空间数据库原理及开发技术

第4章 空间数据查询与索引

中南大学地信系 李光强
QQ：41733233

教学目标

- 掌握数据定义、操纵、连接等结构化查询语法
- 熟悉PSQL常用运算和函数
- 掌握空间数据操作的结构化查询语法，熟悉空间结构化查询的优化策略
- 掌握数据库索引和空间索引的概念及其使用
- 熟悉PostgreSQL管理索引和PostGIS管理空间索引的SQL语法

教学重点

- 结构化查询语法和空间结构化查询语法
- 空间索引及其使用

教学内容

- 4.1 结构化查询概述
- 4.2 结构化查询语言
- 4.3 PSQL常用运算符和函数
- 4.4 空间结构化查询语言及其优化
- 4.5 空间数据索引技术

4.1 结构化查询概述

- 4.1.1 SQL概述
- 4.1.2 SQL特点

4.1.1 SQL概述

- **结构化查询语言**（structured query language，**SQL**）是管理关系型数据库的标准化语言，是一种通用的、功能强大的关系数据库语言。
- SQL的功能包括数据库定义、数据表定义，数据的插入、编辑和删除，以及数据库安全性和完整性定义与控制等功能。

4.1.1 SQL概述

(一)SQL发展历程

- (1)早期阶段（20世纪70年代）
  - IBM公司研发了SEQUEL语言，语法简单，只有基本的**SELECT、INSERT、UPDATE、DELETE语句**。
- (2)标准化阶段（20世纪80年代）
  - ANSI（美国国家标准协会）开始着手制定SQL的标准，发布了**SQL-86**(或SQL1)。
- (3)扩展阶段（20世纪90年代）
  - SQL的应用范围逐渐扩展到了Web应用、企业信息系统等领域，功能增加了存储过程、触发器、事务等的操作，发布了**SQL-99**(或SQL3)。
- (4)安全阶段（21世纪初）
  - SQL标准增加了数据安全性操作相关功能，发布了**SQL-2003**。
- (5)大数据阶段（21世纪10年代以后）
  - 发布了**SQL-2011**标准，增加了JSON和XML相关的功能，并且逐渐与NoSQL技术融合

4.1.1 SQL概述

- (二)SQL特点
  - （1）功能强大，综合统一
  - （2）非过程化
  - （3）性能高效，功能可扩展
  - （4）支持复杂查询
  - （5）语言开放，可移植
  - （6）简单易学，语言独立

4.1.2 PSQL工具

- PSQL是PostgreSQL关系型数据库管理系统的命令行接口工具，或称为客户终端命令交互操作界面。
- PSQL提供了丰富的功能和命令，可以用来执行SQL查询，管理数据库对象（如表、索引、触发器等），执行数据库备份和还原操作，管理用户及其权限，查看和分析数据库性能指标等。
- PSQL工具存放在PostgreSQL的安装目录下的bin文件夹中，输入以下命令启动PSQL：

  psql [-h<host>] [-p<port> [-U <username>] [-d <database_name>]

4.1.2 PSQL工具

- PSQL启动方法
  - （1）Windows操作系统：在cmd窗口中，输入“**runas /user:postgres cmd**”以postgresql用户启动cmd窗口，并在启动cmd窗口中输入“**psql** “使用默认端口（5432）连接本地PostgreSQL数据库。如果PostgreSQL数据库修改了服务端口，则需要使用”**psql –p 端口号**”命令连接。
  - (2) centos操作系统：在终端shell窗口中，输入”su postgres”命令，将用户切换到postgres，然后输入”psql “使用默认端口（5432）连接本地PostgreSQL数据库。如果修改了服务端口，则使用” **psql –p 端口号**”命令连接。
- 操作演示[centos]
  - [root@gis ~]# su postgres
  - bash-4.4$ psql -p 7788
  - could not change directory to "/root": Permission denied
  - psql (14.4)
  - Type "help" for help.
  - postgres=#

4.2 结构化查询语言

结构化查询语言主要包括**数据定义语言**（data definition language，**DDL**）、**数据操纵语言**（data manipulation language，**DML**）、**数据控制语言**（data control languange，**DCL**）等功能语法。

- 4.2.1 数据定义语句
- 4.2.2 数据操纵语句
- 4.2.3 连接查询语句
- 4.2.4 嵌套查询语句

4.2.1 数据定义语句

- 数据定义语言用于定义数据库对象，包括**创建、修改、删除数据库、数据表、视图、存储过程和函数**等对象的操作，关键词包括**Create、Alter、Drop**等。
- （一）数据库的定义
  - (1)创建数据库基本语法：

    CREATE DATABASE <name>;
  - (2)创建数据库名称的语法：

    ALTER DATABASE <old-name> RENAME TO <new-name>;
  - (3)删除数据库的基本语法：

    DROP DATABASE [IF EXISTS] <name> [FORCE];
  - 操作演示

4.2.1 数据定义语句

- （二）数据表的定义
  - 在创建数据表之前，必须在**psql cli**命令里连接已创建的数据库，才能进行数据表定义的操作。
  - (1)创建数据表基本语法：

```
CREATE TABLE [IF EXISTS] [scheme.]<name>([
  { <column_name> <data_type> [ column_constraint [ ... ] ]}
    [,...]
] );
```

    - 例4-8：

```
mygdb=# CREATE TABLE users(
                id              serial4             primary key         ,
                name            varchar(20)         not null            ,
                password        varchar(32)         not null            ,
                logined_times   int                 default 0           ,
                last_logined    timestamp                               ,
                descript        text
               );
```

