域
```

```
            SELECT
                *,
                '最低' AS category
            FROM (
                SELECT TOP 3 *
                FROM IncomeRanked
                ORDER BY disposableincome ASC
            ) AS BottomRegions
        )
        -- 输出最终结果
        SELECT
            province AS 区域,
            disposableincome AS 可支配收入,
            population AS 人口数量,
            population_density AS 人口密度,
            category AS 收入类别 -- 新增列
        FROM TopAndBottomRegions
        ORDER BY category DESC, disposableincome DESC;

        -- 提交事务
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- 如果发生错误，则回滚事务
        ROLLBACK TRANSACTION;
        -- 返回错误信息
        THROW;
    END CATCH
END;
```

分析：

1.输入参数

- `@Year`：指定查询的年份，存储过程将根据这个年份过滤数据。

2.逻辑流程

- **事务控制**
  - 开始事务：使用 `BEGIN TRANSACTION;` 启动一个事务，以保证查询过程中的数据一致性。
  - 错误处理：使用 `TRY...CATCH` 语句块来捕捉并处理执行过程中可能出现的任何错误。如果发生错误，则回滚事务，否则提交事务。
- **使用CTE（公共表表达式）**
  - PopulationDensity：此CTE用来计算每个区域在指定年份的人口密度，公式为：

    $$\text{人口密度} = \frac{\text{人口数量}}{\text{区域面积}}$$

    它通过连接 `population` 表和 `area` 表来获取人口数量和区域面积，进而计算人口密度。
  - IncomeRanked：此CTE用来连接 `disposableincome` 和 `PopulationDensity`，为每个区域提供可支配收入以及相应的人口数量和人口密度。它确保了通过指定年份，收入和人口数据的一致性。

- TopAndBottomRegions：这是查询的核心部分，分别获取收入最高和最低的3个区域。通过 `ORDER BY disposableincome DESC` 和 `ORDER BY disposableincome ASC`，分别获得收入最高和最低的3个区域，并使用 `UNION ALL` 将两个结果集合并。最终添加一个 `category` 字段来标识这些区域属于“最高”还是“最低”收入类别。

- **输出最终结果**

  最终的 `SELECT` 查询输出如下信息：

  - 区域（`province`）
  - 可支配收入（`disposableincome`）
  - 人口数量（`population`）
  - 人口密度（`population_density`）
  - 收入类别（`category`，指示是收入最高还是最低）

查询结果按 `category` 和 `disposableincome` 排序，首先显示“最高”收入类别，然后按收入从高到低排序。

- **事务提交与回滚**
  - 提交事务：如果整个查询过程没有发生错误，事务会被提交，确保所有的数据操作都得到持久化。
  - 回滚事务：如果发生错误，事务会被回滚，确保不会影响数据库的一致性。
- **存储过程的优点**
  - 动态查询：通过 `@Year` 参数，可以灵活地查询不同年份的数据，不需要修改存储过程。
  - 事务保障：通过 `BEGIN TRANSACTION` 和 `TRY...CATCH` 机制，确保了查询的原子性与一致性，即使发生错误也能保证数据状态的一致性。
  - 整合多表数据：通过CTE，整合了 `population`、`area`、`disposableincome` 三个表的数据，为分析提供了完整的信息（人口、面积、收入）。
  - 清晰的分类：通过收入类别（“最高”/“最低”），明确标识了收入最高和最低的区域，输出结果易于理解。

（6-1）如果将居民收入划分为高、中、低三个等级，给每个区域收入赋予等级，将结果存储在数据库中，并且统计在指定时间段各种等级的区域数量。

代码：

```
CREATE PROCEDURE GetIncomeLevelStatistics
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    -- 查询统计指定时间段各等级的区域数量
    SELECT
        [level],
        COUNT(DISTINCT province) AS region_count -- 统计每种等级的区域数量
    FROM dbo.incomeLevels
    WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
    GROUP BY [level];
END;
```

```
EXEC GetIncomeLevelStatistics @startYear = 2020, @endYear = 2022;
```

分析：

1.输入参数

- `@startYear`：起始年份，用户指定查询的时间段的开始年份。
- `@endYear`：结束年份，用户指定查询的时间段的结束年份。

2.查询结构

- **查询逻辑**
  - 该存储过程从 `incomeLevels` 表中查询，统计指定时间段内每个收入等级（`level`）对应的区域数量。
  - 字段：
    - `[level]`：表示收入等级（如高、中、低）。假设 `incomeLevels` 表中有这个字段，存储了每个区域的收入等级。
    - `COUNT(DISTINCT province) AS region_count`：统计每个收入等级下区域的数量（去重），即每种等级下有多少个不同的区域。
- **工作原理**
  - incomeLevels表应包含以下字段（假设）：
    - `province`：区域的名称或编号。
    - `year`：年份。
    - `level`：收入等级（如高、中、低）。
  - 查询会根据给定的时间范围，统计每种收入等级的区域数量。
- **存储过程的优点**
  - 动态查询：通过传入 `@startYear` 和 `@endYear` 参数，用户可以灵活地查询不同时间段内的统计数据。
  - 简洁明了：查询非常简洁，只需要查询 `incomeLevels` 表，并进行简单的 `GROUP BY` 和 `COUNT` 聚合操作。
  - 实时统计：通过动态查询，可以实时地获得每个时间段内各收入等级的区域分布，适用于不同时期的分析。

（6-2）查询哪些区域居民收入一直位于高等级，哪些一直位于低等级。注意不同时间段划分高中低等级的标准不同。

代码：

```
CREATE PROCEDURE GetExtremeLevelRegions
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH HighLevelRegions AS (
        -- 查询一直位于高等级的区域
        SELECT
            province,
```

```
            '最高' AS level_status -- 标注为最高
        FROM dbo.incomeLevels
        WHERE [level] = '高'
          AND year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY province
        HAVING COUNT(DISTINCT year) = (SELECT COUNT(DISTINCT year) FROM
dbo.incomeLevels WHERE year BETWEEN @startYear AND @endYear)
    ),
    LowLevelRegions AS (
        -- 查询一直位于低等级的区域
        SELECT
            province,
            '最低' AS level_status -- 标注为最低
        FROM dbo.incomeLevels
        WHERE [level] = '低'
          AND year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY province
        HAVING COUNT(DISTINCT year) = (SELECT COUNT(DISTINCT year) FROM
dbo.incomeLevels WHERE year BETWEEN @startYear AND @endYear)
    )
    -- 合并两个结果
    SELECT
        province,
        level_status
    FROM HighLevelRegions
    UNION ALL
    SELECT
        province,
        level_status
    FROM LowLevelRegions;
END;


EXEC GetExtremeLevelRegions @startYear = 2020, @endYear = 2022;
```

分析：

1.输入参数

- `@startYear`：起始年份，用于指定查询的时间范围的起始年份。
- `@endYear`：结束年份，用于指定查询的时间范围的结束年份。

2.存储过程逻辑

存储过程的核心任务是查询每个区域在给定时间段内是否始终处于高等级（`'高'`）或低等级（`'低'`）。要实现这个功能，存储过程分为两个主要步骤：

- **查询高等级区域（HighLevelRegions CTE）**
  - 目的：查询在指定时间段内，所有年份的收入等级始终为“高”的区域。
  - 查询条件：
    - `WHERE [level] = '高'`：筛选收入等级为高的记录。
    - `AND year BETWEEN @startYear AND @endYear`：限定在指定的年份范围内。
  - `HAVING` **子句**：

        - `COUNT(DISTINCT year)`：计算该区域在所有年份中的唯一年份数。
        - `= (SELECT COUNT(DISTINCT year) ...)`：确保该区域在所有年份中都有“高”收入等级，即区域的收入等级在整个时间段内始终为高。
- **查询低等级区域（LowLevelRegions CTE）**
    - 目的：查询在指定时间段内，所有年份的收入等级始终为“低”的区域。
    - 查询条件：
        - `WHERE [level] = '低'`：筛选收入等级为低的记录。
        - `AND year BETWEEN @startYear AND @endYear`：限定在指定的年份范围内。
    - `HAVING` **子句**：
        - 与 `HighLevelRegions` 相同，使用 `COUNT(DISTINCT year)` 确保该区域的收入等级在整个时间段内始终为低。
- **合并查询结果**
    - `UNION ALL`：将 `HighLevelRegions` 和 `LowLevelRegions` 查询结果合并，得到最终的结果集。
    - `province`：表示区域。
    - `level_status`：收入等级状态，分为 `'最高'` 或 `'最低'`。
- **存储过程的优点**
    - **动态查询**：存储过程通过 `@startYear` 和 `@endYear` 参数，使得查询可以灵活地应用于任何时间段，适应不同时间范围内的分析需求。
    - **清晰的分组和过滤逻辑**：通过使用 `GROUP BY` 和 `HAVING COUNT(DISTINCT year)`，可以确保每个区域的收入等级在所选的时间段内是始终一致的（即始终处于高或低等级）。
    - **使用** `WITH` **子句（CTE）**：使用了公共表表达式（CTE），使查询逻辑清晰，容易维护和理解。将高等级和低等级区域的查询分别写在两个 CTE 中，逻辑上分明。
    - **简洁高效**：存储过程的查询较为简洁，执行效率较高。通过一次查询，直接获取了始终处于高等级和低等级的区域信息。

（7）比较东部、西部、中部、东北部地区的居民收入的环比增长率。

代码：

```
CREATE PROCEDURE CompareRegionalGrowthRate
    @startYear INT, -- 起始年份
    @endYear INT    -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH RegionalIncome AS (
        -- 计算每个区域每年的居民收入总和
        SELECT
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
```

```
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知区域'
            END AS region,
            year,
            SUM(disposableincome) AS total_income
        FROM disposableincome
        WHERE year BETWEEN @startYear AND @endYear -- 动态指定时间段
        GROUP BY
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知区域'
            END,
            year
    ),
    RegionalGrowthRate AS (
        -- 计算环比增长率
        SELECT
            region,
            year,
            total_income,
            LAG(total_income) OVER (PARTITION BY region ORDER BY year) AS
prev_income,
            CASE
                WHEN LAG(total_income) OVER (PARTITION BY region ORDER BY year)
IS NOT NULL THEN
                    ((total_income - LAG(total_income) OVER (PARTITION BY region
ORDER BY year)) * 1.0
                    / LAG(total_income) OVER (PARTITION BY region ORDER BY year))
* 100
                ELSE NULL
            END AS growth_rate
        FROM RegionalIncome
    )
    -- 输出结果
    SELECT
        region,
        year,
        total_income,
        prev_income,
        growth_rate
    FROM RegionalGrowthRate
    ORDER BY region, year;
END;


EXEC CompareRegionalGrowthRate @startYear = 2020, @endYear = 2022;
```

分析：

1.输入参数

- `@startYear INT`：起始年份，用户指定查询的开始年份。
- `@endYear INT`：结束年份，用户指定查询的结束年份。

2.环比增长率的计算公式

环比增长率的公式通常为：

$$\text{环比增长率} = \frac{\text{当前年度收入} - \text{前一年收入}}{\text{前一年收入}} \times 100\%$$

这个公式可以用来计算东部、西部、中部、东北部地区每年的收入增长率。

3.存储过程设计

- **地区划分**

假设在数据库中，`province` 表示区域名称。我们将区域划分为东部、西部、中部和东北部，并假设这些区域的划分在另一个表（比如 `region` 表）中已经定义。

- **环比增长率计算**

  为了比较各区域居民收入的环比增长率，存储过程的步骤可能如下：

  - **通过 `WITH` 子句定义公共表表达式（CTE）**：
    - 获取每个区域每年可支配收入的统计数据。
    - 计算环比增长率。
  - **通过 `JOIN` 操作**：
    - 将 `disposableincome` 表和 `region` 表连接，得到每个区域每年收入数据的同时，确保每个区域都能按年份进行排序，计算环比增长。
  - **最终查询结果**：
    - 返回各个区域每年的收入和环比增长率。

4.存储过程分析

- **CTE：IncomeData**
  - **目的**：该 CTE 用于计算每个区域每年可支配收入，并使用 `LAG()` 函数获取前一年该区域的收入。
  - `LAG(d.disposableincome)`：`LAG` 函数获取某一行之前的行数据，这里它获取了当前年份的收入和前一年收入。通过 `PARTITION BY d.province` 来确保按区域分组，`ORDER BY d.year` 保证按年份排序。
  - `WHERE d.year BETWEEN @startYear AND @endYear`：限定查询的年份范围。
- **环比增长率计算**
  - **公式**：通过 `(disposableincome - previous_year_income) * 1.0 / previous_year_income * 100` 计算环比增长率。
  - `CASE` **语句**：处理除以零的情况，防止前一年收入为零的情况导致错误。
- **输出结果**
  - 返回字段：
    - `区域`：每个区域的名称。
    - `地区类型`：对应的地区类型（东部、西部、中部、东北部）。

- `年份`：每年的数据。
- `当前年份收入`：当前年度的收入数据。
- `前一年收入`：前一年的收入数据。
- `环比增长率`：环比增长率，表示当前年份和前一年的收入差异。

（8-1）分析比较在指定时间段指定区域的居民收入与社会消费品零售额的变化趋势。

代码：

```
CREATE PROCEDURE AnalyzeIncomeAndSalesTrends
    @province NVARCHAR(50), -- 区域
    @startYear INT,         -- 起始年份
    @endYear INT            -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    WITH IncomeGrowth AS (
        SELECT
            di.province,
            di.year,
            di.disposableincome AS total_income,
            LAG(di.disposableincome) OVER (PARTITION BY di.province ORDER BY
di.year) AS prev_income,
            CASE
                WHEN LAG(di.disposableincome) OVER (PARTITION BY di.province
ORDER BY di.year) IS NOT NULL THEN
                    (di.disposableincome - LAG(di.disposableincome) OVER
(PARTITION BY di.province ORDER BY di.year)) * 1.0
                    / LAG(di.disposableincome) OVER (PARTITION BY di.province
ORDER BY di.year) * 100
                ELSE NULL
            END AS income_growth_rate
        FROM disposableincome di
        WHERE di.year BETWEEN @startYear AND @endYear
    ),
    SalesGrowth AS (
        SELECT
            sa.province,
            sa.year,
            sa.sales_amount AS total_sales,
            LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER BY sa.year)
AS prev_sales,
            CASE
                WHEN LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER BY
sa.year) IS NOT NULL THEN
                    (sa.sales_amount - LAG(sa.sales_amount) OVER (PARTITION BY
sa.province ORDER BY sa.year)) * 1.0
                    / LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER
BY sa.year) * 100
                ELSE NULL
            END AS sales_growth_rate
        FROM salesAmount sa
        WHERE sa.year BETWEEN @startYear AND @endYear
```

