

北京理工大学

数据库设计与开发

作业二 数据库设计

学 院：	计算机学院
专 业：	软件工程
班 级：	08012301 班
学生姓名：	蒋浩天
学 号：	1120231337
指导教师：	

2025 年 4 月 21 日

目 录

第 1 章 原创性说明	1
第 2 章 作业要求	1
第 3 章 设计学科表	2
3.1 数据结构设计	2
3.1.1 数据嵌套	2
3.1.2 可追溯的数据变更	3
3.2 示例数据说明	3
3.3 触发器更新逻辑设计	4
3.4 查询逻辑设计	6
第 4 章 设计籍贯表	8
第 5 章 设计学生表	10
5.1 查询学生信息	11
5.2 修改学生专业信息	13
第 6 章 设计成绩表	15
6.1 查询成绩表	16
6.2 实现交叉表查询	16
第 7 章 总结与体会	19

第 1 章 原创性说明

本作业系本人自主独立完成，特此申明！

第 2 章 作业要求

在 OpenGauss 数据库中, 完成以下内容.

- 设计实现“北京理工大学教学系统”, 要求能实现以下功能:
 - 处理学生专业变动
 - 学科数据字典设计
 - 处理编码长度改变
 - 如一级学科变成 3 位
 - 编码方式改变
 - 增加四级学科、将门类与一级学科合并
 - 代码有效期改变
 - 如某个二级学科从 2010 年 10 月 1 日开始不再使用
 - 代码被覆盖或替换
 - 如 081202 (计算机软件与理论) 改变为软件工程, 新增 081204 (计算机软
件与理论)
 - 要求保留原信息
 - 在引用数据字典的表中的信息, 要能查询到某人在 2010 年 10 月 1 日前的
学科为计算机软件与理论, 之后依然不变
 - 但籍贯会发生变化 (四川→重庆) (1997)
 - 进行学生成绩展示
 - 显示成绩单
 - 不使用 DBMS 提供的 `CROSS` 方法, 实现交叉表查询

第3章 设计学科表

3.1 数据结构设计

中华人民共和国学科分类指以国家标准为形式发布的学科分类方式及名称提法。学科分类采用树形结构,一级学科下设二级学科,二级学科下设三级学科,如此类推。每一级均有其编码与名称:

- 一级学科: 2 位数字编码, 如“01”表示“哲学”
- 二级学科: 4 位数字编码, 前 2 位为对应一级学科编码, 后 2 位为本学科编号
- 如此类推, 三级学科编码为 6 位数字, 四级学科编码为 8 位数字

此编码方式便于管理与扩展, 也方便检索上下级关系. 例如, “010102” 编码可直观地看出属于“01”哲学一级学科下的“01”哲学类二级学科下的“02”逻辑学。

但随着学科分类的不断发展, 学科编码也在不断变化。例如, 新设立的学科可能会使用已经消失的旧学科的编号, 三级学科可能会变更为二级学科等。

由于学科编号与所有大学毕业生的学位证书、学籍档案等密切相关, 在某次变更前毕业的学生, 其学位证书上可能会使用旧的学科编码, 而在变更后毕业的学生则使用新的学科编码。因此学科编码的变更需要有明确的可追溯性, 以便于在查询时能够准确地找到对应的学科信息。

考虑到学科编码结构的特殊性、对可追溯性的特殊要求, 将数据表设计如下:

```
CREATE TABLE subject (  
  id SERIAL NOT NULL PRIMARY KEY,  
  name VARCHAR(50),  
  parent_id VARCHAR(10),          -- 递归层级关系  
  self_id VARCHAR(10) NOT NULL,   -- 代码唯一标识  
  start_date DATE NOT NULL,       -- 代码生效时间  
  end_date DATE                   -- 代码失效时间 (NULL表示当前有效)  
);
```

下面对表中各处设计进行详细说明:

3.1.1 数据嵌套

在作业要求中, 学科表要能处理学科代码的变更, 如一级学科的编码长度变为 3 位。

作业二 数据库设计

由于学科分类的结构设计,在这种情况下,需要将该一级学科下的各级学科的编码都进行相应的调整,以确保数据的一致性和完整性。

因此,将学科信息的储存设计为嵌套结构,每个数据行都有 `parent_id` 列,指向上级学科's `id`。

这样使得一级学科下的二级学科、三级学科等都可以在同一表中进行存储。更新时,只需更新一级学科的编码,其下的各级学科编码可通过触发器进行自动更新,确保了数据的完整性和一致性。查询时,只需将 `parent_id` 列与 `self_id` 列进行连接,即可得到完整且准确的学科编码。

3.1.2 可追溯的数据变更

由于上述原因,学科编码的变更需要有明确的可追溯性,以便于在查询时能够准确地找到对应的学科信息。

因此,在数据表中设计了每条记录的生效时间和失效时间,失效时间设为 `NULL` 时表示截止当前有效;同时,所有的数据更改都以 `INSERT` 的方式记录,所有历史数据都以数据行的形式保存在表中,避免数据丢失。