  - 操作演示

4.2.1 数据定义语句

中南大学 CENTRAL SOUTH UNIVERSITY

PostgreSQL常用数值类型

名字	存储长度	描述	范围
smallint或int2	2 字节	小范围整数	-32768 到 +32767
integer或int	4 字节	常用的整数	-2147483648 到 +2147483647
bigint或int8	8 字节	大范围整数	-9223372036854775808 到 +9223372036854775807
numeric	可变长	用户指定的精度，精确	小数点前 131072 位；小数点后 16383 位
real	4 字节	可变精度，不精确	6 位十进制数字精度
float	8 字节	可变精度，不精确	15 位十进制数字精度
smallserial 或 serial2	2 字节	自增的小范围整数	1 到 32767
serial	4 字节	自增整数	1 到 2147483647
bigserial或serial8	8 字节	自增的大范围整数	1 到 9223372036854775807
char(n)	n字节	n个定长字符	
varchar(n)	变长字节，最多n字节	最多可存储n个字符，存储为实际使用字节数	0～n个字符
text	变长字符	用于存储超长字符	字符数无长度限制
date	4字节	只用于日期	4713BC到5874897 AD
timestamp	8字节	日期和时间(无时区)	4713 BC到294276 AD，精度为1ms
time	8字节	只用于一日内时间	00:00:00到24:00:00，精度为1ms
boolean	1字节	逻辑型	true或false

4.2.1 数据定义语句

- (1)创建数据表基本语法（续）：
  - 例4-9：

```
mygdb=# CREATE SCHEMA sys;
mygdb=# CREATE TABLE users(
            id              serial4         primary key     ,
            name            varchar(20)     not null        ,
            password        varchar(32)     not null        ,
            logined_times   int             default 0       ,
            last_logined    timestamp                       ,
            descript  text
        );
```

- 操作演示

4.2.1 数据定义语句

- (2)修改数据表基本语法：

  ALTER TABLE <name> action [,...];

  - 其中常用的action包括：
    - ① 修改数据表名称：RENAME TO <new_name>
    - ② 设置数据表的模式：SET SCHEMA <new_schema>
    - ③ 修改字段名称：RENAME <column_name> TO <new_column_name>
    - ④ 修改约束名称：RENAME CONSTRAINT <constraint_name> TO <new_constraint_name>
    - ⑤ 添加字段：ADD [COLUMN] [IF NOT EXISTS] <column_name> <data_type> [column_constrant]
    - ⑥ 删除字段：DROP [COLUMN] [IF EXISTS] <column_name>
    - ⑦ 修改字段类型：ALTER [COLUMN] <column_name> [SET DATA] TYPE <data_type>
    - ⑧ 删除约束：DROP CONSTRAINT [IF EXISTS] <constraint_name>

4.2.1 数据定义语句

- (2)修改数据表基本语法（续）
  - 例4-10：
    ① mygdb=# ALTER TABLE users RENAME TO login_users;
    ② ALTER TABLE[操作提示，下同]
    ③ mygdb=# ALTER TABLE login_users RENAME descript TO remark;
    ④ ALTER TABLE
    ⑤ mygdb=# ALTER TABLE login_users ALTER remark TYPE varchar(500);
    ⑥ ALTER TABLE
    ⑦ mygdb=# ALTER TABLE login_users ADD COLUMN email varchar(30);
    ⑧ ALTER TABLE
    ⑨ mygdb=# ALTER TABLE login_users DROP COLUMN logined_times;
    ⑩ ALTER TABLE
    ⑪ mygdb=# ALTER TABLE login_users SET SCHEMA public;
    ⑫ ALTER TABLE

4.2.1 数据定义语句

- (3)删除数据表基本语法

  DROP TABLE [IF EXISTS] <name> [CASCADE | RESTRICT]

  - 例4-11：
    ① mygdb=# DROP TABLE public.users CASCADE;
    ② DROP TABLE
  - 操作演示

4.2.1 数据定义语句

（三）视图的定义

- **视图**（view）是一张虚拟数据表（简称**虚表**），不是实际存储数据的数据表（简称**实表**）。视图是一种预定义查询形式存在的表组合，可以从一个数据表或多个数据表中选择字段以及要满足条件的数据记录集。
- (1)创建视图基本语法：

```
CREATE VIEW <view_name> AS
        SELECT <column1>[, column2.....]
        FROM <table_name> [,...]
        [WHERE condition];
```

  - 例4-12：

```
mygdb=#CREATE VIEW v_users
        AS
        SELECT *
        FROM login_users;
```

4.2.1 数据定义语句

- (2)删除视图基本语法：

  DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT];

- 例4-13：

  DROP VIEW v_users;

4.2.2 数据操纵语句

- **数据操纵语言**（data manipulation language，**DML**）包括数据插入(**Create**）、查询（**Read**）、更新（**Update**）、删除（**Delete**）等操作，一些开发资料中将数据操纵语句合并简称为**CRUD**操作。
- (一）数据插入
  - 基本语法：

```
INSERT INTO <table_name> [ AS alias ] [ ( column_name [, ...] ) ]
    < DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query >
```

  - 参数说明：

① <table_name>为要操作的数据表名称。

② [column_name]为要插入的字段列表，该参数为可选项。当该项为空时，将向数据表所有字段插入数据。

③ [DEFAULT VALUES|VALUES(...)]为要插入的对应数据，该参数为可选项。当该项为空时，将向数据表中插入字段默认值。

  - 例4-14. 向login_users数据表中添加两条记录，语句如下：

```
mygdb=# INSERT INTO public.login_users(name,password,email,remark)
            VALUES('admin' ,md5('123456') ,'ligq168@163.com'   ,'系统管理员'),
                  ('test'      ,md5('abc')       ,'41733233@qq.com'  ,'普通用户'   );
```