```
    ),
    CombinedGrowth AS (
        SELECT
            ig.province,
            ig.year,
            ig.income_growth_rate,
            sg.sales_growth_rate,
            CASE
                WHEN ig.income_growth_rate IS NOT NULL AND sg.sales_growth_rate
IS NOT NULL THEN
                    CASE
                        WHEN (ig.income_growth_rate >= 0 AND sg.sales_growth_rate
>= 0) OR
                             (ig.income_growth_rate < 0 AND sg.sales_growth_rate
< 0) THEN '相同'
                        ELSE '不同'
                    END
                ELSE '数据不足'
            END AS trend_comparison
        FROM IncomeGrowth ig
        JOIN SalesGrowth sg ON ig.province = sg.province AND ig.year = sg.year
    )
    SELECT
        province AS 区域,
        year AS 年份,
        income_growth_rate AS 居民收入变化率,
        sales_growth_rate AS 社会消费品零售额变化率,
        trend_comparison AS 趋势说明
    FROM CombinedGrowth
    WHERE province = @province
      AND year BETWEEN @startYear AND @endYear
    ORDER BY year;
END;


EXEC AnalyzeIncomeAndSalesTrends @province = '北京', @startYear = 2020, @endYear =
2022;
```

分析：

1.输入参数

- `@province NVARCHAR(50)`：用户指定要分析的区域。
- `@startYear INT`：指定起始年份，用于筛选数据。
- `@endYear INT`：指定结束年份，用于筛选数据。

2.存储过程步骤

存储过程分为四个主要部分：

- **IncomeGrowth CTE（居民收入增长率）**
  - **目的**：计算指定区域在指定年份范围内每年居民收入的增长率。
  - **LAG() 函数**：通过 `LAG(di.disposableincome)` 获取每个区域前一年的收入，以计算年收入的变化。

- **增长率公式**：如果前一年的收入不为 `NULL`，则计算该年的收入增长率：

$$\text{收入增长率} = \frac{\text{当前年收入} - \text{前一年收入}}{\text{前一年收入}} \times 100$$

- **SalesGrowth CTE（社会消费品零售额增长率）**
  - **目的**：计算指定区域在指定年份范围内每年社会消费品零售额的增长率。
  - **LAG() 函数**：同样使用 `LAG(sa.sales_amount)` 获取前一年的零售额，以便计算增长率。
  - **增长率公式**：使用和收入增长率相同的公式计算社会消费品零售额的增长率。
- **CombinedGrowth CTE（收入与销售的趋势对比）**
  - **目的**：将 `IncomeGrowth` 和 `SalesGrowth` 两个 CTE 的结果进行合并，并对比它们的增长趋势。
  - 趋势比较：
    - 如果收入增长率和销售增长率的符号相同（都为正或都为负），则认为它们的趋势相同（`'相同'`）。
    - 如果一个增长率为正，一个为负，则认为它们的趋势不同（`'不同'`）。
    - 如果有缺失数据（例如某些年份的收入或销售数据缺失），则标记为 `'数据不足'`。
- **最终输出**
  - 从 `CombinedGrowth` 中筛选出指定区域（`province = @province`）和指定年份范围内（`year BETWEEN @startYear AND @endYear`）的数据。
  - 输出的字段包括：
    - `区域`：显示区域名。
    - `年份`：显示年份。
    - `居民收入变化率`：收入的年度变化率。
    - `社会消费品零售额变化率`：零售额的年度变化率。
    - `趋势说明`：说明收入和零售额变化趋势是否一致。

3.存储过程分析

- **`IncomeGrowth` CTE（收入增长率）**
  - 通过 `LAG()` 函数获取前一年的收入数据，并计算当前年份的收入增长率。
  - 使用 `CASE` 语句处理前一年数据缺失的情况。
  - 按年份排序并筛选指定时间段的数据。
- **`SalesGrowth` CTE（销售额增长率）**
  - 计算销售额的年增长率，同样使用 `LAG()` 函数来获取前一年的销售额。
  - 同样通过 `CASE` 语句处理数据缺失情况，确保查询的数据完整。
- **`CombinedGrowth` CTE（趋势对比）**
  - 将 `IncomeGrowth` 和 `SalesGrowth` 两个 CTE 合并，并对比它们的年增长率趋势。
  - 如果收入和销售额的增长率符号一致，则认为趋势一致（`'相同'`）；如果符号相反，则认为趋势不同（`'不同'`）；若数据不足，则返回 `'数据不足'`。
- **查询结果**

- 根据用户指定的区域（`province`）和时间段（`startYear` 至 `endYear`）筛选数据，输出每年收入和销售额的增长率，以及它们的趋势比较结果。

（8-2）在最近三年，哪些区域的居民收入与社会消费品零售额的变化趋势相同?

代码：

```
CREATE PROCEDURE GetMatchingRegions
AS
BEGIN
    -- 临时表：计算居民收入的环比增长率
    WITH IncomeGrowth AS (
        SELECT
            di.province,
            di.year,
            LAG(di.disposableincome) OVER (PARTITION BY di.province ORDER BY
di.year) AS prev_income,
            CASE
                WHEN LAG(di.disposableincome) OVER (PARTITION BY di.province
ORDER BY di.year) IS NOT NULL THEN
                    (di.disposableincome - LAG(di.disposableincome) OVER
(PARTITION BY di.province ORDER BY di.year)) * 1.0
                    / LAG(di.disposableincome) OVER (PARTITION BY di.province
ORDER BY di.year) * 100
                ELSE NULL
            END AS income_growth_rate
        FROM disposableincome di
        WHERE di.year BETWEEN 2020 AND 2023 -- 最近三年
    ),

    -- 临时表：计算社会消费品零售额的环比增长率
    SalesGrowth AS (
        SELECT
            sa.province,
            sa.year,
            LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER BY sa.year)
AS prev_sales,
            CASE
                WHEN LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER BY
sa.year) IS NOT NULL THEN
                    (sa.sales_amount - LAG(sa.sales_amount) OVER (PARTITION BY
sa.province ORDER BY sa.year)) * 1.0
                    / LAG(sa.sales_amount) OVER (PARTITION BY sa.province ORDER
BY sa.year) * 100
                ELSE NULL
            END AS sales_growth_rate
        FROM salesAmount sa
        WHERE sa.year BETWEEN 2020 AND 2023 -- 最近三年
    ),

    -- 临时表：合并居民收入和社会消费品零售额的增长率，并比较变化趋势
    CombinedGrowth AS (
        SELECT
            ig.province,
            ig.year,
            ig.income_growth_rate,
```

```
            sg.sales_growth_rate,
            CASE
                WHEN ig.income_growth_rate IS NOT NULL AND sg.sales_growth_rate
IS NOT NULL THEN
                    CASE
                        WHEN (ig.income_growth_rate >= 0 AND sg.sales_growth_rate
>= 0) OR
                             (ig.income_growth_rate < 0 AND sg.sales_growth_rate
< 0) THEN '相同'
                        ELSE '不同'
                    END
                ELSE '数据不足'
            END AS trend_comparison
        FROM IncomeGrowth ig
        JOIN SalesGrowth sg ON ig.province = sg.province AND ig.year = sg.year
    ),

    -- 临时表：统计每个区域在最近三年中变化趋势相同的年份数量
    MatchingRegions AS (
        SELECT
            province,
            COUNT(*) AS matching_years
        FROM CombinedGrowth
        WHERE trend_comparison = '相同' -- 变化趋势相同
        GROUP BY province
    )

    -- 查询哪些区域最近三年变化趋势全部相同
    SELECT
        province AS 区域
    FROM MatchingRegions
    WHERE matching_years = 3; -- 确保三年全部相同

END;

EXEC GetMatchingRegions;
```

分析：

1,输入参数：

本存储过程不接收任何参数。它查询的是 **最近三年**（2020-2023年）数据，并通过比较居民收入和社会消费品零售额的年增长率，得出趋势相同的区域。

2.存储过程结构

- **IncomeGrowth CTE（居民收入增长率）**
  - 计算每个区域在2020至2023年间的居民收入增长率。
  - **LAG() 函数**：`LAG(di.disposableincome)` 用于获取前一年（同一区域）的收入，以便计算收入的年增长率。
  - **增长率计算**：通过计算当前年和前一年收入的差异，并除以前一年的收入，再乘以100，得到收入的年增长率：

$$\text{收入增长率} = \frac{\text{当前年收入} - \text{前一年收入}}{\text{前一年收入}} \times 100$$

    - 过滤条件：`WHERE di.year BETWEEN 2020 AND 2023`，只选取最近三年的数据。
- **SalesGrowth CTE（社会消费品零售额增长率）**
    - 计算每个区域在2020至2023年间的社会消费品零售额增长率。
    - 同样使用 `LAG(sa.sales_amount)` 获取前一年（同一区域）的销售额，用于计算销售额的年增长率。
    - **增长率计算**：类似收入的计算方式，计算零售额的年增长率。
    - 过滤条件：`WHERE sa.year BETWEEN 2020 AND 2023`，确保查询的是最近三年的数据。
- **CombinedGrowth CTE（收入与销售的增长率趋势对比）**
    - 将 `IncomeGrowth` 和 `SalesGrowth` 两个 CTE 合并，并对比它们的增长趋势。
- 趋势比较：
    - 如果收入和销售额的增长率符号相同（都为正或都为负），则认为趋势相同（`'相同'`）。
    - 如果一个增长率为正，一个为负，则认为趋势不同（`'不同'`）。
    - 如果某年某区域的收入或销售额数据缺失（`NULL`），则标记为 `'数据不足'`。
- 趋势比较逻辑：
    - `WHEN (ig.income_growth_rate >= 0 AND sg.sales_growth_rate >= 0) OR (ig.income_growth_rate < 0 AND sg.sales_growth_rate < 0) THEN '相同'`
- 如果两者增长率的符号相同，则标记为 `'相同'`；否则，标记为 `'不同'`。
- **MatchingRegions CTE（符合条件的区域）**
    - 统计每个区域在最近三年中变化趋势相同的年份数量。
    - **筛选条件**：`WHERE trend_comparison = '相同'`，只计算那些收入和销售额增长趋势相同的年份。
    - 对每个区域进行分组，计算每个区域趋势相同的年份数（`COUNT(*) AS matching_years`）。
- **最终查询：选出三年全部相同趋势的区域**
    - 查询哪些区域的 **趋势相同的年份数** 为3，即最近三年内，收入和销售额的年增长率始终相同。
    - **条件**：`WHERE matching_years = 3`，只筛选出趋势相同的年份数为3的区域。
- **最终输出**
    - **输出字段**：查询结果包括区域名（`province`）。
    - 这些区域是那些在2020至2023年间，收入和社会消费品零售额的年增长率趋势始终相同的区域。

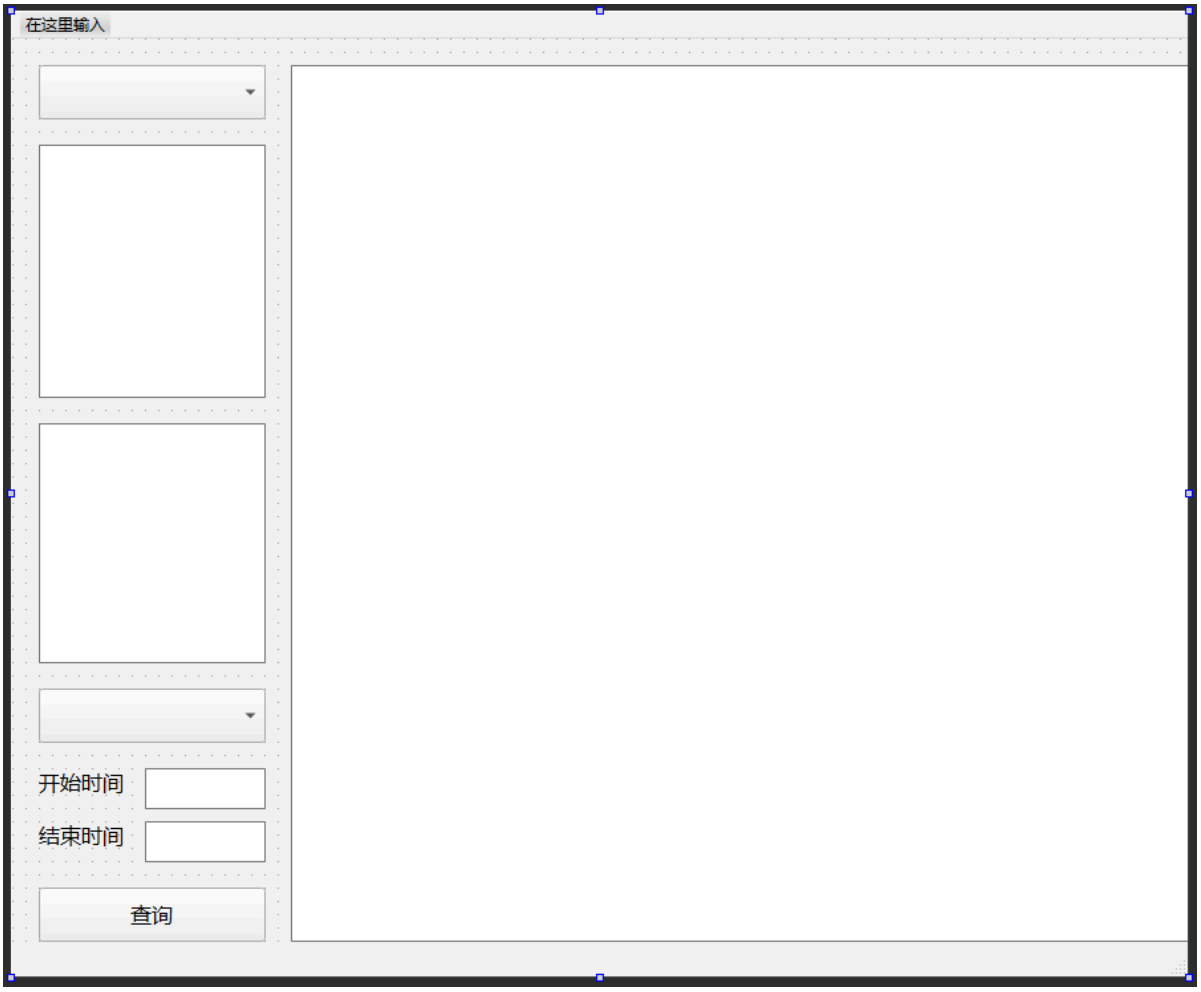
3.存储过程的逻辑总结

- **计算居民收入的年增长率**：通过 `LAG()` 函数获取每个区域前一年的收入，计算收入的年增长率。
- **计算社会消费品零售额的年增长率**：同样使用 `LAG()` 函数获取前一年销售额，计算销售额的年增长率。
- **比较收入和零售额的增长率趋势**：对比两者的增长率，判断是否一致（符号相同即认为趋势相同）。

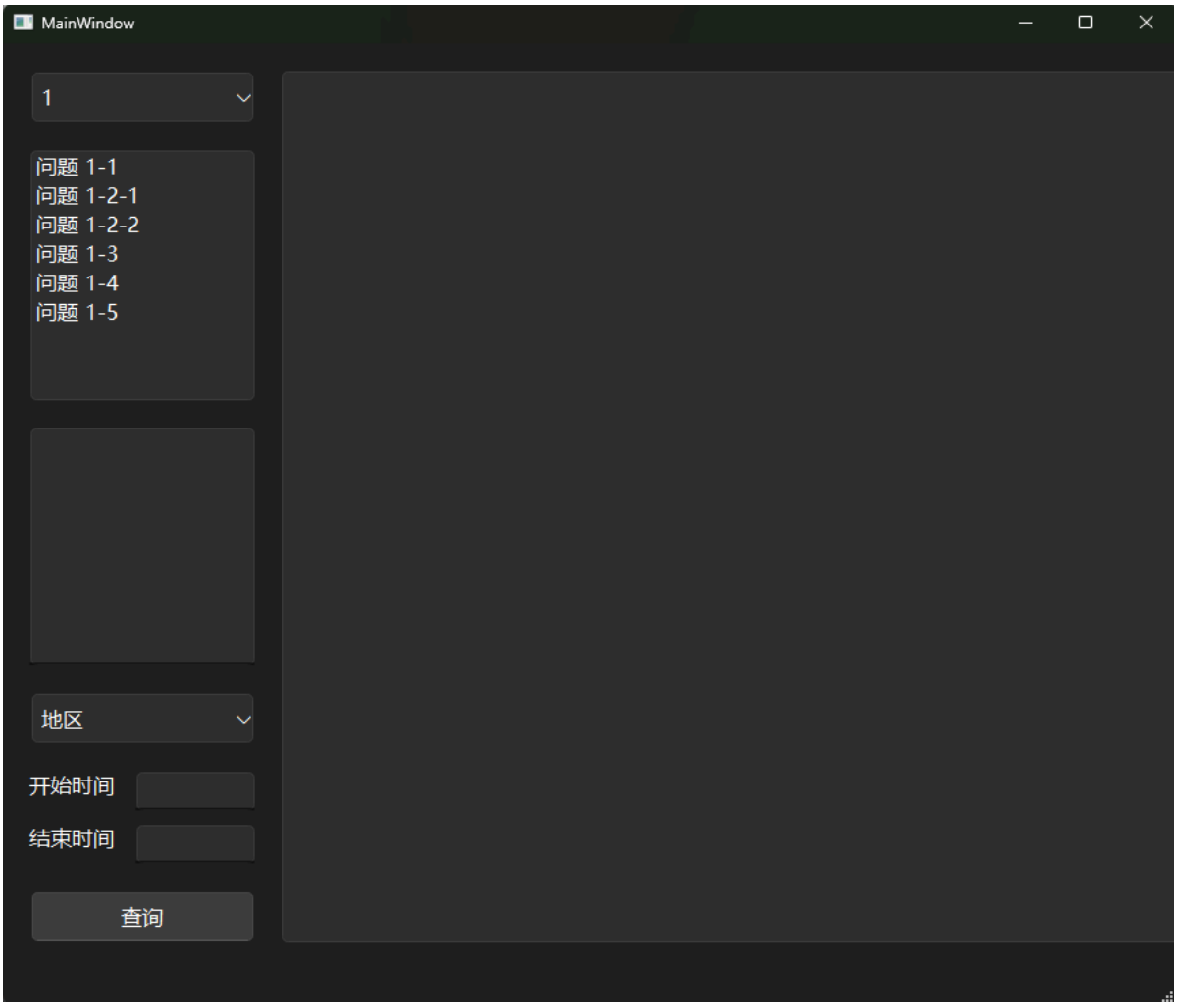
- **筛选符合条件的区域**：统计每个区域在最近三年内的趋势相同的年份数量，筛选出趋势一致的区域。
- **输出符合条件的区域**：最终输出在三年内趋势一致的区域。

5.UI设计

使用Qt设计前端界面，页面布局如图所示。



运行界面如图所示。



6.其他查询设计

6.1 问题设计

（1）查询在指定时间段指定区域的GDP。

（2）查询指定时间段GDP总额最高和最低区域。

（3）分析比较指定时间段东部、西部、中部、东北部地区的GDP，并进行排序。

6.2表格设计

从中经数据库中下载的各个省份的年度GDP数据如图1所示，转化为图2所示的格式，导入到数据库中，数据库表格如图3所示。

1	数据来源：中经数据（CEIdata）		
2	序列ID	3	20
3	指标	GDP	GDP
4	地区	北京	天津
5	频度	年	年
6	单位	亿元	亿元
7	2010	14964.02	6830.76
8	2011	17188.8	8112.51
9	2012	19024.73	9043.02

图1 中经数据库下载的GDP数据

1	北京	2010	14964.02
2	北京	2011	17188.8
3	北京	2012	19024.73
4	北京	2013	21134.58
5	北京	2014	22926
6	北京	2015	24779.1
7	北京	2016	27041.2
8	北京	2017	29883

图2 转化格式后的GDP数据

	列名	数据类型	允许 Null 值
▶	province	varchar(50)	☑
	year	int	☑
	gdp	decimal(10, 2)	☑

图3 数据库中GDP表格结构

6.3查询设计

（1）查询在指定时间段指定区域的GDP。

代码：