在这样的设计下,需要进行查询时,可通过学生的毕业时间对数据行的生效时间和失效时间进行筛选,从而来确定使用的学科编码,以确保数据准确无误。

3.2 示例数据说明

在本次作业中,为方便实验与说明,设计以下测试数据:

```
INSERT INTO subject (name, parent_id, self_id, start_date, end_date) VALUES
('理学', NULL, '07', '1970-01-01', NULL),
('工学', NULL, '08', '1970-01-01', NULL),
('农学', NULL, '09', '1970-01-01', NULL),
('控制科学与工程', '08', '11', '1970-01-01', NULL),
('计算机科学与技术', '08', '12', '1970-01-01', NULL),
('建筑学', '08', '13', '1970-01-01', NULL),
('计算机系统结构', '0812', '01', '1970-01-01', NULL),
('计算机软件与理论', '0812', '02', '1970-01-01', NULL),
('计算机应用技术', '0812', '03', '1970-01-01', NULL),
('嵌入式软件', '081202', '01', '1970-01-01', NULL);
```

```
-- 假设2009年发生学科编号变更
INSERT INTO subject (name, parent_id, self_id, start_date, end_date) VALUES
    ('软件工程', '0812', '02', '2009-08-31', NULL),
    ('计算机软件与理论', '0812', '04', '2009-08-31', NULL);

-- 假设2014年发生大类编号变更
INSERT INTO subject (name, parent_id, self_id, start_date, end_date) VALUES
    ('理学', NULL, '007', '2014-01-01', NULL),
    ('工学', NULL, '008', '2014-01-01', NULL),
    ('农学', NULL, '009', '2014-01-01', NULL);

-- 假设2018年加入四级学科, 停用某些学科
INSERT INTO subject (name, parent_id, self_id, start_date, end_date) VALUES
    -- 停用这个学科
    ('计算机软件与理论', '00812', '04', '2018-01-01', '2018-01-01'),
    -- 加入这个四级学科
    ('嵌入式软件系统架构', '008120201', '01', '2018-01-01', NULL);
```

在测试数据中, 假设发生了以下变更事件:

1. 2009 年 8 月 31 日, 学科编号发生变更, **计算机软件与理论** 的编码由 *081202* 变更为 *081204*, 其原编号 *081202* 分配给了新增的学科 **软件工程**。
2. 2014 年 1 月 1 日, 学科大类编号发生变更, **理学** 的编码由 *07* 变更为 *007*, **工学** 的编码由 *08* 变更为 *008*, **农学** 的编码由 *09* 变更为 *009*。
3. 2018 年 1 月 1 日, **计算机软件与理论** 这个学科停用。新增了四级学科 **嵌入式软件系统架构**, 其编码为 *008120201*。

在下面的设计与测试中, 将以此数据为基础进行学科编码的查询与验证。

3.3 触发器更新逻辑设计

为了实现学科编码和过期信息的自动更新, 设计了一个触发器 `update_subject_id`, 用于在插入新数据时自动更新下级学科的编码和过期信息。

该触发器会在插入新数据后, 检查下级学科的编码, 并根据需要进行更新; 同时在插入新数据时, 将原先可能有的旧数据标记为过期。

作业二 数据库设计

```
DROP FUNCTION IF EXISTS subject_insert_trigger_func;
CREATE FUNCTION subject_insert_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    -- 更新parent_id
    INSERT INTO subject (name, parent_id, self_id, start_date, end_date)
    SELECT
        name,
        COALESCE(CAST(NEW.parent_id AS TEXT), '') || NEW.self_id,
        self_id,
        NEW.start_date,
        NULL
    FROM subject_full_id(NEW.start_date)
    WHERE parent_id IN (
        SELECT full_id
        FROM subject_full_id(NEW.start_date)
        WHERE name = NEW.name
    );

    -- 更新end_date
    UPDATE subject
    SET end_date = NEW.start_date
    WHERE id IN (
        SELECT id
        FROM subject_full_id(NEW.start_date)
        WHERE (
            (self_id = NEW.self_id AND parent_id = NEW.parent_id) OR
            name = NEW.name
        ) AND
        end_date IS NULL
    );

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

在这个触发器对学科编码更改的处理逻辑中，由于对数据表进行了下一级学科的插入操作，于是触发器会被递归触发，以确保所有下级学科的编码都能得到更新。

3.4 查询逻辑设计

为了方便针对特定时间节点进行学科编码的查询，设计了一个函数 `subject_full_id`，用于根据时间节点查询拼接后的完整学科编码。该函数将根据输入的时间参数，结合 `start_date` 和 `end_date` 字段，返回对应的学科编码。

```
DROP FUNCTION IF EXISTS subject_full_id;
CREATE FUNCTION subject_full_id(target_date DATE)
RETURNS TABLE(
    id INTEGER,
    name VARCHAR(50),
    full_id TEXT,
    parent_id VARCHAR(10),
    self_id VARCHAR(10),
    start_date DATE,
    end_date DATE
) AS $$
BEGIN
    RETURN QUERY SELECT
        subject.id AS id,
        subject.name AS name,
        -- 拼接完整的学科编码
        COALESCE(subject.parent_id, '') || subject.self_id AS full_id,
        subject.parent_id AS parent_id,
        subject.self_id AS self_id,
        subject.start_date AS start_date,
        subject.end_date AS end_date
    FROM subject
    WHERE
        -- 根据时间节点筛选数据
        subject.start_date ≤ target_date AND
        (subject.end_date IS NULL OR subject.end_date > target_date);
END;
$$ LANGUAGE plpgsql;
```

创建这个函数后，可以通过调用 `subject_full_id` 函数来查询特定时间节点的学科编码。

例如，查询 2010 年 8 月 31 日的学科编码，可以使用以下 SQL 语句：

作业二 数据库设计