4.2.2 数据操纵语句

(二）数据查询

- 数据查询SELECT基本语法如下：

```
SELECT [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY <column_name> ]
    [ HAVING condition ]
    [ { UNION | INTERSECT | EXCEPT }SELECT ]
    [ ORDER BY expression [ ASC | DESC] ]
```

- 参数说明：

① [* | expression [[AS] output_name] [, ...]] 为查询字段列表，*表示所有字段，AS用于定义字段或查询表达式的别名。

② [FROM from_item [, ...]]为查询的数据表或视图等，可以同时从多个表或视图中查询。

③ [WHERE condition]定义查询的条件，condition为条件表达式。

④ [GROUP BY <column_name>]定义分组查询字段，<column_name>为分组字段。

⑤ [HAVING condition]定义分组查询要满足的条件。

⑥ [{ UNION | INTERSECT | EXCEPT }SELECT]查询结果合并、求交、排除等语句。

⑦ [ORDER BY expression [ASC | DESC]]定义查询排序条件。

4.2.2 数据操纵语句

中南大學 CENTRAL SOUTH UNIVERSITY

(二）数据查询(续)

WHERE子句的condition从句可以是任何条件表达式。常用的条件表达式有：

- ① 比较运算符：>、<、>=、<=、=、< >、**like**、**is**、**in**、**between**等。
  - 1）**like**为模糊查询谓词，如查询所有以“a”开始的用户名称，则条件子句为“name like ‘a%’”。
  - 2）**is**是类型比较，常用于判断变量或字段是否为NULL（空），如查询所有remark字段为NULL的记录，则条件子句为“remark is NULL”。
  - 3）**in**谓词用于查询变量或字段的取值在指定的常量集合或子查询结果中，如查询role_id取值为0或1的用户记录，则条件子句为“role_id in (0,1)”。
  - 4）**between**谓词用于查询变量或字段的取值在指定区间的所有记录.
- ② 逻辑运算符：**OR**、**AND**、**NOT**，分别表示逻辑或、逻辑与和逻辑非。

4.2.2 数据操纵语句

(二）数据查询(续)

- 例4-15. 查询名为test和密码明文为abc的账号，并显示账号名、email信息，语句如下：

```
mygdb=# SELECT name,email
        FROM login_users
        WHERE name='test' AND password = md5('abc');
```

- 例4-16. login_users表role_id用户角色id，管理员为0，普通用户为1。统计各类账号数量的语句如下：

```
mygdb=# SELECT role_id as 角色ID, COUNT(0) as 账号数量
        FROM login_users
        GROUP BY role_id;
```

- 例4-17. 按照role_id升序查询显示账号的id,role_id,name,email信息，语句如下：

```
mygdb=# SELECT id,role_id,name,email
        FROM login_users
        ORDER BY role_id;
```

- 例4-18. 在上例中，将role_id的取值显示为“管理员”、“普通用户”，语句如下：

```
mygdb=# SELECT id,
                CASE WHEN role_id=0 THEN '管理员'
                     WHEN role_id=1 THEN '普通用户'
                END as 用户角色,
                name,email
         FROM login_users
         ORDER BY role_id;
```

- 操作演示

4.2.2 数据操纵语句

(二）数据更新

- 数据更新的基本语法如下：

```
UPDATE <table_name> [ [ AS ] alias ]
    SET < column_name = expression[, ...]
    [ FROM from_item [[ AS ] alias] [, ...] ]
    [ WHERE condition ]
```

  - table_name是要更新的数据表名称，column_name是要更新的字段名称，expression要更新的字段值表达式，FROM子句中的from_item可以是数据表、视图、子查询块等，WHERE是约束要更新的满足条件的记录，即仅更新满足condition条件的记录。当更新语句中没有WHERE从句时，将更新数据表的全部记录。

- 例4-19. 将login_users数据表名为admin的账号密码修改为“CSU2023”的MD5密文，语句如下：

```
mygdb=# UPDATE login_users
            SET password = md5('CSU2023')
            WHERE name = 'admin';
```

4.2.2 数据操纵语句

(二）数据更新(续)

- 例4-20. 在一个管理系统里，经常会记录用户的最后登录时间和登录次数。记录admin账号登录信息的更新语句如下：

```
mygdb=# UPDATE login_users
        SET last_logined_time = now(),
            logined_times = logined_times+1
        WHERE name = 'admin';
```

4.2.2 数据操纵语句

(二）数据删除

- 数据删除的基本语法如下：

```
DELETE FROM <table_name> [ [ AS ] alias ]
  [ USING from_item [, ...] ]
  [ WHERE condition];
```

  - 其中，table_name为要删除数据的数据表名称，USING子句中的from_item可以是数据表、视图、子查询块等，WHERE控制删除数据的条件。当删除语句中没有WHERE从句时，将删除所有数据。

- 例4-22. 删除login_users中名称为 “test”的账号，语句如下：

```
mygdb=# DELETE FROM login_users
            WHERE name = 'test';
```

- 例4-23. 删除login_users中所有角色名为 “访客” 的数据记录，语句如下：

```
mygdb=# DELETE FROM login_users a
            USING role b
            WHERE a.role_id = b.role_id AND b.role_name = '访客';
```

role_id	role_name
0	系统管理员
1	普通用户
2	访客

4.2.3 连接查询语句

- **连接查询**是指通过连接两个或以上数据表，并从这些表中检索满足条件的组合数据集的操作过程。
- PostgreSQL 支持多种连接查询方式，包括
  - 自然连接
  - WHERE条件连接
  - 内连接
  - 左连接
  - 右连接
  - 外连接
  - …… ……