```
CREATE PROCEDURE GetGDPByRegionAndTime
    @Province NVARCHAR(50), -- 指定省份
    @StartYear INT,         -- 开始年份
    @EndYear INT            -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    SELECT province, year, gdp
    FROM dbo.gdp
    WHERE province = @Province
      AND year BETWEEN @StartYear AND @EndYear;
END;

EXEC GetGDPByRegionAndTime @Province = '北京', @StartYear = 2015, @EndYear = 2020;
```

分析：

1.查询参数：

- 参数 `@Province` 指定省份（如“北京”）。
- 参数 `@StartYear` 和 `@EndYear` 指定时间范围。

2.查询结果格式：

- 返回的结果包括 `province`、`year` 和 `gdp` 三列，便于进一步处理。

（2）查询指定时间段GDP总额最高和最低区域。

代码：

```
CREATE PROCEDURE GetMaxAndMinGDPByTime
    @StartYear INT, -- 开始年份
```

```
    @EndYear INT     -- 结束年份
AS
BEGIN
    SET NOCOUNT ON;

    -- 使用公共表表达式计算各省份的GDP总额
    WITH TotalGDP AS (
        SELECT
            province,
            SUM(gdp) AS total_gdp
        FROM dbo.gdp
        WHERE year BETWEEN @StartYear AND @EndYear -- 指定时间段
        GROUP BY province
    )
    -- 查询GDP总额最高和最低的省份
    SELECT
        province,
        total_gdp,
        'Max' AS Type
    FROM TotalGDP
    WHERE total_gdp = (SELECT MAX(total_gdp) FROM TotalGDP)
    UNION
    SELECT
        province,
        total_gdp,
        'Min' AS Type
    FROM TotalGDP
    WHERE total_gdp = (SELECT MIN(total_gdp) FROM TotalGDP);
END;

EXEC GetMaxAndMinGDPByTime @StartYear = 2015, @EndYear = 2020;
```

分析：

1.目标：

- 在指定的时间范围内，通过 `@StartYear` 和 `@EndYear` 参数动态计算各区域的GDP总额，筛选出GDP总额最高和最低的区域。

2.实现逻辑：

- **Step 1**：通过公共表表达式 (CTE) `TotalGDP` 计算各区域在指定时间段的GDP总额。
- **Step 2**：从 `TotalGDP` 中筛选出 GDP 总额的最大值（`MAX(total_gdp)`）和最小值（`MIN(total_gdp)`）的区域。
- **Step 3**：返回结果包含三个字段：`province`（区域名称）、`total_gdp`（总GDP值）和 `Type`（标识“Max”或“Min”）。

3.输入参数：

- `@StartYear`：指定查询的起始年份。
- `@EndYear`：指定查询的结束年份。

4.输出结果：

- 查询结果包含 GDP 总额最高和最低的省份，以及对应的总GDP值。

（3）分析比较指定时间段东部、西部、中部、东北部地区的GDP，并进行排序。

代码：

```
CREATE PROCEDURE GetRegionalGDP
    @StartYear INT,
    @EndYear INT
AS
BEGIN
    SET NOCOUNT ON;

    WITH RegionalGDP AS (
        SELECT
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知'
            END AS region,
            SUM(gdp) AS total_gdp
        FROM dbo.gdp
        WHERE year BETWEEN @StartYear AND @EndYear
        GROUP BY
            CASE
                WHEN province IN ('北京', '天津', '河北', '上海', '江苏', '浙江', '福
建', '山东', '广东', '海南') THEN '东部'
                WHEN province IN ('山西', '安徽', '江西', '河南', '湖北', '湖南')
THEN '中部'
                WHEN province IN ('内蒙古', '广西', '重庆', '四川', '贵州', '云南',
'西藏', '陕西', '甘肃', '青海', '宁夏', '新疆') THEN '西部'
                WHEN province IN ('辽宁', '吉林', '黑龙江') THEN '东北'
                ELSE '未知'
            END
    )
    SELECT
        region,
        total_gdp
    FROM RegionalGDP
    ORDER BY total_gdp DESC;
END;