```
SELECT * FROM subject_full_id('2010-08-31');
```

查询结果如下：

Figure 3-1 shows the OpenGauss database interface with the query results for the date 2010-08-31. The results pane displays a table with 12 rows and 8 columns. The columns are: #, id, name, rmc_full_id, rmc_parent_id, rmc_self_id, start_date, and end_date. The data is as follows:

#	id	name	rmc_full_id	rmc_parent_id	rmc_self_id	start_date	end_date
1	1	理学	07	[NULL]	07	1970-01-01	2014-01-01
2	2	工学	08	[NULL]	08	1970-01-01	2014-01-01
3	3	农学	09	[NULL]	09	1970-01-01	2014-01-01
4	4	控制科学与工程	0811	08	11	1970-01-01	2014-01-01
5	5	计算机科学与技术	0812	08	12	1970-01-01	2014-01-01
6	6	建筑学	0813	08	13	1970-01-01	2014-01-01
7	7	计算机系统结构	081201	0812	01	1970-01-01	2014-01-01
8	9	计算机应用技术	081203	0812	03	1970-01-01	2014-01-01
9	10	嵌入式软件	08120201	081202	01	1970-01-01	2014-01-01
10	11	软件工程	081202	0812	02	2009-08-31	2014-01-01
11	12	计算机软件与理论	081204	0812	04	2009-08-31	2014-01-01

图 3-1 2010 年 8 月 31 日的学科编码查询结果

再进行 2016 年 8 月 31 日的学科编码查询，结果如下：

Figure 3-2 shows the OpenGauss database interface with the query results for the date 2016-08-31. The results pane displays a table with 13 rows and 8 columns. The columns are: #, id, name, rmc_full_id, rmc_parent_id, rmc_self_id, start_date, and end_date. The data is as follows:

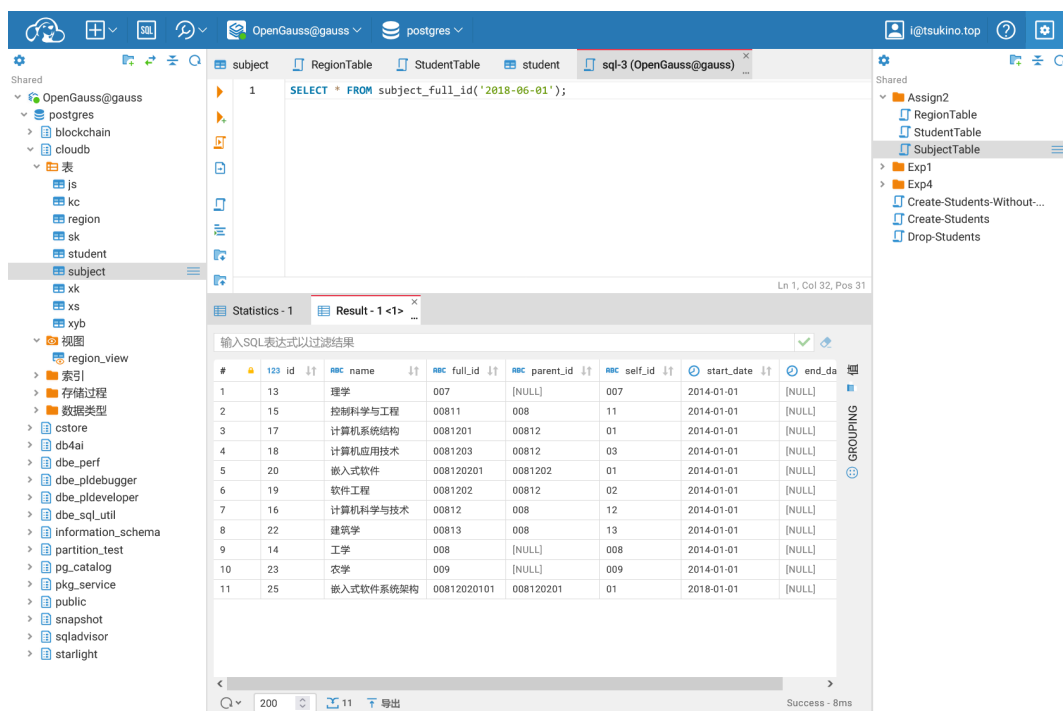
#	id	name	rmc_full_id	rmc_parent_id	rmc_self_id	start_date	end_date
1	13	理学	007	[NULL]	007	2014-01-01	[NULL]
2	14	工学	008	[NULL]	008	2014-01-01	[NULL]
3	15	控制科学与工程	00811	008	11	2014-01-01	[NULL]
4	16	计算机科学与技术	00812	008	12	2014-01-01	[NULL]
5	17	计算机系统结构	0081201	00812	01	2014-01-01	[NULL]
6	18	计算机应用技术	0081203	00812	03	2014-01-01	[NULL]
7	19	软件工程	0081202	00812	02	2014-01-01	[NULL]
8	20	嵌入式软件	008120201	0081202	01	2014-01-01	[NULL]
9	21	计算机软件与理论	0081204	00812	04	2014-01-01	[NULL]
10	22	建筑学	00813	008	13	2014-01-01	[NULL]
11	23	农学	009	[NULL]	009	2014-01-01	[NULL]

图 3-2 2016 年 8 月 31 日的学科编码查询结果

作业二 数据库设计

可以看见，在我们的学科编号数据中，由于 2014 年学科大类编号从 2 位变为了 3 位，导致了在后续的时间点进行查询时，各个学科的编号和生效时间、失效时间都发生了相对应的变化。

再对 2018 年 6 月 1 日的学科编码进行查询，结果如下：



The screenshot shows a database management tool interface. The main window displays a query result for the 'subject' table, filtered by the date '2018-06-01'. The query is: `SELECT * FROM subject_full_id('2018-06-01');`. The result is shown in a table with 11 rows and 8 columns. The columns are: #, id, nrc, name, nrc_full_id, nrc_parent_id, nrc_self_id, start_date, and end_date. The data shows a hierarchy of subjects, with some subjects having parent and self IDs. The start_date for all subjects is 2014-01-01, and the end_date is NULL.

#	id	nrc	name	nrc_full_id	nrc_parent_id	nrc_self_id	start_date	end_date
1	13	理学	007	[NULL]	007		2014-01-01	[NULL]
2	15	控制科学与工程	00811	008	11		2014-01-01	[NULL]
3	17	计算机系统结构	0081201	00812	01		2014-01-01	[NULL]
4	18	计算机应用技术	0081203	00812	03		2014-01-01	[NULL]
5	20	嵌入式软件	008120201	0081202	01		2014-01-01	[NULL]
6	19	软件工程	0081202	00812	02		2014-01-01	[NULL]
7	16	计算机科学与技术	00812	008	12		2014-01-01	[NULL]
8	22	建筑学	00813	008	13		2014-01-01	[NULL]
9	14	工学	008	[NULL]	008		2014-01-01	[NULL]
10	23	农学	009	[NULL]	009		2014-01-01	[NULL]
11	25	嵌入式软件系统架构	00812020101	008120201	01		2018-01-01	[NULL]

图 3-3 2018 年 6 月 1 日的学科编码查询结果

可以看见，由于 2018 年 1 月 1 日发生的变化，在后续的时间点进行查询时，各个学科的数据行也都发生了相对应的变化。

第 4 章 设计籍贯表

储存学生个人信息时，籍贯信息是经常设计的一部分。但由于我国在过去数十年间，对行政区划进行了大量的调整，导致了籍贯信息的设计需要不断更新和调整，以确保其准确性和有效性。

为确保籍贯信息的准确性，设计了籍贯信息表，以便及时反映行政区划的变化。该表将包含各个地区的名称、编码、以及地区的变更信息，以支持对籍贯信息的动态管理和查询。

数据表的设计如下：

作业二 数据库设计