4.2.3 连接查询语句

（一）自然连接

- **自然连接**是根据参与连接的多表相同字段的等值条件建立的隐式连接。
- PostgreSQL的自然连接语法如下：

  SELECT <*|字段列表> FROM <T1>[[AS] A1]
  NATURAL [INNER|LEFT|RIGHT] JOIN <T2> [[AS] A2]
  [**NATURAL** [INNER|LEFT|RIGHT] JOIN <T3> [[AS] A3] ...];

  - <T1>和<T2>是参与连接的两个表，T1和T2必须存在相同的字段名称，且将相同字段值相等的记录通过笛卡尔积组合生成查询结果，即PostgreSQL将自动匹配两表中相同字段相同值的记录，[AS]是定义表别名的子句。
  - 自然连接还可以是INNER JOIN、LEFT JOIN 或 RIGHT JOIN 等连接方式。
  - 如果没有指定连接方式，PostgreSQL 将使用默认的**INNER JOIN** 方式，并且参数是两张表中所有同名的列作为关联条件。如果在 SELECT 中使用了星号 *，结果中将仅包括一个连接使用的同名字段和两表不同名字段列表。
- 例4-24. 若要查询用户名称及角色名称，则要将login_users表和role表进行连接，语句如下：

```
mygdb=# SELECT name,role_name
        FROM login_users
        NATURAL JOIN role;
```

role_id	role_name
0	系统管理员
1	普通用户
2	访客

4.2.3 连接查询语句

（二）WHERE条件连接

- **WHERE条件连接**是将满足WHERE条件的两个或多个表进行笛卡尔积运算，得到查询结果的连接方式。基本语法如下：

```
SELECT <*|字段列表> FROM <T1> [[AS] A1],<T2> [[AS] A2] [,...]
WHERE [T1|A1].column <比较运算符> [T2|A2].column
      [OR|AND [T1|A1].column <比较运算符> [T2|A2].column] [...]
```

  - <T1>、<T2>为参与连接操作的数据表，WHERE子句定义的匹配条件。

- 例4-25. 使用WHERE条件连接语句实现例4-24的查询，语句如下：

```
mygdb=# SELECT a.name,b.role_name
        FROM   login_users AS a,role b
        WHERE  a.role_id = b.role_id;
```

role_id	role_name
0	系统管理员
1	普通用户
2	访客

4.2.3 连接查询语句

（三）内连接

- **内连接查询**返回两个表之间完全满足匹配条件的记录，基本语法如下：

```
SELECT <*|字段列表> FROM <T1>[[AS] A1]
INNER JOIN <T2> [[AS] A2]
ON <T1|A1>.column <比较运算符> <T2|A2>.column <比较运算符>;
```

- 例4-26. 使用内连接实现例4-24的查询，语句如下：

```
mygdb=# SELECT
            a.name,b.role_name
        FROM
            login_users AS a
        INNER JOIN
            role AS b
        ON
            a.role_id = b.role_id;
```

4.2.3 连接查询语句

- （四）外连接

  外连接又分为左（外）连接、右（外）连接和全（外）连接。

  - (1)左连接

    左外连接（left outer join）简称左连接（left join），这种连接方式显示左表的所有记录，然后再分别与右表进行笛卡积组合。如果右表中没有与左表匹配的行，则结果集中右表字段的值显示为 NULL。基本语法如下：

    ```
    SELECT <*|字段列表> FROM <T1>[[AS] A1]
    LEFT [OUTER] JOIN <T2> [[AS] A2]
    ON <T1|A1>.column <比较运算符> <T2|A2>.column <比较运算符>;
    ```

  - 例4-27. 使用左（外）连接实现例4-24的查询，语句如下：

    ```
    mygdb=# SELECT a.name,b.role_name
                FROM login_users AS a
                LEFT JOIN role AS b
                ON a.role_id = b.role_id;
    ```

4.2.3 连接查询语句

- （四）外连接(续)
  - (2)右连接

右外连接（right outer join）简称右连接（right join），这种连接方式显示右表的所有记录，然后再分别与左表进行笛卡尔积组合。如果左表中没有与右表匹配的行，则结果集中左表字段的值显示为 NULL。基本语法如下：

```
SELECT <*|字段列表> FROM <T1>[[AS] A1]
RIGHT [OUTER] JOIN <T2> [[AS] A2]
ON <T1|A1>.column <比较运算符> <T2|A2>.column <比较运算符>;
```

- 例4-28. 使用右（外）连接实现例4-24的查询，语句如下：

```
mygdb=# SELECT a.name,b.role_name
        FROM login_users AS a
        RIGHT JOIN role AS b
        ON a.role_id = b.role_id;
```

4.2.3 连接查询语句

- （四）外连接(续)
  - (3)全连接

    全外连接（full outer join）简称全连接（full join），这种连接方式显示左、右表中的所有记录，若左或右表存在没有匹配的记录，则结果集对应字段显示为 NULL。基本语法如下：

    SELECT <*|字段列表> FROM <T1>[[AS] A1]

    FULL [OUTER] JOIN <T2> [[AS] A2]

    ON <T1|A1>.column <比较运算符> <T2|A2>.column <比较运算符>;

  - 例4-29. 使用全（外）连接实现例4-24的查询，语句如下：

```
mygdb=# SELECT a.name,b.role_name
        FROM login_users AS a
        FULL JOIN role AS b
        ON a.role_id = b.role_id;
```