EXEC GetRegionalGDP @StartYear = 2015, @EndYear = 2020;
```

分析：

1.参数化查询：

- 通过 `@StartYear` 和 `@EndYear` 参数，用户可以指定分析的时间范围，提高了存储过程的通用性。

2.区域划分：

- 使用 `CASE` 表达式对省份进行区域划分，包括东部、中部、西部和东北部四个区域，并为其他未知省份分配为 `未知` 区域。
- 这种划分方式与题目需求一致，能够清晰地比较各个区域的 GDP 总量。

3.GDP 汇总：

- 通过 `SUM(gdp)` 对每个区域的 GDP 进行汇总，完成了总量统计。
- 聚合操作基于 `GROUP BY`，对四大区域进行了分组。

4.时间范围过滤：

- 使用 `WHERE year BETWEEN @StartYear AND @EndYear`，对数据按时间区间进行了筛选，确保只统计指定时间段内的数据。

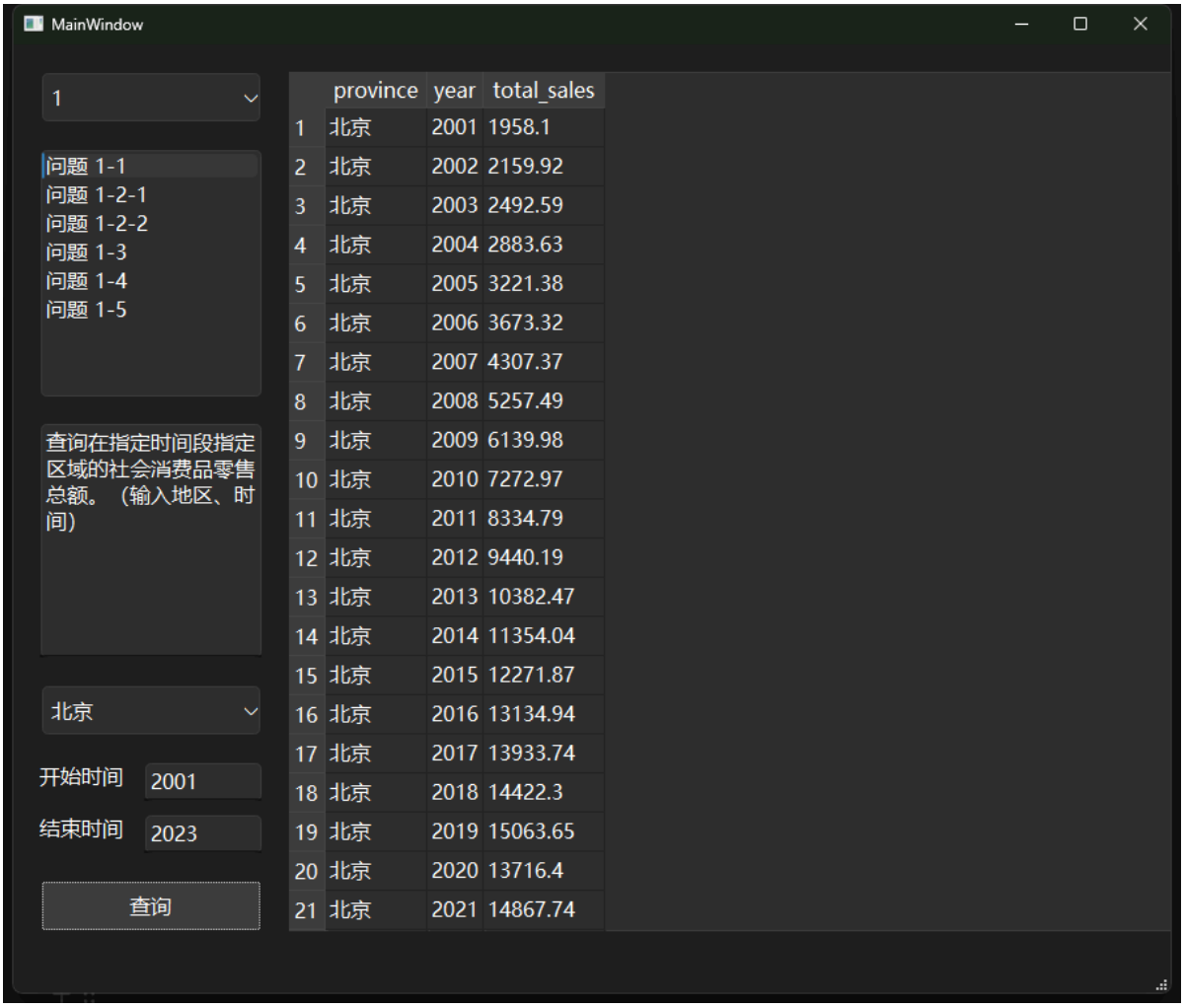
5.排序输出：

- 使用 `ORDER BY total_gdp DESC`，对结果按 GDP 总量从高到低排序，便于直观地查看各区域的经济贡献大小。

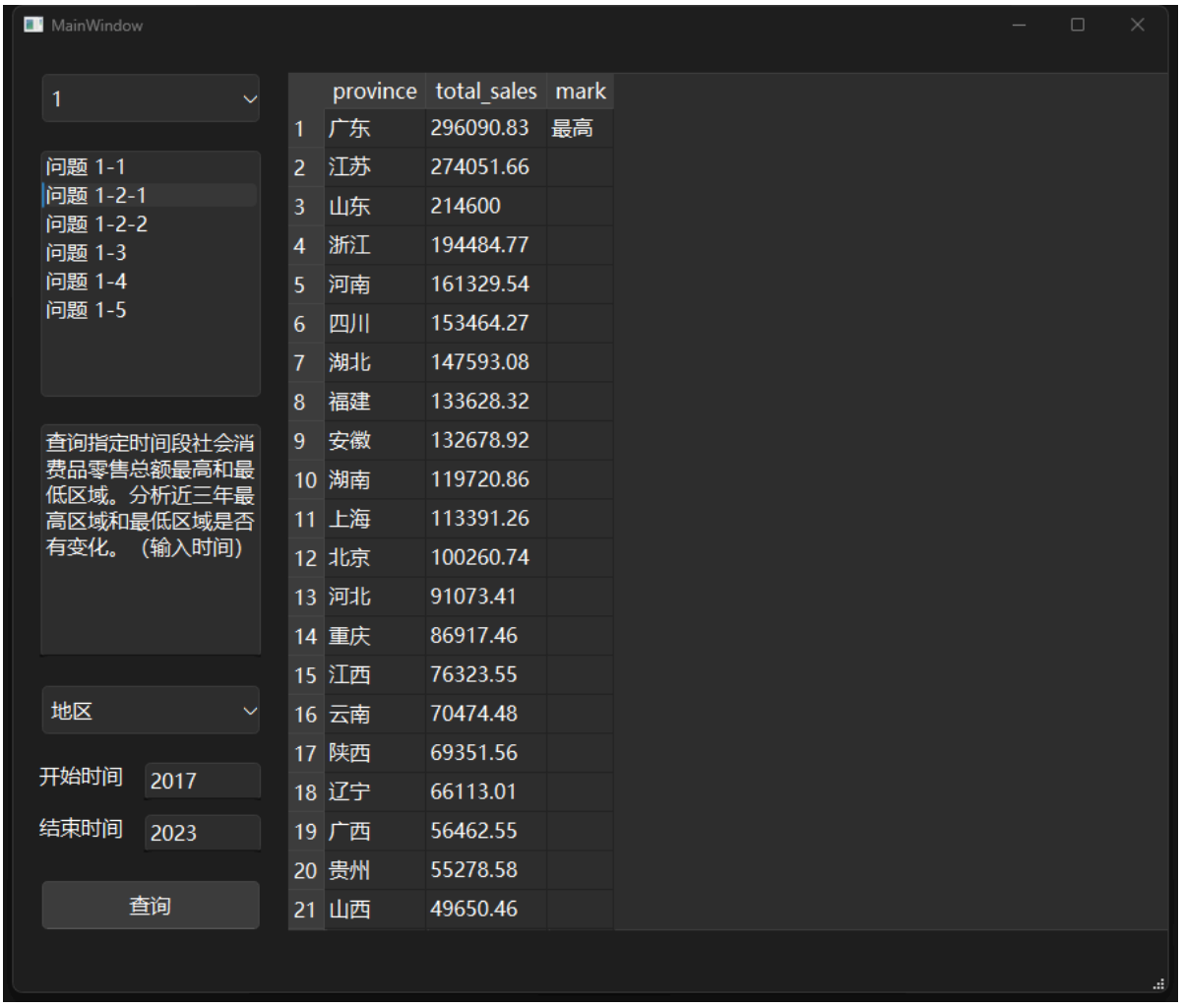