```
CREATE TABLE region (  
  id VARCHAR(6) NOT NULL PRIMARY KEY,  
  name VARCHAR(40) NOT NULL,  
  -- 变更后的编号  
  replaced_by VARCHAR(6),  
  UNIQUE(id)  
);
```

其中，`id` 列储存的是符合我国居民身份证号中地区信息编码规则的六位编码，如：`110101` 表示北京市东城区。`replaced_by` 列储存的是变更后的编号，例如，在重庆市从四川省脱离，成为直辖市时，原编码 `510200` 的 `replaced_by` 列将被更新为新的编码 `500100`。此时进行查询，不论是使用 `510200` 还是 `500100`，查询到的地区名称都将是 **重庆市** 而非过时的 **四川省重庆市**。

插入的数据如下：

```
INSERT INTO region (id, name, replaced_by) VALUES  
('110000', '北京市', NULL),  
('110100', '北京市市辖区', NULL),  
('110200', '北京市县', '110100'),    -- 已撤销，改为市辖区  
('120000', '天津市', NULL),  
('120100', '天津市市辖区', NULL),  
('120200', '天津市县', '120100'),    -- 已撤销，改为市辖区  
.....  
('654200', '新疆维吾尔自治区塔城地区', NULL),  
('654300', '新疆维吾尔自治区阿勒泰地区', NULL),  
('659000', '新疆维吾尔自治区直辖县级行政单位', NULL);
```

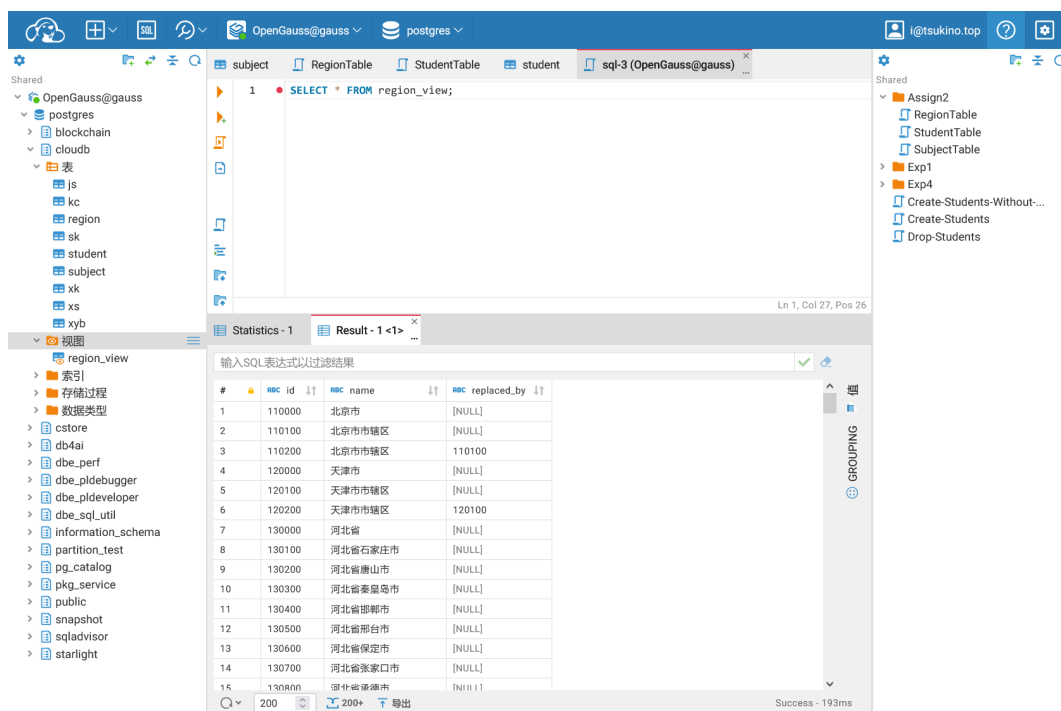
为方便查询，可以创建视图以简化查询过程。创建视图的 SQL 语句如下：