4.2.4 嵌套查询

- **嵌套查询**是在一个SELECT查询语句中嵌入了其它的SELECT查询语句，嵌入的查询语句称为**子查询语句**（或块）。
  - 例4-30. 查询所有角色名称为“管理员”的用户名称，查询语句如下：

```
mygdb=#  SELECT name
         FROM login_users
         WHERE role_id in
                  (
                    SELECT role_id
                    FROM role
                    WHERE role_name = '管理员'
                  );
```

子查询

  - 上例也可以写成：

```
mygdb=#  SELECT name
         FROM login_users a,
             (
               SELECT role_id FROM role
               WHERE role_name = '管理员'
             ) foo
         WHERE a.role_id = foo.role_id;
```

4.2.4 嵌套查询

- 嵌套查询不仅用于SELECT查询语句，也可以用于UPDATE和DELETE语句，实现多表联合更新和多表联合删除操作。
  - **例4-31.** 将例4-23改为嵌套删除语句，语句如下：

```
mygdb=# DELETE FROM login_users
            WHERE role_id in
                    (
                      SELECT role_id FROM role WHERE role_name = '访客'
                );
```

4.3 PSQL常用运算符和函数

- 4.3.1 PSQL运算符
- 4.3.2 PSQL常用函数

4.3.1 PSQL运算符

类别	运算符	描述	实例
算术运算符	+	加	2 + 3 结果为 5
	-	减	2 - 3 结果为 -1
	*	乘	2 * 3 结果为 6
	/	除	3 / 2 结果为 1
	%	模（取余）	3 % 2 结果为 1
	^	指数	2 ^ 3 结果为 8
	|/	平方根	|/ 25.0 结果为 5
	||/	立方根	||/ 27.0 结果为 3
	!	阶乘	5 ! 结果为 120
	!!	阶乘	!! 5 结果为 120
字符运算符	||	字符串连接	'Hello '||' CSU'结果为"Hello CSU"

类别	运算符	描述	实例
日期时间运算符	+	日期时间加	date '2023-06-28' + integer '7'结果为2023-07-05
	-	日期时间减	date '2023-06-28' - integer '7'结果为2023-06-21
比较运算符	=	等于	(1 = 2) 为 false。
	!=	不等于	(1 != 2) 为 true。
	<>	不等于	(1 < > 2) 为 true。
	>	大于	(1 > 2) 为 false。
	<	小于	(1 < 2) 为 true。
	>=	大于等于	(1 >= 2) 为 false。
	<=	小于等于	(1 <= 2) 为 true。
逻辑运算符	AND	逻辑与运算符	(1>2) AND(3>4) 为false
	NOT	逻辑非运算符	NOT (1>2) 为true
	OR	逻辑或运算符	(1>2) OR (3<4) 为true

4.3.2 PSQL常用函数

- PostgreSQL提供的函数包括**聚合函数、数学函数、三角函数、字符函数、日期函数、类型转换函数**等。
- （一）常用的**聚合**函数：
  - (1) **count(*)** 函数　：计算数据表记录数，其中“*”也可以使用数字表示，如count(0)。
  - (2) **max(x)** 函数　：查询指定字段x的最大值。
  - (3) **min(x)** 函数　：查询指定字段x的最小值。
  - (4) **avg (x)** 函数　：计算指定数值型字段x的平均值。
  - (5) **sum(x)** 函数　：计算指定数值型字段x所有值的和。
  - (6) **var_pop(x)**函数：计算指定数值型字段x的总体方差。
  - (7) **variance(x)**函数或**var_samp(x)**函数　：计算指定数值型字段x的样本方差。
  - (8) **stddev_pop(x)**函数　：计算指定数值型字段x的总体标准差。
  - (9) **stddev(x)**函数或**stddev_samp(x)**函数　：计算指定数值型字段x的样本标准差。
  - (10) **array[x]** 函数　：将x作为元素添加到数组。

4.3.2 PSQL常用函数

- （二）常用的**数学**函数：

(1) **abs(x)**函数　　：计算x的绝对值。
(2) **ceil(x)**函数　　：向上取整，即计算不小于x的最小整数。
(3) **degrees(x)**函数　　：把弧度x转换为角度数。
(4) **exp(x)**函数　　：计算x的自然指数。
(5) **floor(x)**函数　　：向下取整，即计算不大于x的最大整数。
(6) **ln(x)**函数　　：计算x的自然对数。
(7) **mod(x,y)**函数　　：计算x除以y的余数。
(8) **power(x,y)**函数　　：计算x的y次幂。
(9) **random( )**函数　　：计算0～1.0之间的随机数。
(10) **round(x,n)**函数　　：计算x保留n位小数的四舍五入结果。
(11) **sqrt(x)**函数　　：计算x的平方根。

4.3.2 PSQL常用函数

- （三）常用的**三角**函数：
  - (1) **acos(x)**函数　：计算x的反余弦值。
  - (2) **asin(x)**函数　：计算x的反正弦值。
  - (3) **cos(x)**函数　：计算x的余弦值。
  - (4) **sin(x)**函数　：计算x的正弦值。
  - (5) **cot(x)**函数　：计算x的余切值。
  - (6) **tan(x)**函数　：计算x的正切值。

4.3.2 PSQL常用函数

- （四）常用的**字符**函数：

(1) **lower(x)**函数：返回x的全部小写字母字符串。

(2) **position(x in y)**函数：返回子串x在y字符串中的位置，当x不在y中时，返回0。如position('CSU' in 'Hello CSU') 返回7。