7.结果展示

7.1 社会消费变化及区域差异

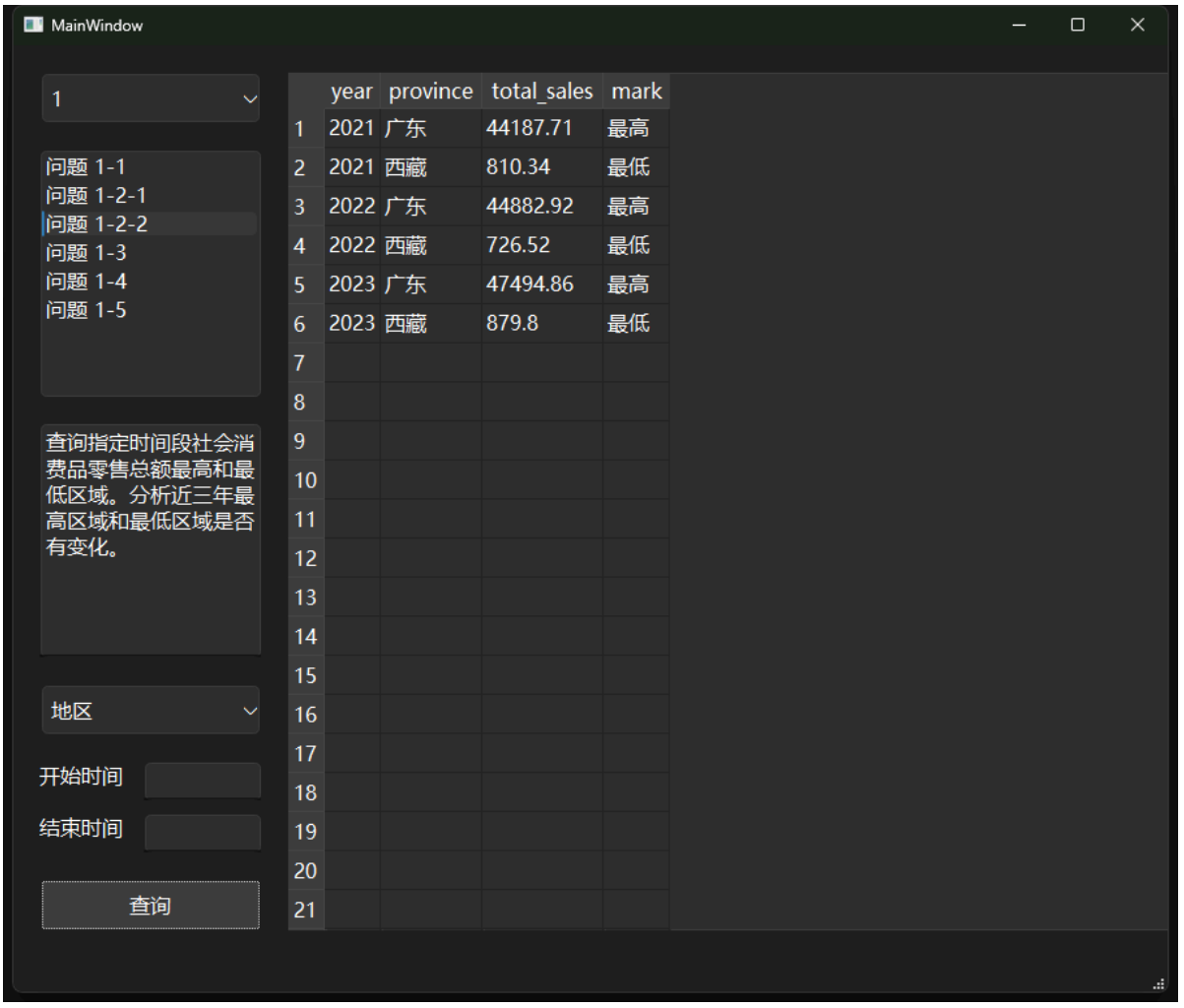
问题1-1运行结果如图所示



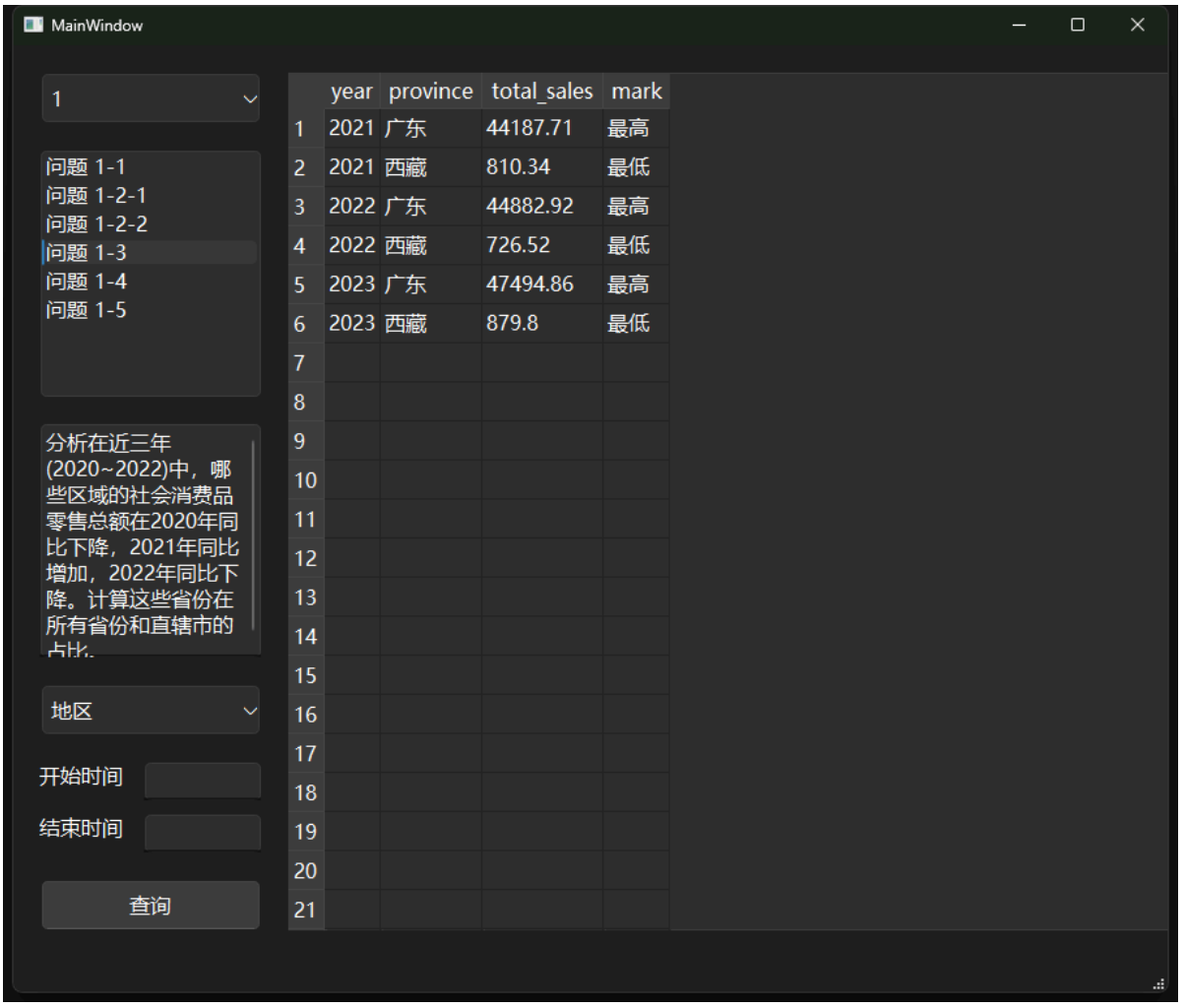
问题1-2-1运行结果如图所示



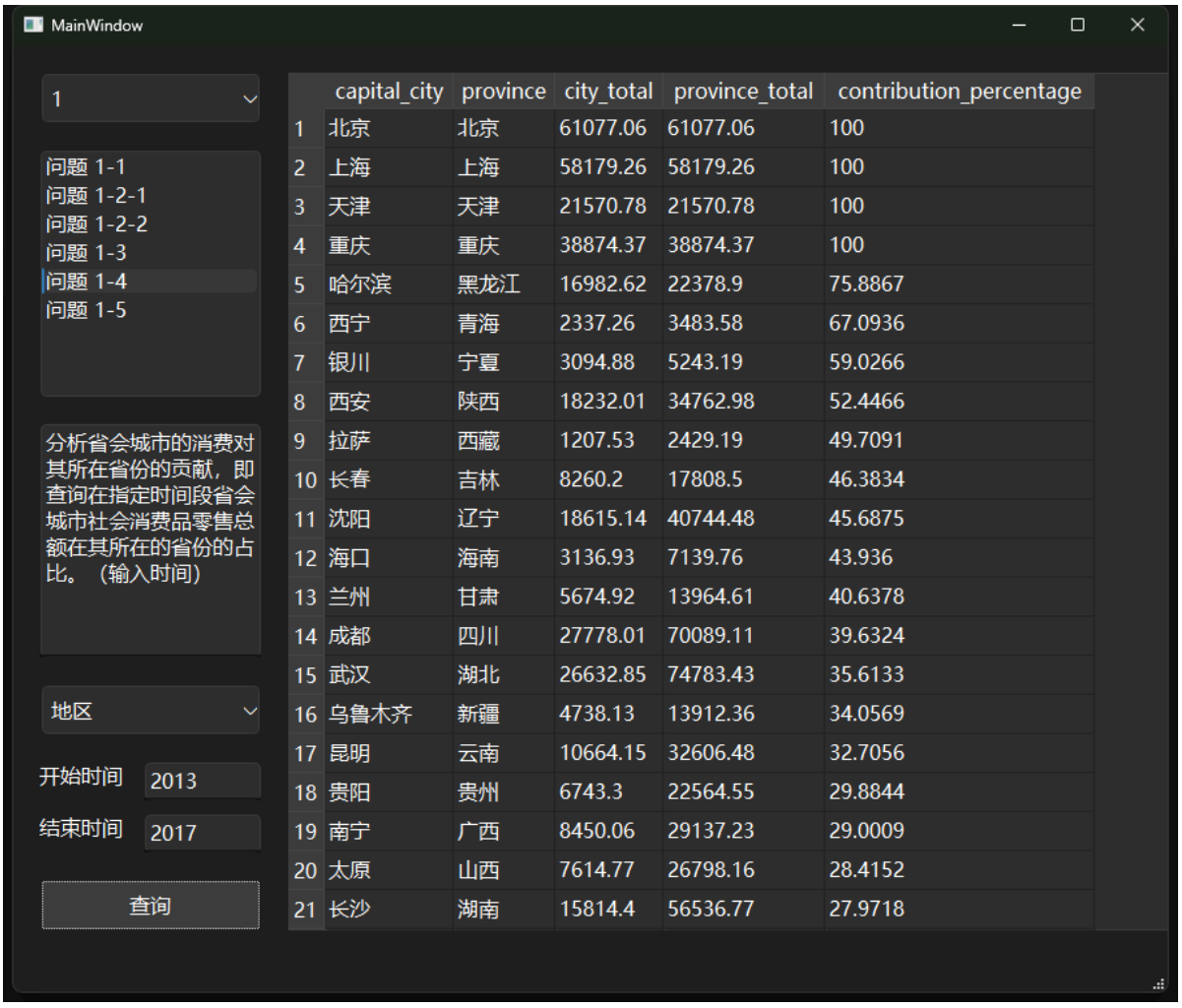
问题1-2-2运行结果如图所示



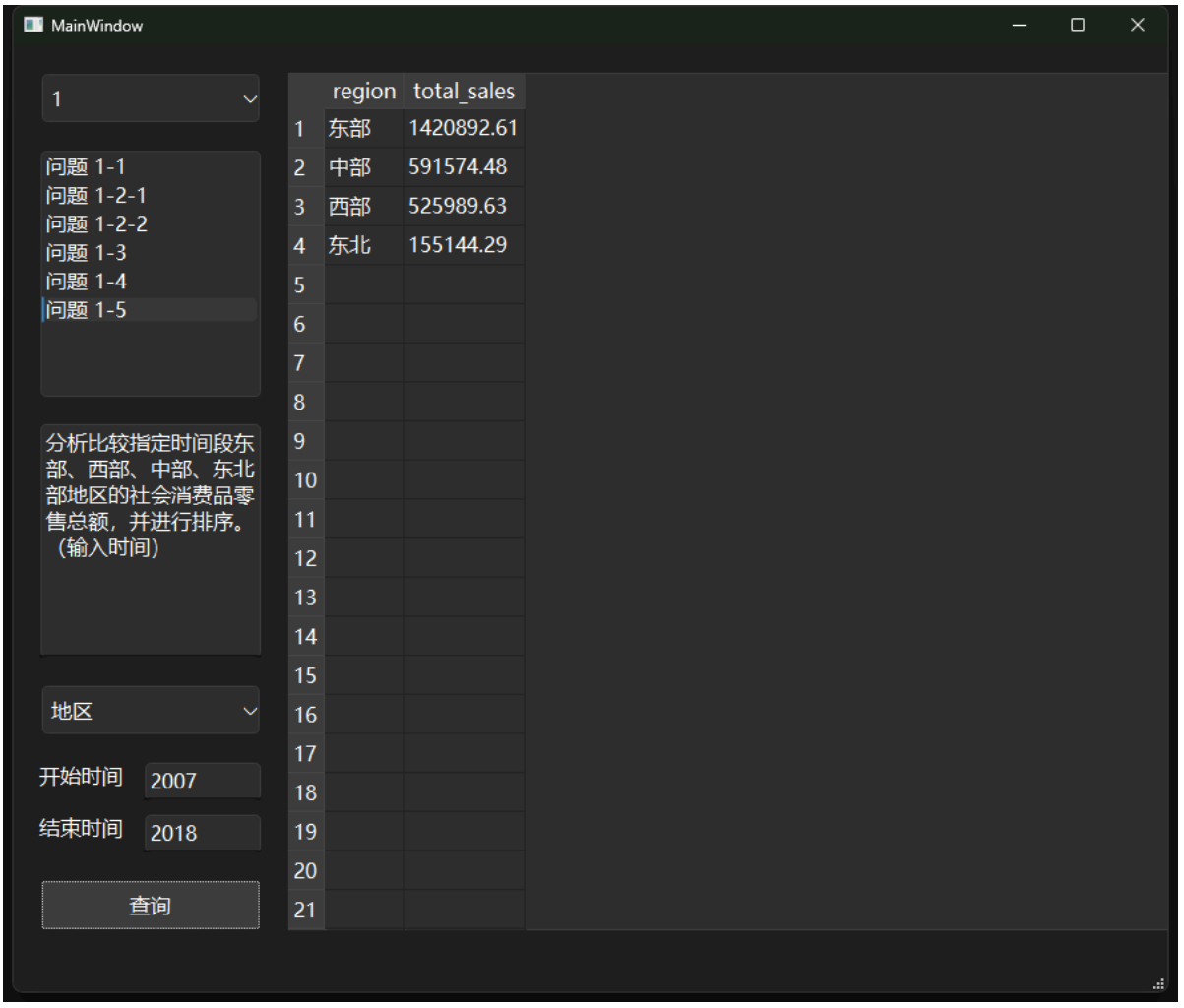
问题1-3运行结果如图所示



问题1-4运行结果如图所示

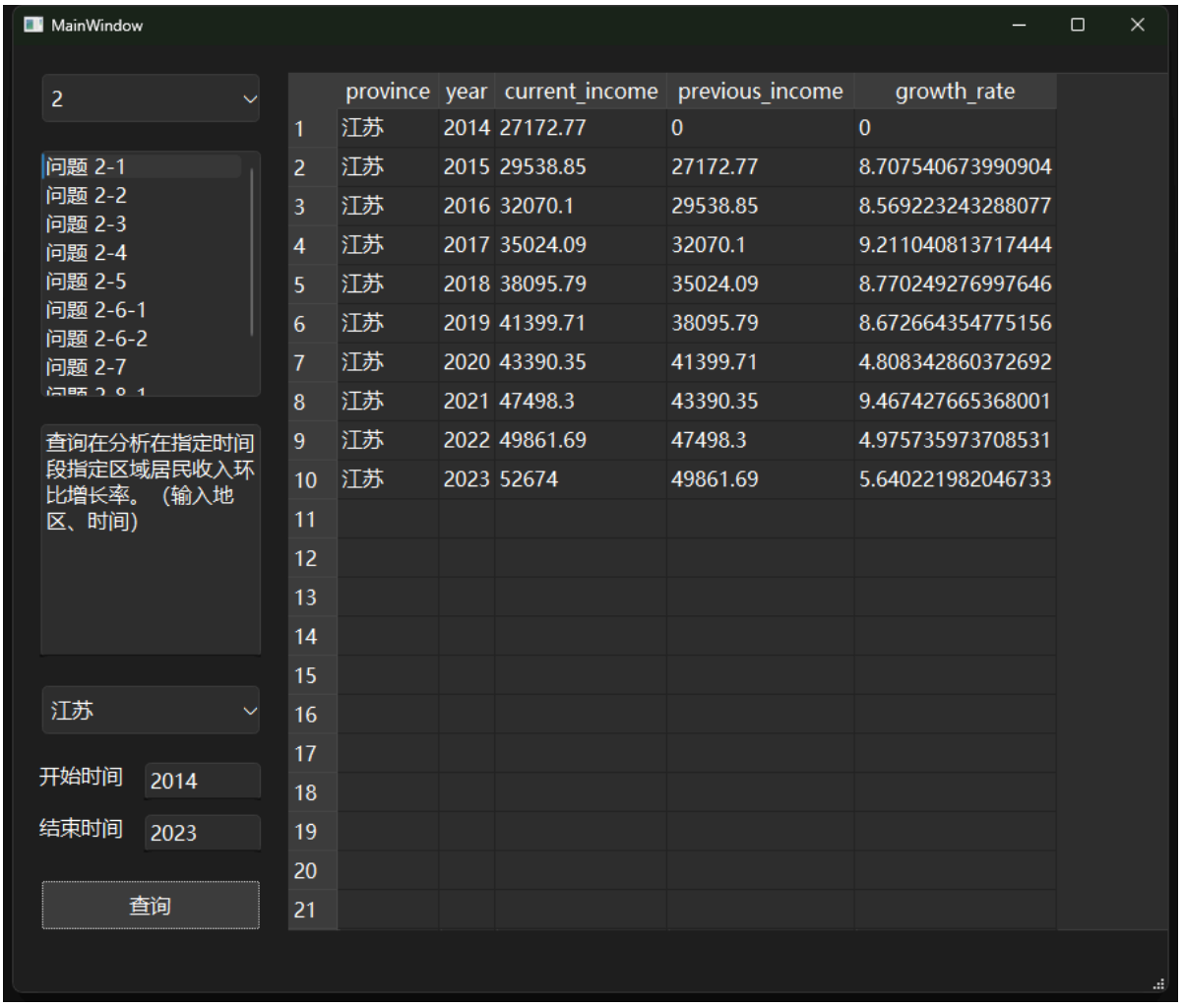


问题1-5运行结果如图所示

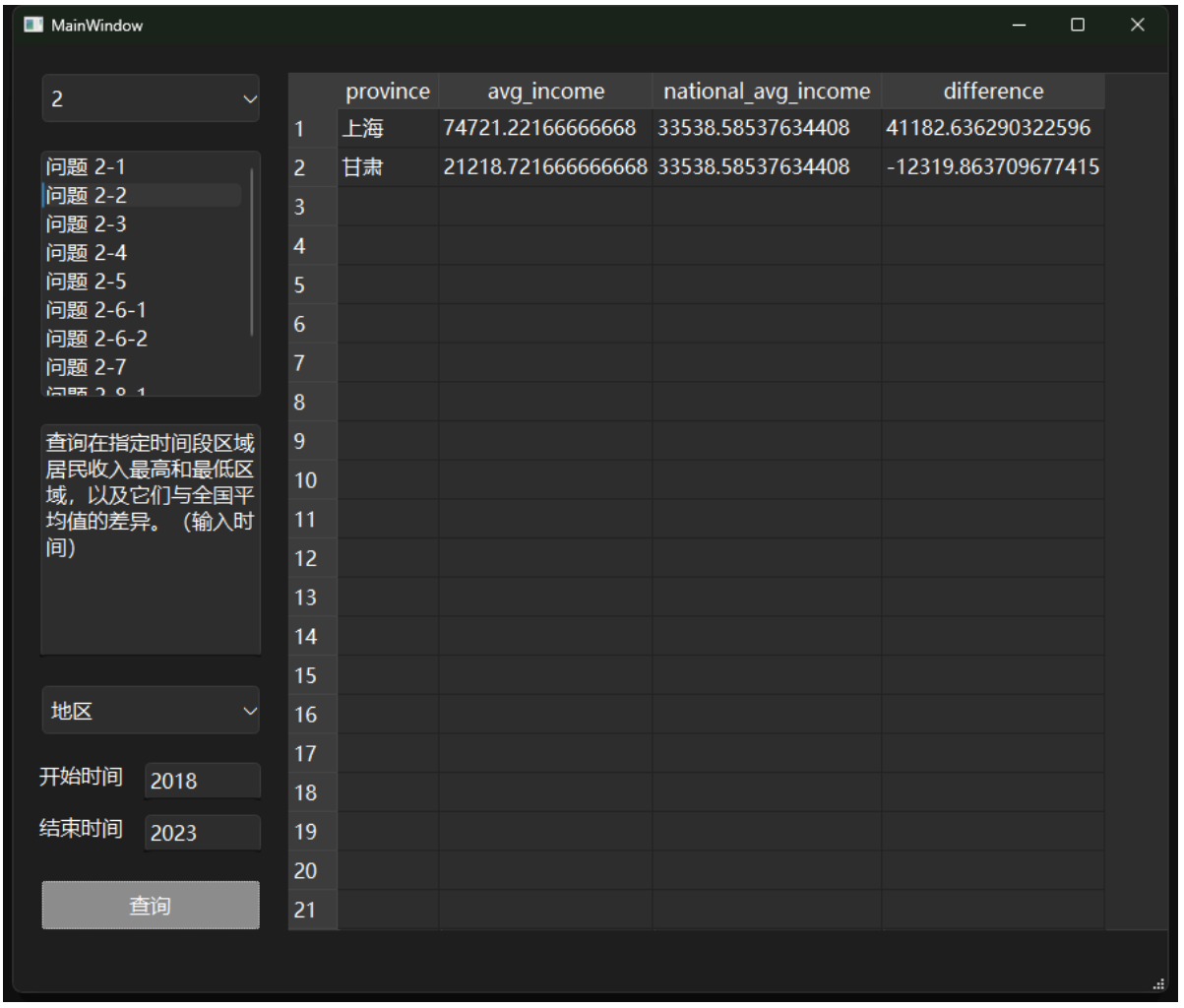


7.2 居民收入变化及区域差异

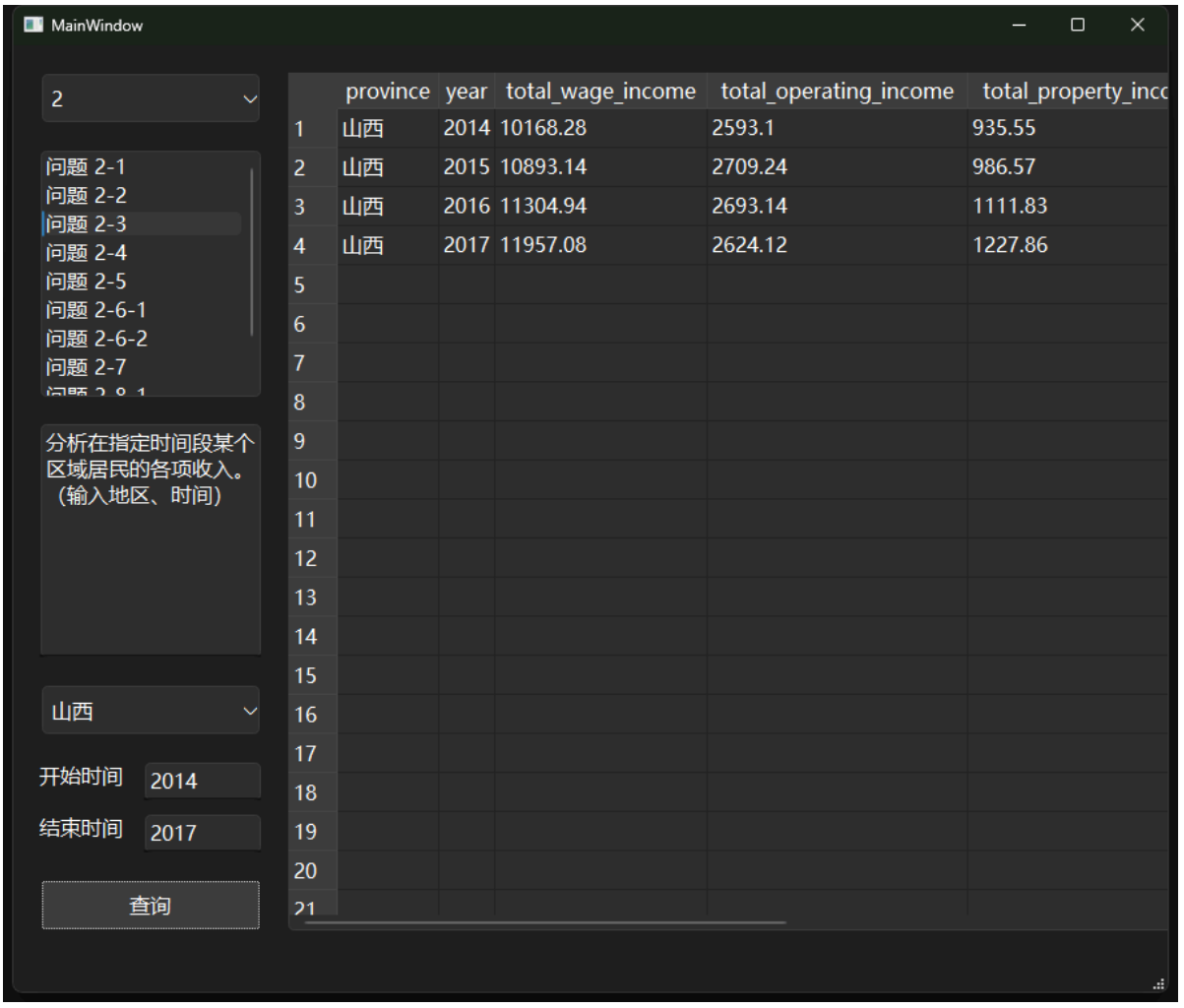
问题2-1运行结果如图所示



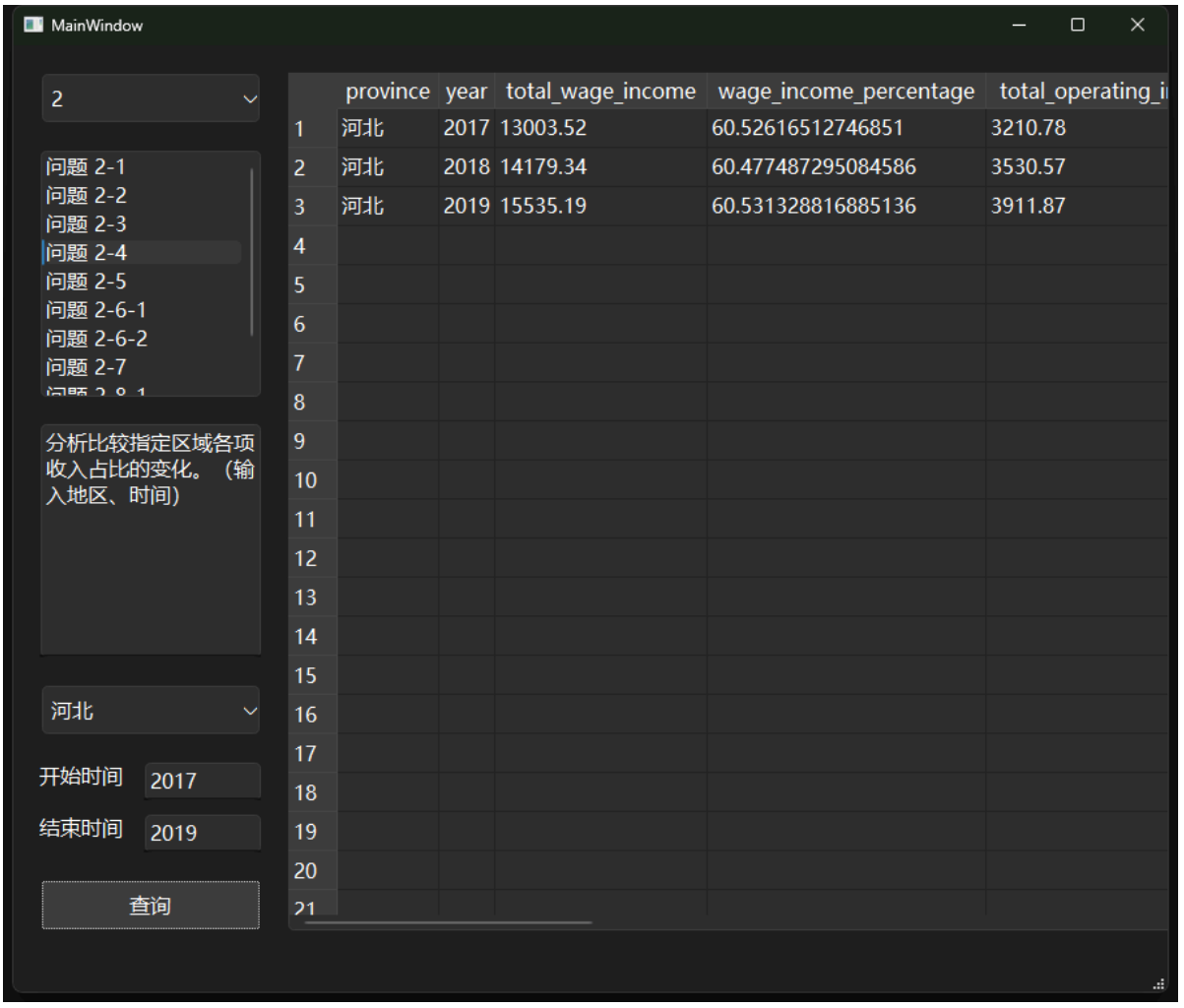
问题2-2运行结果如图所示



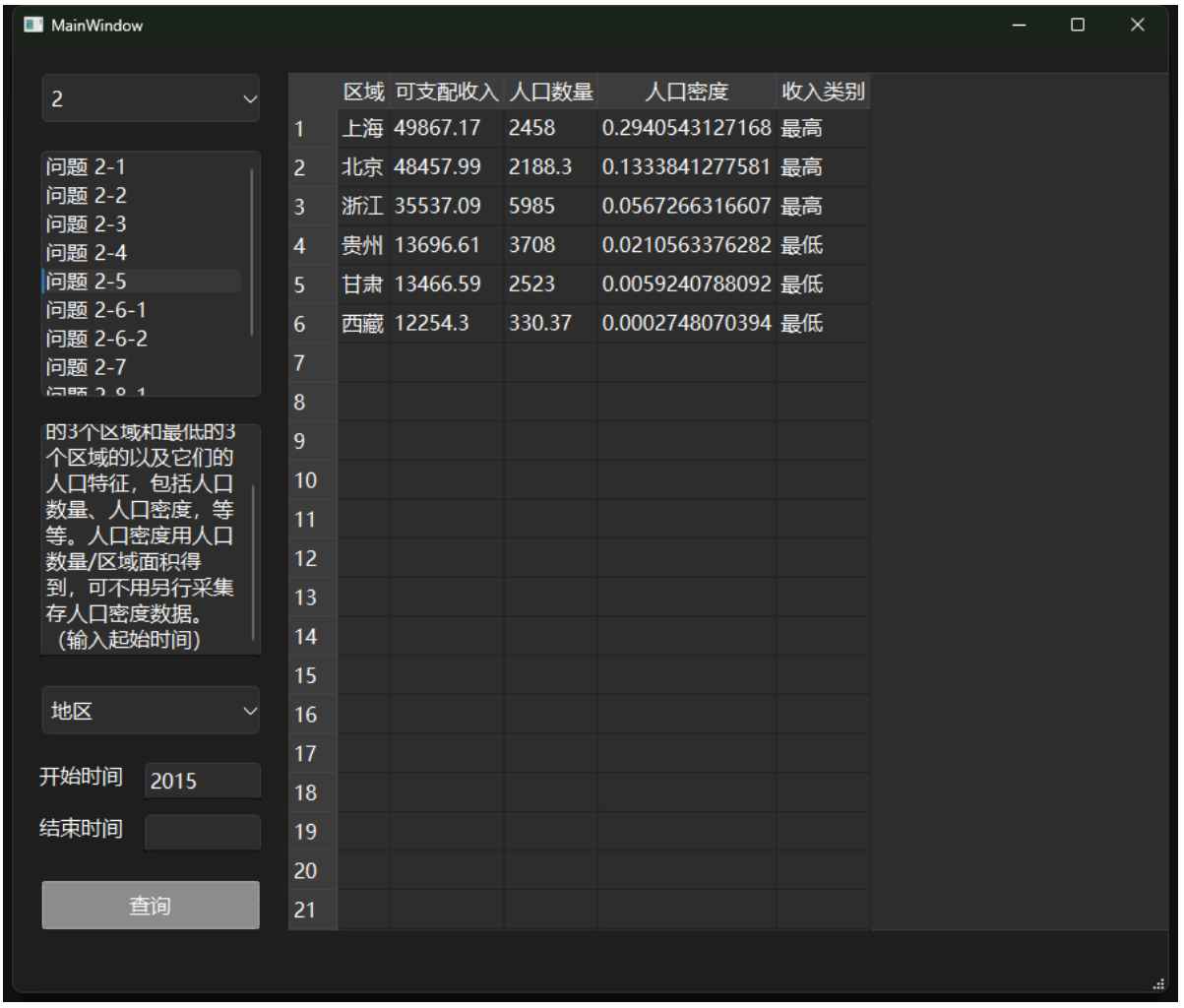
问题2-3运行结果如图所示