```
CREATE VIEW region_view AS  
SELECT  
  l.id,  
  COALESCE(r.name, l.name) AS name,  
  l.replaced_by  
FROM region l  
LEFT JOIN region r ON l.replaced_by = r.id;
```

作业二 数据库设计

可以看到，视图 `region_view` 通过 `JOIN` 进行表的连接，显示所有地区的现行名称和编号。通过使用 `COALESCE` 函数，可以确保即使在没有变更的情况下，查询结果也能正确显示地区名称。

对视图进行查询，结果如下：



#	rrc id	rrc name	rrc replaced_by
1	110000	北京市	[NULL]
2	110100	北京市市辖区	[NULL]
3	110200	北京市市辖区	110100
4	120000	天津市	[NULL]
5	120100	天津市市辖区	[NULL]
6	120200	天津市市辖区	120100
7	130000	河北省	[NULL]
8	130100	河北省石家庄市	[NULL]
9	130200	河北省唐山市	[NULL]
10	130300	河北省秦皇岛市	[NULL]
11	130400	河北省邯郸市	[NULL]
12	130500	河北省邢台市	[NULL]
13	130600	河北省保定市	[NULL]
14	130700	河北省张家口市	[NULL]
15	130800	河北省承德市	[NULL]

图 4-1 籍贯视图查询结果

第 5 章 设计学生表

为了储存学生信息，学生表格设计如下：

```
CREATE TABLE student (  
  id VARCHAR(20) NOT NULL PRIMARY KEY,  
  name VARCHAR(20) NOT NULL,  
  graduate_date DATE,  
  subject_id VARCHAR(20),  
  UNIQUE(id),  
  region VARCHAR(6),  
  FOREIGN KEY (region) REFERENCES region(id)  
);
```

插入以下测试用例数据：

作业二 数据库设计

```
INSERT INTO student (id, name, graduate_date, subject_id, region) VALUES
-- 2009年的学生
('1120090001', '张三', '2013-06-30', '081203', '110100'),
('1120090002', '李四', '2013-06-30', '081202', '510200'), -- 原四川省重庆市
('1120090003', '王五', '2013-06-30', '081201', '420400'), -- 原湖北省沙市市
('1120090004', '赵六', '2013-06-30', '081202', '542300'), -- 原西藏自治区日喀则地区
-- 2000年的学生
-- 毕业时软件工程尚不是二级专业
('1120000001', '刘七', '2004-06-30', '081203', '110100'),
('1120000002', '朱八', '2004-06-30', '081202', '510200'),
('1120000003', '徐九', '2004-06-30', '081201', '420400'),
('1120000004', '吴十', '2004-06-30', '081202', '542300'),
-- 2015年的学生
-- 毕业时大类编号已变更
('1120150001', '钱十一', '2019-06-30', '0081203', '110100'),
('1120150002', '孙十二', '2019-06-30', '0081201', '500100'),
('1120150003', '周十三', '2019-06-30', '0081201', '421000'),
('1120150004', '郑十四', '2019-06-30', '0081203', '540200'),
-- 在校生
('1120240001', '林十五', NULL, '0081201', '310100');
```

5.1 查询学生信息

为方便查询全部学生的信息，设计了查询函数如下：

```
CREATE FUNCTION get_all_students()
RETURNS TABLE (
    id VARCHAR(20),
    name VARCHAR(20),
    graduate_date DATE,
    region_name VARCHAR(40),
    region_id VARCHAR(6),
    subject_name VARCHAR(50),
    subject_id VARCHAR(20) -- 修改 subject_id 的数据类型为 VARCHAR(20)
) AS $$
DECLARE
```

作业二 数据库设计

```
student_row RECORD;      -- 保存正在处理的 student 行记录
rname TEXT;              -- 存储查询到的区域名称
rid VARCHAR(6);          -- 存储查询到的区域ID
subj_name VARCHAR(50);   -- 存储查询到的科目名称
BEGIN
    -- 遍历 student 表的每一行数据
    FOR student_row IN
        SELECT s.id, s.name, s.graduate_date, s.region, s.subject_id
        FROM student s
    LOOP
        -- 计算 region_name 和 region_id (使用更清晰的变量命名避免歧义)
        SELECT r.name, r.id
        INTO rname, rid
        FROM region_view r
        WHERE r.id = student_row.region;