(3) **substring(x from i for n)**函数：返回字符串x从第i位置开始长度为n的子字符串，如substring('Hello CSU' from 7 for 3) 返回“CSU”。

(4) **trim(x)**函数：返回去除字符串x起首和尾部的空格、回车等非打印字符后的字符串。

(5) **upper(x)**函数：返回x的全部大写字母字符串。

(6) **ascii(x)**函数：返回x第1个字符的ASCII(美国信息交换标准代码)值。

(7) **chr(x)**函数：返回ASCII值为x的字符。

(8) **length(x)**函数：返回字符串x的长度。

(9) **md5(x)**函数：返回x的MD5密文。

(10) **repeat(x, n)**函数：返回x重复n次的字符串。

(11) **replace(x, y, z)**函数：用字符串z替换x中所有y子串并返回，如replace('Hello CSU','CSU','GIS')返回“Hello GIS”。

(12) **split_part(x, y, i)**函数：返回用y分隔x生成的第i个子串，如split_part('Hello CSU&GIS','&',2)返回“GIS”。

(13) **strpos(x, y)**函数：返回字符串y在字符串x中的位置，当y不在x中时，返回0。如strpos('Hello CSU','CSU')返回7。

(14) **substr(x, i [, n])**函数：从字符串x的第i位置开始抽取子字串。当给定参数n>0时，抽取n个字符；如果n未给定，则返回从第i位置开始的所有后续字符。如substr('Hello CSU',7)返回“CSU”，substr('Hello CSU',7,1)返回“C”.

(15) **to_hex(n)**函数：返回数值n对应的十六进制数值，如to_hex(256)返回100。

4.3.2 PSQL常用函数

- （五）常用的日期函数：

(1) **current_date**：返回当前日期。

(2) **current_time**：返回当前时间。

(3) **current_timestamp**：返回当前时间戳。

(4) **date_part(f, x)**函数：根据f指定的域类型返回时间戳x的子域

如date_part('hour', timestamp '2023-06-08 09:38:40')返回9。

(5) **make_date(y, m, d)**函数：返回由y、m和d三个整数构建的年月日日期。

(6) **make_time(h, m, s)**函数：返回由整数h、m和数值s构建的时分秒时间。

(7) **make_timestamp(y, m, d, h, mm, s)**：返回由整数y、m、d、h、mm和数值s构建的年月日时分秒时间戳。

(8) **now( )**函数：返回当前时间戳。

4.3.2 PSQL常用函数

- （六）常用的转换函数：

(1) **to_char(t, fmt)**函数　　：将时间戳t转换为fmt指定格式的字符串，如to_char(now(), 'HH24:MI:SS')返回当前时间“15:32:40”。

(2) **to_char(n, fmt)**函数　　：将整型n转换为fmt指定格式的字符串，如to_char(125, '99999.99')返回“125.00”，其中“”表示空格。

(3) **to_char(d, fmt)**函数　　：将双精度d转换为fmt指定格式的字符串，如to_char(125.8::real, '99999D99')返回“125.80”。

(4) **to_char(n,fmt)**函数　　：将数字n转换为fmt指定格式的字符串，如to_char(-125.8, 'S99999D99')返回“-125.80”。

(5) **to_date(x, fmt)**函数　　：将字符串x按fmt指定的格式转换为日期，如to_date('2023 Jun 09','YYYY Mon DD')返回“2023-06-09”。

(6) **to_number(x, fmt)**函数　　：将字符串x按fmt指定格式转换为数字，如to_number('125.80', '999.99')返回“125.80”数字。

(7) **to_timestamp(x, fmt)**函数：将字符串x转换为fmt指定时间格式的时间戳。

(8) **cast(x as type)**函数　　：将变量或字段x转换为type类型，如cast('2023-06-09' as date)。

- 在转换函数中，cast函数称为强制转换函数，可以将多种类型表达式转换为指定的类型。此外，PostgreSQL还有一种强制转换运算符“::”，可以将“::”前的表达式强制转换为“::”后指定的类型。如'2023-06-09 10:12:22'::timestamp将字符串强制转换为timestamp类型。

4.4 空间结构化查询语言

- 4.4.1 空间结构化查询语言概述
- 4.4.2 OGC扩展的SQL/MM
- 4.4.3 空间结构化查询示例

4.4.1 空间结构化查询语言概述

- **空间数据结构化查询语言**（Spatial Structured Query Language，**SSQL**）是一种用于查询空间数据的结构化查询语言，能够查询、分析和处理空间对象相关的数据.
- （一）空间结构化查询语言特点

(1)SSQL语言是**标准化**的空间数据查询语言，可以与其他标准化的数据库查询语言**集成和交互**。

(2)SSQL语言遵循一定的**规范**，支持多种空间数据库管理系统的多种空间数据类型，如点、线、面、多边形等。

(3)SSQL语言操作对象是**空间数据和空间关系**，除了SQL支持的交、并、差、包含等，还可以实现空间数据的空间关系计算和空间分析等功能。

(4)SSQL语言提供多种**空间度量计算和空间分析**函数，如距离函数、面积函数、长度函数等度量计算函数，以及缓冲区计算、空间相交计算等空间分析函数。

(5)SSQL语言支持多种**空间索引**，如R树索引、四叉树索引、格网索引等，能够提高查询效率和空间数据处理的速度。

(6)SSQL语言能够应用于多种**空间应用场景**，如地理信息系统、位置服务、无人驾驶、物联网等领域。

4.4.1 空间结构化查询语言概述

- （二）空间结构化查询语言发展历程
  - （1）SSQL**萌芽**阶段（1980年代中期到1990年代初期）

    1986年，美国地质调查局（USGS）提出了一种名为SDQL的空间数据查询语言（Spatial Data Query Language）。1992年，美国地球物理学学会（AGU）提出了名为SQL/2的标准化关系数据库查询语言扩展，用于处理空间数据。
  - （2）SSQL**标准化**阶段（1990年代）