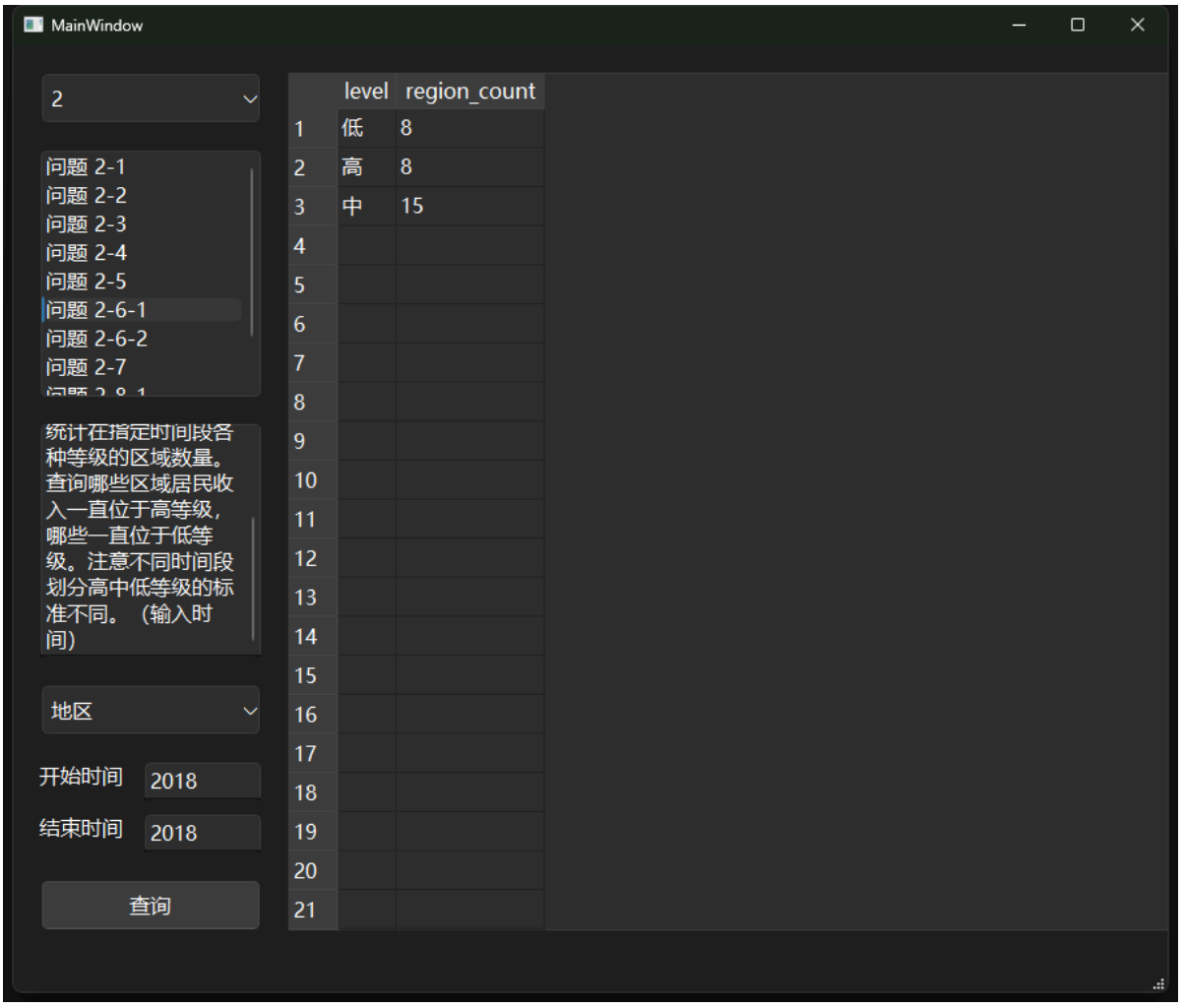
问题2-4运行结果如图所示



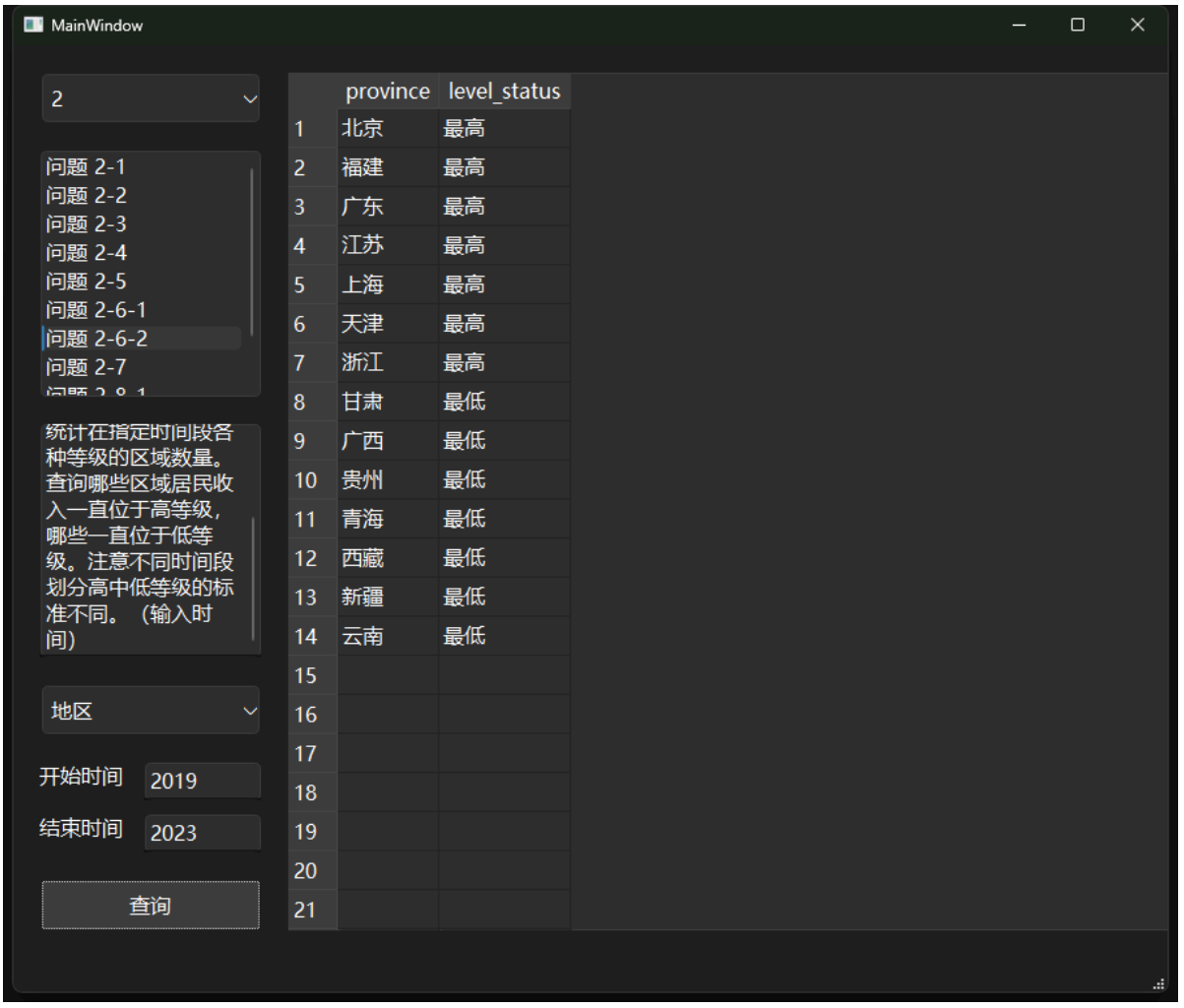
问题2-5运行结果如图所示



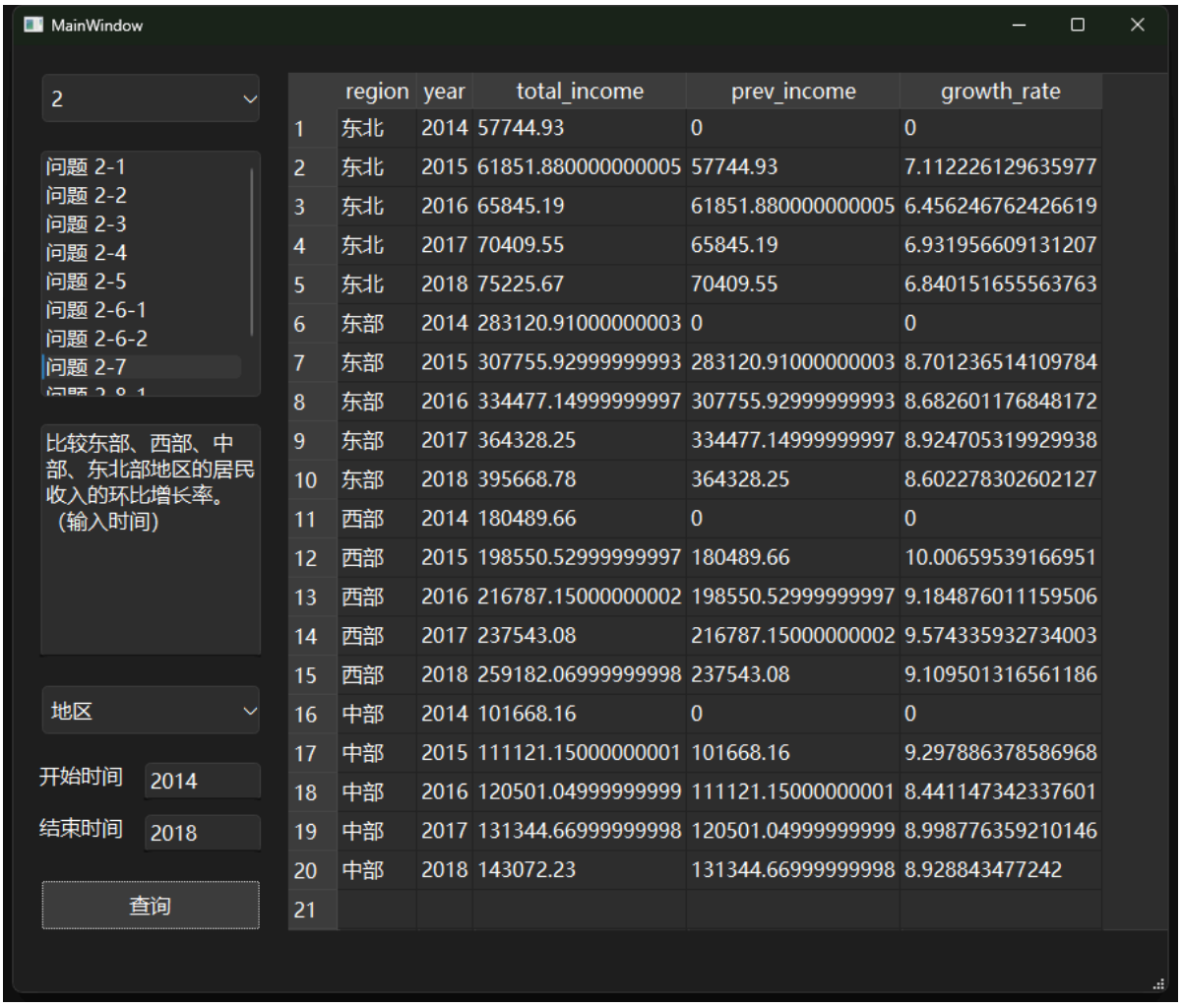
问题2-6-1运行结果如图所示



问题2-6-2运行结果如图所示

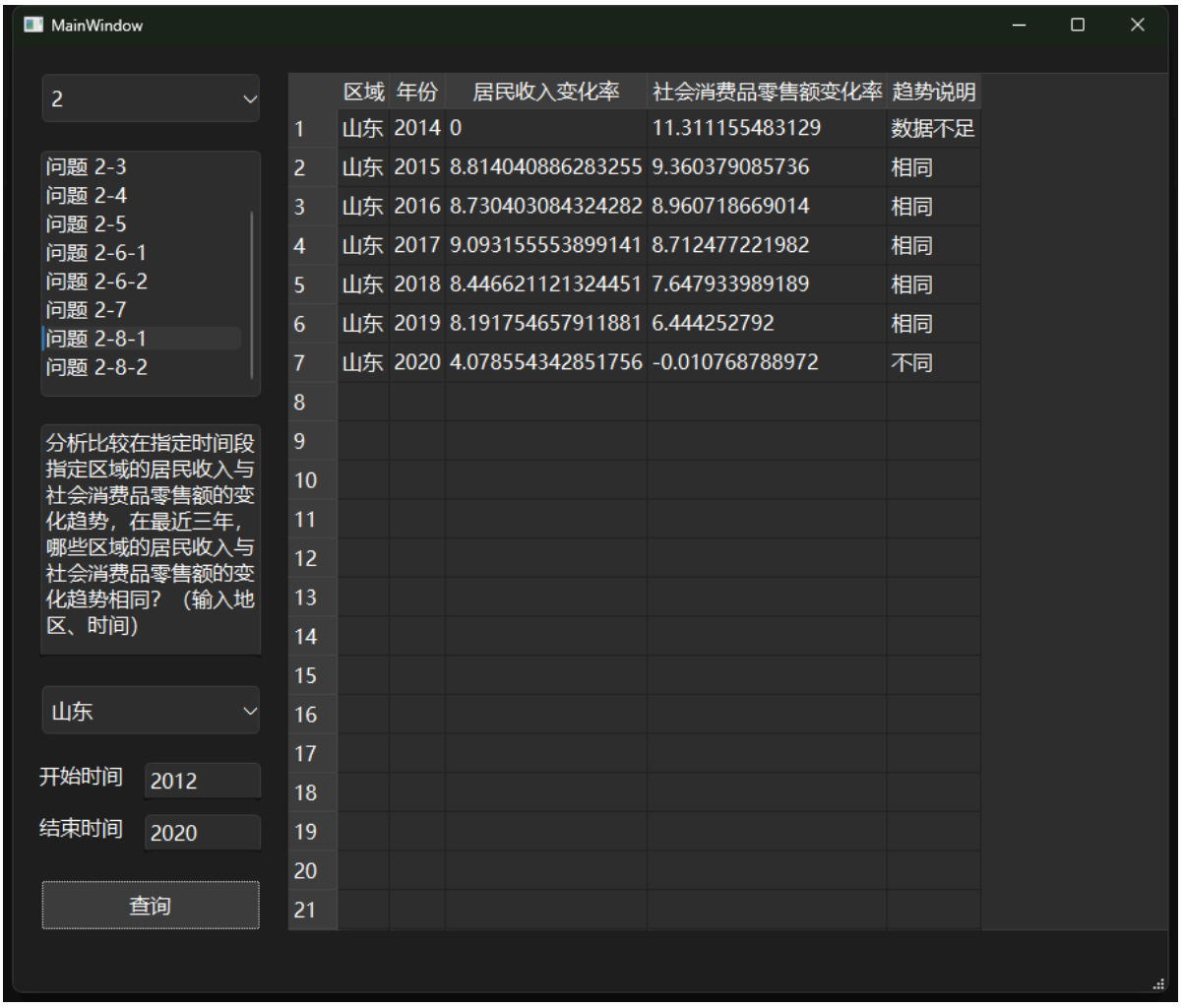


问题2-7运行结果如图所示

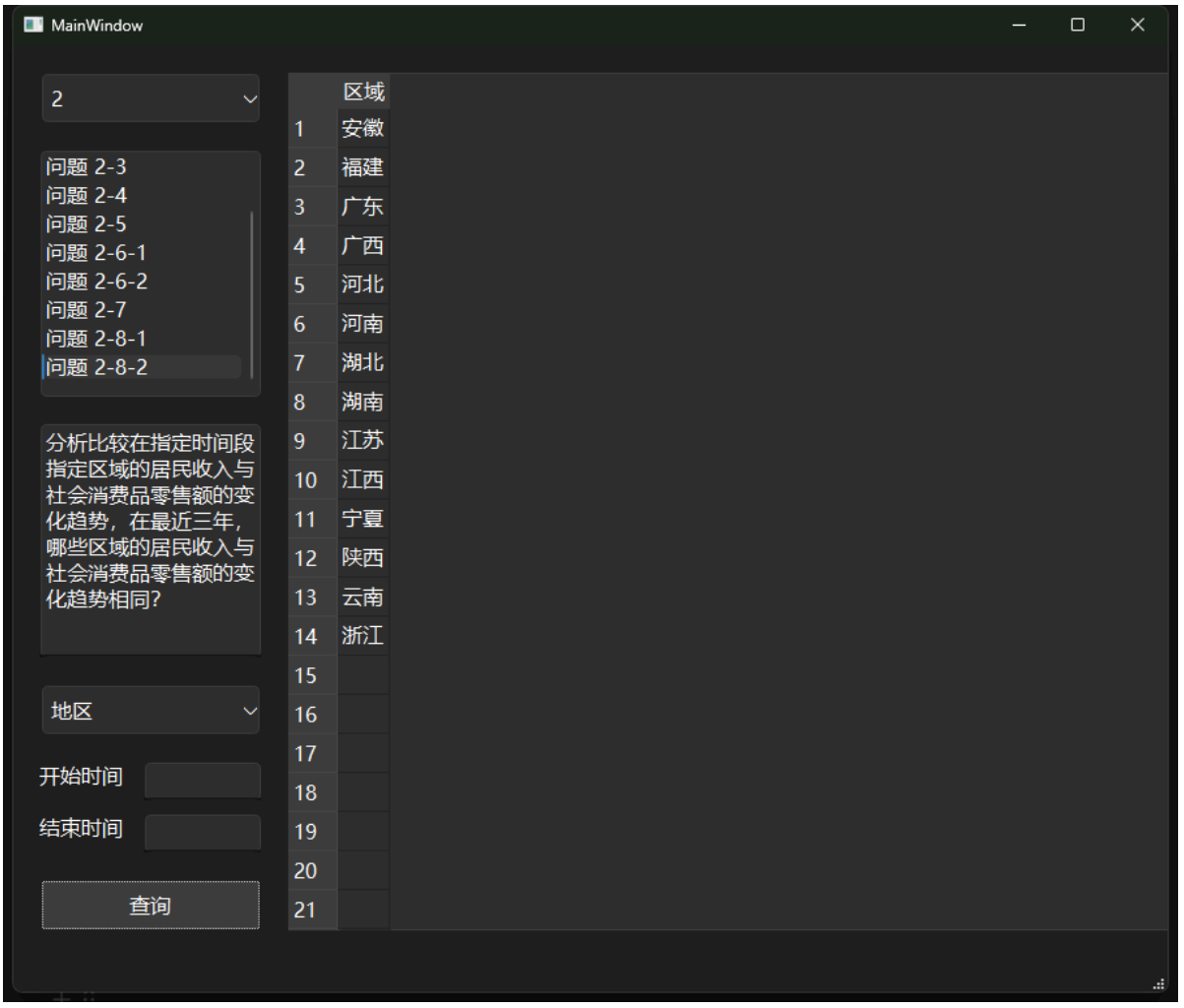


	region	year	total_income	prev_income	growth_rate
1	东北	2014	57744.93	0	0
2	东北	2015	61851.880000000005	57744.93	7.112226129635977
3	东北	2016	65845.19	61851.880000000005	6.456246762426619
4	东北	2017	70409.55	65845.19	6.931956609131207
5	东北	2018	75225.67	70409.55	6.840151655563763
6	东部	2014	283120.91000000003	0	0