        -- 计算 subject_name (模拟 LATERAL 子查询)
        SELECT sub.name
        INTO subj_name
        FROM subject_full_id(COALESCE(student_row.graduate_date, CURRENT_DATE))
        sub
        WHERE sub.full_id = student_row.subject_id;

        -- 将数据赋值给 OUT 参数
        id := student_row.id;          -- 学生ID
        name := student_row.name;      -- 学生姓名
        graduate_date := student_row.graduate_date; -- 毕业日期
        region_name := rname;          -- 区域名称
        region_id := rid;              -- 区域ID
        subject_name := subj_name;     -- 科目名称
        subject_id := student_row.subject_id; -- 科目ID

        -- 返回当前处理行结果
        RETURN NEXT;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

执行查询函数时，使用如下语句：

作业二 数据库设计

```
SELECT * FROM get_all_students();
```

查询结果如下：

表 5-1 查询所有学生的结果

id	name	graduate date	region name	region id	subject name	subject id
1120090001	张三	2013-06-30	北京市市辖区	110100	计算机应用技术	081203
1120090002	李四	2013-06-30	重庆市	510200	软件工程	081202
1120090003	王五	2013-06-30	湖北省荆州市	420400	计算机系统结构	081201
1120090004	赵六	2013-06-30	西藏自治区日喀则市	542300	软件工程	081202
1120000001	刘七	2004-06-30	北京市市辖区	110100	计算机应用技术	081203
1120000002	朱八	2004-06-30	重庆市	510200	计算机软件与理论	081202
1120000003	徐九	2004-06-30	湖北省荆州市	420400	计算机系统结构	081201
1120000004	吴十	2004-06-30	西藏自治区日喀则市	542300	计算机软件与理论	081202
1120150001	钱十一	2019-06-30	北京市市辖区	110100	计算机应用技术	0081203
1120150002	孙十二	2019-06-30	重庆市市辖区	500100	计算机系统结构	0081201
1120150003	周十三	2019-06-30	湖北省荆州市	421000	计算机系统结构	0081201
1120150004	郑十四	2019-06-30	西藏自治区日喀则市	540200	计算机应用技术	0081203
1120240001	林十五	NULL	上海市市辖区	310100	计算机系统结构	0081201

可以看见，查询结果中根据学生的毕业日期，查询到了毕业时学科编号所对应的学科名称；而学生的籍贯信息显示的是当前的区域名称。

例如，学生朱八于2004年毕业时，其学科编号081202所对应的学科为计算机
软件与理论而不是软件工程；虽然其籍贯信息为旧四川省重庆市的编号，显示其籍
贯仍为重庆市。

5.2 修改学生专业信息

由于只有在校生有可能修改其专业信息，因此设计了如下函数：

```
DROP FUNCTION IF EXISTS update_student_subject;
CREATE FUNCTION update_student_subject(
    target_id VARCHAR(20),
    new_subject_id VARCHAR(20)
) RETURNS TABLE (
    学号 VARCHAR(20),
    姓名 VARCHAR(20),
```

作业二 数据库设计

```
    毕业时间 DATE,  
    地区编号 VARCHAR(6),  
    专业编号 VARCHAR(20)  
  ) AS $$  
BEGIN  
    UPDATE student  
    SET subject_id = new_subject_id  
    WHERE id = target_id AND graduate_date IS NULL;  
  
    RETURN QUERY SELECT * FROM student WHERE id = target_id;  
END;  
$$ LANGUAGE plpgsql;
```

添加函数之后，可以使用如下语句修改学生 林十五 的专业信息为嵌入式软件系统架构：

```
SELECT update_student_subject('1120240001', '008120201');
```