    1993年，开放地理空间信息联盟成立，在ISO-SQL/MM标准基础上，开始制定OGC空间数据和空间数据查询语言标准（OGC-SQL/MM），并于1999年发布了空间数据结构化查询标准。
  - （3）SSQL**进一步发展**阶段（21世纪以来）

    2001年，OGC发布了OGC Simple Feature Access（SFA）的标准，其中包括了OGC SFA SQL的空间查询语言规范。2003年，PostGIS发布了PostGIS Extended SQL（ESQL）的空间查询语言。2005年，OGC发布了OGC SFS（Simple Feature Specification）标准和OGC SFS SQL的空间查询语言。2012年，Esri发布了ArcGIS SQL的空间数据查询语言，支持多种空间操作和函数。

4.4.2 OGC扩展的SQL/MM

- ISO标准《Information technology - Database languages - SQL multimedia and application packages - Part 3: Spatial》（简称**ISO-SQL/MM**，ISO/IEC 13249-3）定义了一组用于处理空间数据的数据类型、空间数据索引、空间查询优化、函数和操作符等规范。
- OGC在ISO-SQL/MM标准基础上，制定了《Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option》规范，简称OGC-**SFA**规范（OGC 06-104r4）。
- OGC-**SFA**范定义了具体的空间数据处理与分析的相关函数，对SQL/MM标准进行了扩展，形成了一套标准化的OGC空间查询语言.
- OGC扩展的SQL/MM进一步提升了空间数据查询语言的**标准化**程度，可以**与SQL-92标准兼容**，同时支持地理坐标系和投影坐标系，为处理空间数据提供了丰富的语法和功能.

4.4.2 OGC扩展的SQL/MM

- OGC扩展SQL/MM具有以下几点意义：

① 有助于空间数据管理人员快捷方便地实现专题属性与空间几何图形的**一体化计算与分析**，分担了GIS平台空间数据处理与分析的工作。

② 有助于降低用户使用空间数据处理与分析功能的环境与条件，免除了GIS平台安装与配置相关空间分析组件的困难，用户只需输入结构化查询语句就能**快速完成相关的处理与分析**工作。

③ 将功能强大的空间数据处理与分析功能**集成在空间数据库**中，方便前端GIS系统的并发调用，能够减少GIS系统管理员的维护负担。

4.4.3 空间结构化查询示例

(一)示例使用的数据说明

字段名称	id	geom	prov_code	prov_name
字段类型	int	geometry	int8	varchar(50)
字段说明	Id号	几何字段	省份编码	省份名称
数据	1	略	430000	湖南省

字段名称	id	geom	prov_code	city_code	city_name	population
字段类型	int	geometry	int8	int8	varchar(50)	int4
字段说明	Id号	几何字段	省份编码	市编码	市名称	人口数
数据	186	略	430000	430100	长沙市	10047914
	187		430000	430200	株洲市	3902738
	188		430000	430300	湘潭市	2726181
	189		430000	430400	衡阳市	6645243
	190		430000	430500	邵阳市	6563520
	191		430000	430600	岳阳市	5051922
	192		430000	430700	常德市	5279102
	193		430000	430800	张家界市	1517027
	194		430000	430900	益阳市	3851564
	195		430000	431000	郴州市	4667134
	196		430000	431100	永州市	5289824
	197		430000	431200	怀化市	4587594
	198		430000	431300	娄底市	3826996
	199		430000	433100	湘西土家族苗族自治州	2488105

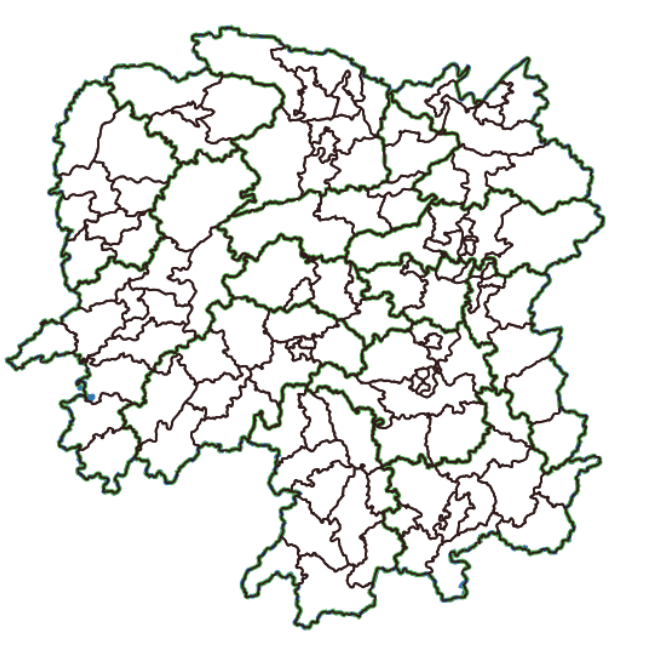
4.4.3 空间结构化查询示例

(一)示例使用的数据说明

字段名称	id	geom	city_code	cnty_code	cnty_name	cnty_type
字段类型	int	geometry	int8	int8	varchar(50)	varchar(6)
字段说明	Id号	几何字段	所属市编码	区县编码	区县名称	类型
部分数据	103	略	430100	430102	芙蓉区	市辖区
	29		430100	430121	长沙县	县
	...		...	...	...	