7	东部	2015	307755.92999999993	283120.91000000003	8.701236514109784
8	东部	2016	334477.14999999997	307755.92999999993	8.682601176848172
9	东部	2017	364328.25	334477.14999999997	8.924705319929938
10	东部	2018	395668.78	364328.25	8.602278302602127
11	西部	2014	180489.66	0	0
12	西部	2015	198550.52999999997	180489.66	10.00659539166951
13	西部	2016	216787.15000000002	198550.52999999997	9.184876011159506
14	西部	2017	237543.08	216787.15000000002	9.574335932734003
15	西部	2018	259182.06999999998	237543.08	9.109501316561186
16	中部	2014	101668.16	0	0
17	中部	2015	111121.15000000001	101668.16	9.297886378586968
18	中部	2016	120501.04999999999	111121.15000000001	8.441147342337601
19	中部	2017	131344.66999999998	120501.04999999999	8.998776359210146
20	中部	2018	143072.23	131344.66999999998	8.928843477242
21					

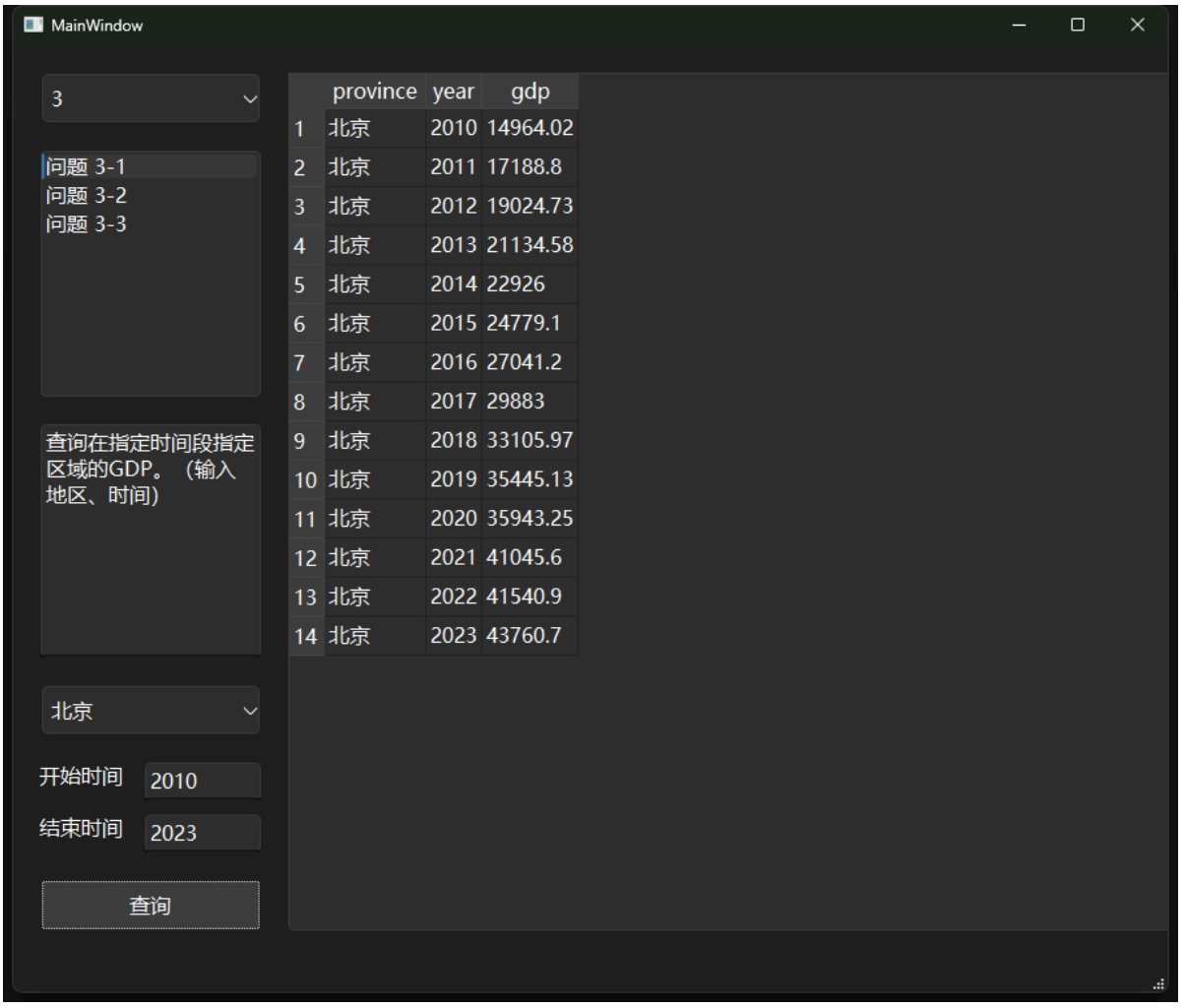
问题2-8-1运行结果如图所示



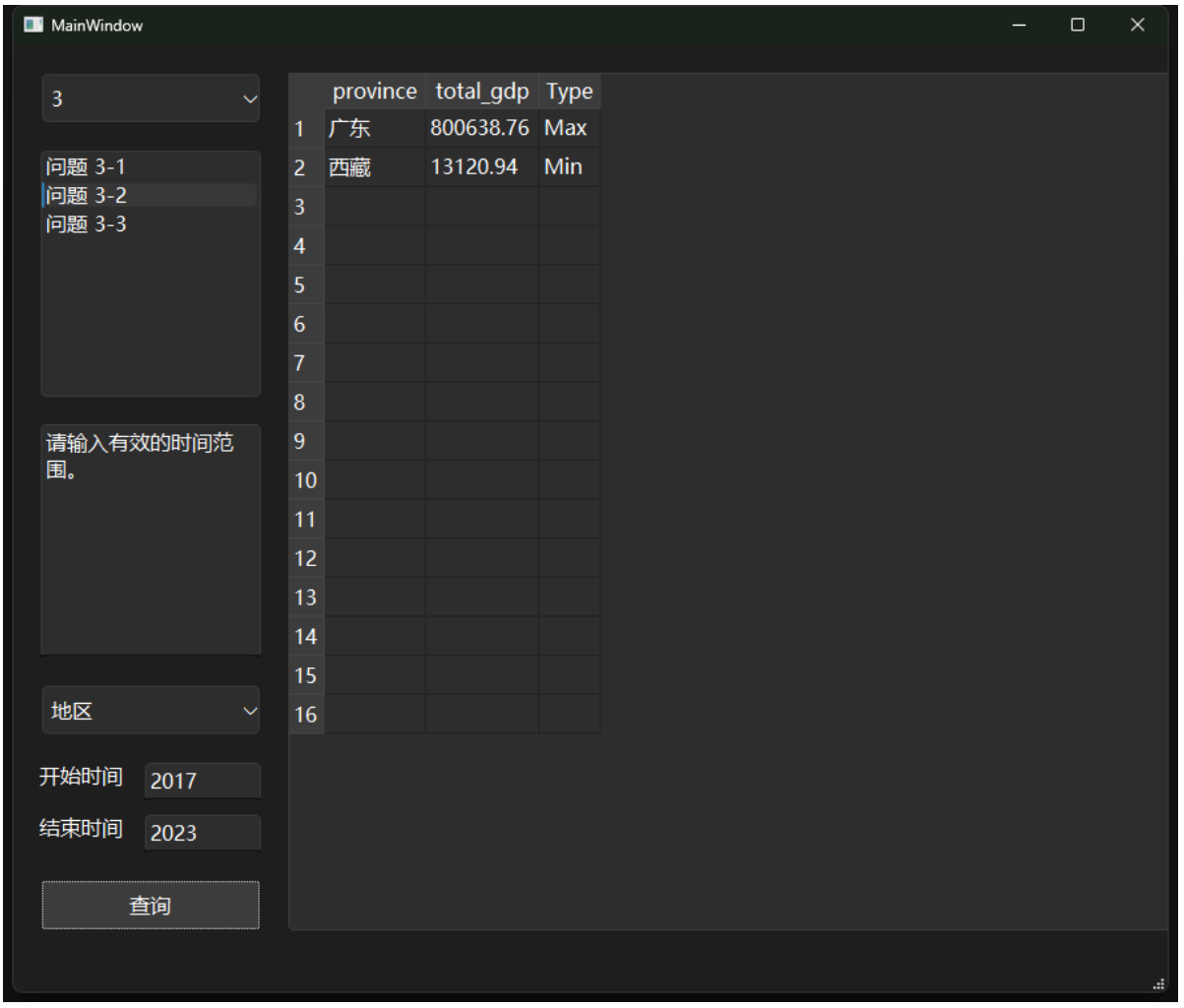
问题2-8-2运行结果如图所示



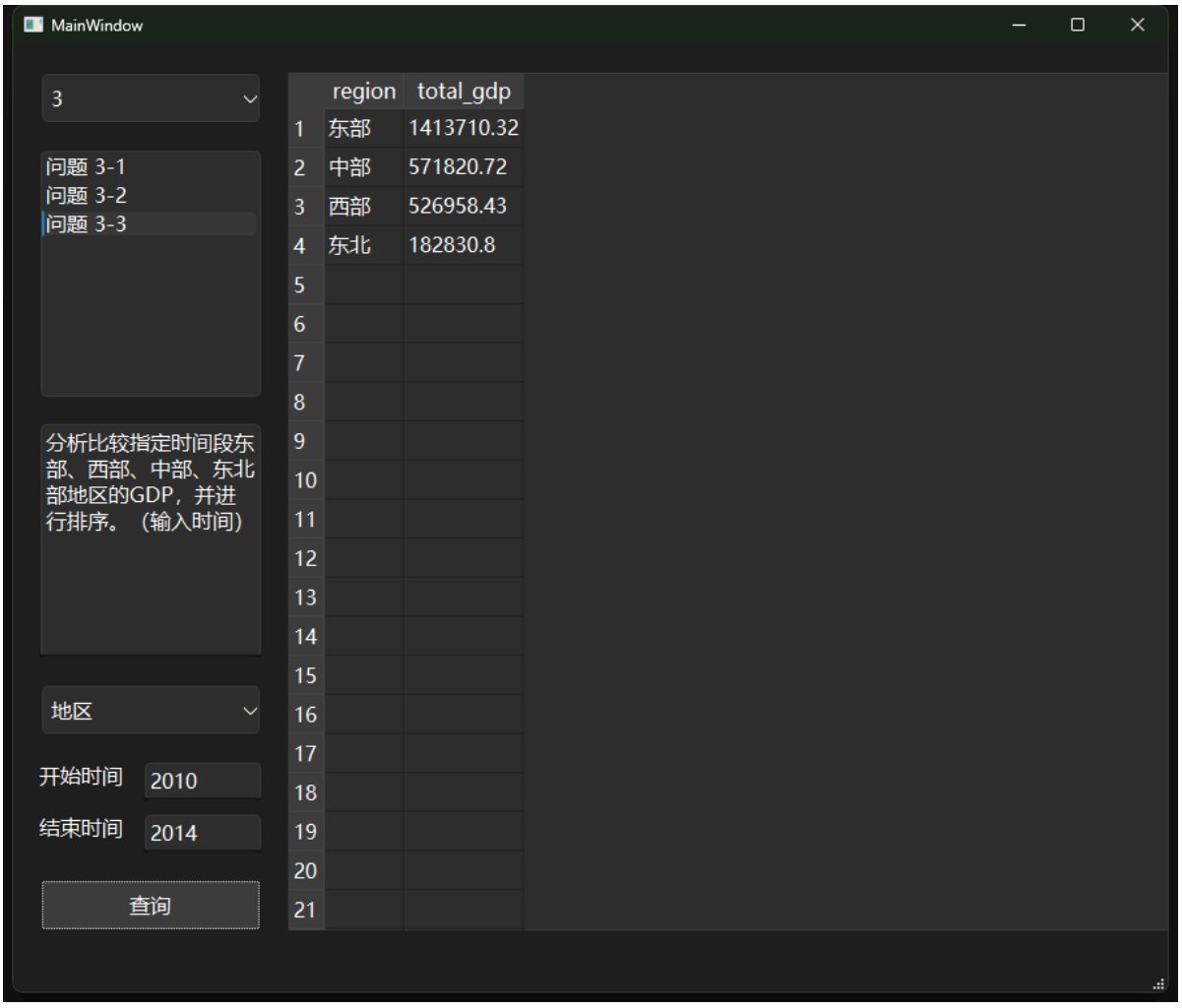
问题3-1运行结果如图所示



问题3-2运行结果如图所示



问题3-3运行结果如图所示


MainWindow
3
问题 3-1
问题 3-2
问题 3-3
分析比较指定时间段东部、西部、中部、东北部地区的GDP，并进行排序。（输入时间）
地区
开始时间
2010
结束时间
2014
查询
region
total_gdp
1 东部 1413710.32
2 中部 571820.72
3 西部 526958.43
4 东北 182830.8