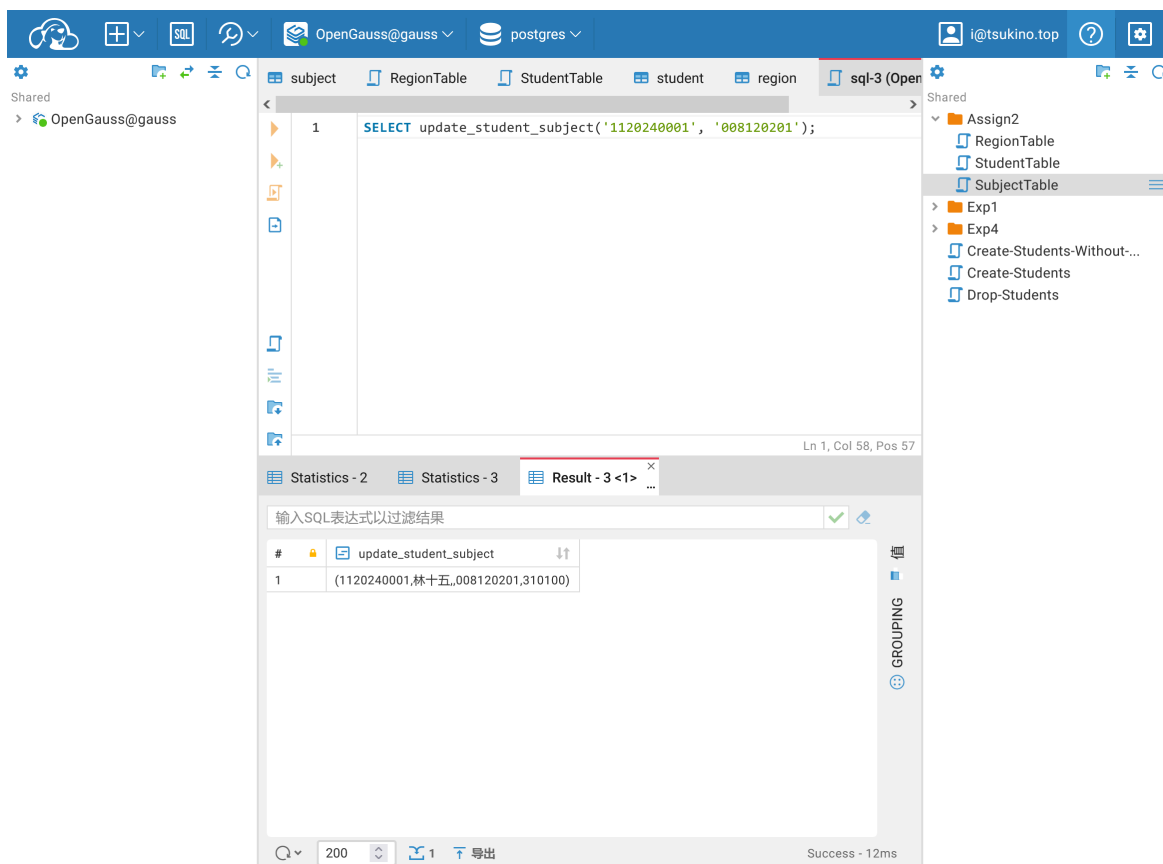


图 5-1 成功修改学生专业信息

第6章 设计成绩表

为进行学生成绩的储存与查询，设计学生成绩表如下：

```
DROP TABLE IF EXISTS score;
CREATE TABLE score (
    id VARCHAR(20) NOT NULL, -- 学生ID, 外键引用学生表的ID字段
    subject_name VARCHAR(20) NOT NULL, -- 科目名称
    score NUMERIC(5,2) NOT NULL, -- 成绩, 数值类型, 最多5位数字和2位小数
    PRIMARY KEY (id, subject_name), -- 主键由学生ID和科目名称组成, 确保唯一性
    FOREIGN KEY (id) REFERENCES student(id) -- 外键约束, 引用学生表的ID字段
);
```

该表格使用外键约束来确保学生 ID 的有效性；使用由学生 ID 和科目名称组成复合主键，以确保每个学生在每个科目上只能有一条成绩记录。

插入以下测试用例数据：

```
INSERT INTO score (id, subject_name, score) VALUES
-- 2009年的学生
('1120090001', '数据库设计与开发', 85.0),
('1120090001', '软件工程导论', 90.0),
('1120090001', '科学发展观', 88.0),
('1120090002', '数据库设计与开发', 78.0),
('1120090002', '软件工程导论', 88.0),
('1120090002', '科学发展观', 88.0),
('1120090003', '数据库设计与开发', 92.0),
('1120090003', '软件工程导论', 95.0),
('1120090003', '科学发展观', 90.0),
('1120090004', '数据库设计与开发', 80.0),
('1120090004', '软件工程导论', 85.0),
('1120090004', '科学发展观', 82.0),
-- 2000年的学生
('1120000001', '数据库设计与开发', 75.0),
('1120000001', '软件工程导论', 80.0),
('1120000001', '三个代表重要思想', 85.0),
('1120000002', '数据库设计与开发', 82.0),
('1120000002', '软件工程导论', 78.0),
```

```
('1120000002', '三个代表重要思想', 80.0),
('1120000003', '数据库设计与开发', 88.0),
('1120000003', '软件工程导论', 90.0),
('1120000003', '三个代表重要思想', 92.0),
('1120000004', '数据库设计与开发', 70.0),
('1120000004', '软件工程导论', 72.0),
('1120000004', '三个代表重要思想', 75.0),
-- 2015年的学生
('1120150001', '数据库设计与开发', 95.0),
('1120150001', '软件工程导论', 98.0),
('1120150001', '习近平新时代中国特色社会主义思想概论', 90.0),
('1120150002', '数据库设计与开发', 85.0),
('1120150002', '软件工程导论', 87.0),
('1120150002', '习近平新时代中国特色社会主义思想概论', 88.0),
('1120150003', '数据库设计与开发', 90.0),
('1120150003', '软件工程导论', 92.0),
('1120150003', '习近平新时代中国特色社会主义思想概论', 91.0),
('1120150004', '数据库设计与开发', 88.0),
('1120150004', '软件工程导论', 91.0),
('1120150004', '习近平新时代中国特色社会主义思想概论', 75.0),
-- 在校生
('1120240001', '数据库设计与开发', 80.0),
('1120240001', '软件工程导论', 85.0),
('1120240001', '习近平新时代中国特色社会主义思想概论', 82.0);
```

6.1 查询成绩表

使用连接语句进行成绩表的查询：

```
SELECT s.id, s.name, sc.subject_name, sc.score
FROM get_all_students() AS s
JOIN score AS sc ON s.id = sc.id;
```

6.2 实现交叉表查询

为了使成绩查询更加直观，可以使用交叉表的思想，将成绩表转换为以下形式：

作业二 数据库设计

姓名	三个代表重要思想	习近平新时代中国特色社会主义思想概论	数据库设计与开发	科学发展观	软件工程导论
李四	NULL	NULL	78.00	88.00	88.00
...

这可以通过以下 SQL 语句实现：

```
CREATE OR REPLACE FUNCTION get_student_scores()
RETURNS SETOF record AS $$
DECLARE
    cols TEXT;           -- 动态生成的列定义
    sql TEXT;            -- 动态生成的完整 SQL 语句
BEGIN
    -- 获取所有科目名称，并生成 CASE 表达式
    SELECT string_agg(
        format(
            'MAX(CASE WHEN subject_name = %L THEN score END) AS %I',
            subject_name, subject_name
        ),
        ', '
    ) INTO cols
    FROM (
        SELECT DISTINCT subject_name
        FROM score
        ORDER BY subject_name
    ) AS subjects;

    -- 如果没有科目，仅返回学生姓名
    IF cols IS NULL THEN
        sql := 'SELECT name FROM student';
    ELSE
        -- 构造完整的 SQL 查询
        sql := format(
            'SELECT s.name, %s
            FROM student s
            LEFT JOIN score sc ON s.id = sc.id
            GROUP BY s.name, s.id',
```

作业二 数据库设计

```
        cols
    );
END IF;

-- 执行动态查询
RETURN QUERY EXECUTE sql;
END;
$$ LANGUAGE plpgsql;
```

要进行查询，需要先对动态的列名称进行查询，可使用这个 `DO` 块来动态生成最终的 SQL 查询语句：

```
-- 进行查询
DO $$
DECLARE
    cols TEXT;          -- 动态列定义
    final_sql TEXT;      -- 最终 SQL 查询
BEGIN
    -- 获取所有科目列定义 (如 "数学 NUMERIC, 物理 NUMERIC")
    SELECT string_agg(format('%I NUMERIC', subject_name), ', ')
    INTO cols
    FROM (
        SELECT DISTINCT subject_name
        FROM score
        ORDER BY subject_name
    ) AS subjects;

    -- 构造完整的查询语句
    final_sql := format(
        'SELECT * FROM get_student_scores() AS (name VARCHAR(20), %s);',
        cols
    );

    RAISE NOTICE '%', final_sql;
    -- 执行并输出结果
    EXECUTE final_sql;
END $$;
```

作业二 数据库设计

执行这个 DO 块后，将会输出动态生成的 SQL 查询语句，在本次作业的示例数据中，结果为：

```
SELECT * FROM get_student_scores() AS (name VARCHAR(20), "三个代表重要思想"
NUMERIC, "习近平新时代中国特色社会主义思想概论" NUMERIC, "数据库设计与开发" NUMERIC,
"科学发展观" NUMERIC, "软件工程导论" NUMERIC);
```

查询结果如下：

#	name	123 习近	123 数	123 科	123 软
1	吴十	75	[NULL]	70	72
2	朱八	80	[NULL]	82	78
3	刘七	85	[NULL]	75	80
4	徐九	92	[NULL]	88	90
5	张三	[NULL]	[NULL]	85	88
6	林十五	[NULL]	82	80	85
7	赵六	[NULL]	[NULL]	80	82
8	钱十一	[NULL]	90	95	98
9	周十三	[NULL]	91	90	92
10	李四	[NULL]	[NULL]	78	88
11	郑十四	[NULL]	75	88	91

图 6-1 成绩表的交叉表形式查询结果

第 7 章 总结与体会

在这次作业中，通过设计和实现一个教学系统的数据库，深入理解了数据库设计中的几个重要概念：

数据的时效性

设计学科表和籍贯表时，都需要考虑数据的时效性问题。例如：

- 学科代码随时间变化，同一编码在不同时期可能对应不同学科

作业二 数据库设计

- 行政区划调整导致籍贯信息变化，如重庆从四川省独立

我们通过以下方式解决这些问题：

- 使用生效时间和失效时间记录数据的有效期
- 使用 `replaced_by` 字段记录替换关系
- 保留历史记录而不是直接更新

数据的层次结构

在设计学科表时，需要处理复杂的层次结构：

- 学科编码具有固定的层次含义（一级、二级等）
- 编码的层级数量可能发生变化（如增加四级学科）
- 编码的长度也可能变化（如从 2 位变为 3 位）

解决思路：

- 使用递归结构储存层级关系
- 设计触发器自动处理下级编码的更新
- 通过视图简化对层次结构的查询

数据的一致性

在处理数据变更时，需要特别注意数据的一致性：

- 学生毕业后的专业信息不应随学科代码变更而改变
- 但籍贯信息应随行政区划变更而更新
- 成绩记录需要与学生信息保持关联

采取的措施：

- 使用外键约束确保数据引用的正确性
- 设计查询函数自动处理时间相关的版本选择
- 通过视图屏蔽底层数据的复杂性

实用性思考

此次设计中还有一些值得思考的实用性问题：

- 数据表的设计需要在完整性和效率之间取舍
- 触发器的使用需要考虑性能影响
- 交叉表查询的实现需要考虑动态性和可扩展性