字段名称	id	geom	cnty_code	town_name
字段类型	int	geometry	int8	varchar(50)
字段说明	Id号	几何字段	所属区县编码	办事处/乡镇名称
部分数据	15469	略	430102	湘湖街道
	15471		430102	马坡岭街道
	...		...	...

字段名称	id	geom	vill_name
字段类型	int	geometry	varchar(50)
字段说明	Id号	几何字段	行政村名称
部分数据	9	略	木溪
	10		岩脚
	11		下大沟
	12		虎头寨
	13		大坪
	14		长冲村
	...		...



4.4.3 空间结构化查询示例

(二)空间结构化查询示例

- 例4-32. 查询并显示所有市界名称及面积，语句如下：

```
SELECT city_name, St_Area(geom) FROM city;
```

- 例4-33. 查询地市的平均面积，语句如下：

```
SELECT avg(St_Area(geom)) FROM city;
```

- 例4-34. 查询面积最大和最小的地市名称及其面积，语句如下：

```
-- 查询面积最大的地市名称及面积
SELECT city_name,St_Area(geom) FROM city
  WHERE St_Area(geom) = (SELECT max(St_Area(geom)) FROM city);
--查询面积最小的地市名称及面积
SELECT city_name,St_Area(geom) FROM city
  WHERE St_Area(geom) = (SELECT min(St_Area(geom)) FROM city);
```

以上示例中的St_Area(geometry)为PostGIS的几何图形面积计算函数。

- 例4-35. 查询与长沙市邻接的地市，语句如下：

```
SELECT city_name
FROM city a WHERE St_Touches(geom, （SELECT geom FROM city WHERE city_name='长沙市') );
```

或者

```
SELECT a.city_name
FROM city a ,  (SELECT geom FROM city WHERE city_name='长沙市') b
WHERE St_Touches(a.geom,b.geom);
```

4.4.3 空间结构化查询示例

(二)空间结构化查询示例

- 例4-36. 查询每个地市的行政村数量，语句如下：

```
WITH foo AS(
            SELECT v.gid,c.city_name
            FROM village v
            JOIN city c
            ON St_Within(v.geom,c.geom)
           )
SELECT city_name,count(0) village_count
FROM foo
GROUP BY city_name;
```

4.5 空间结构化查询优化

- 4.5.1 结构化查询优化
- 4.5.2 空间结构化查询优化
- 4.5.2 空间结构化查询优化策略

4.5.1 结构化查询优化

（一）查询优化概述

- **结构化查询过程**包括**查询分析、查询检查、查询优化和查询执行**4个环节
  - **查询分析**是通过扫描查询语句，分析查询语句的词法和语法，判断查询语句是否符合SQL语法规则。
  - **查询检查**是检查合法的查询语句操作的对象是否存在，包括要查询的数据表或视图是否存在，以及要选取的字段是否存在。
  - **查询优化**是选择一个高效执行查询语句的处理策略，使查询执行更高效。
  - **查询执行**根据优化后的查询策略和执行计划，由代码生成器生成执行查询计划的代码并加以执行，最后将执行结果发送给执行主体。
- **结构化查询优化**（Structured Query Optimization，**SQO**）是优化关系型数据库系统的SQL查询语句，选择高效的处理策略和执行计划，以提高SQL语句执行性能和响应时间。
- 优化查询通常要综合使用多种优化技术，包括**查询重写**、**创建索引**、**优化连接顺序**、**优化表扫描过程**等。

4.5.1 结构化查询优化

（二）SQL语句的优化

- 例4-37. 查询优化示例。在4.4.3示例数据库中，要查询每个地市的办事处/乡镇数量，有以下几种查询方案：
  - **方案1**：首先建立市界-区县界-办事处/乡镇三个数据表的连接，然后在连接结果中统计每个地市的办事处/乡镇数量。语句如下：

```
WITH foo as
(
  SELECT a.city_name,c.id
  FROM city a, county b,town c
  WHERE a.city_code = b.city_code
          AND b.cnty_code = c.cnty_code
)
SELECT city_name,count(0) town_count
FROM foo
GROUP BY city_name;
```

执行时长：**105ms**。

4.5.1 结构化查询优化

（二）SQL语句的优化(续)

- **方案2：**首先建立区县界-办事处/乡镇数据表的连接，然后在连接查询结果中，利用city_code字段分组统计每个地市的办事处/乡镇数量，最后再与市界数据表连接，显示地市名称及其办事处/乡镇数量。语句如下：

```
WITH foo as
(
  SELECT c.city_code,c.cnty_code,t.id
  FROM county c,town t
  WHERE c.cnty_code = t.cnty_code
),
bar AS
(
  SELECT city_code,count(0) cnty_count
  FROM foo
  GROUP BY city_code
)
SELECT c.city_name,b.cnty_count
FROM city c, bar b
WHERE c.city_code = b.city_code;
```

执行时长：**87ms**。

4.5.1 结构化查询优化

（二）SQL语句的优化(续)

- **方案3**：首先建立区县界-办事处/乡镇数据表的连接，然后在连接查询结果中，利用cnty_code分组统计每个区县的办事处/乡镇数量；最后将统计结果与市界数据表连接，统计每个地市的办事处/乡镇数量。语句如下：

```
WITH foo as
(
  SELECT c.city_code,c.cnty_code,t.id
  FROM county c,town t
  WHERE c.cnty_code = t.cnty_code
),
bar AS
(
  SELECT max(city_code) city_code,cnty_code,count(0) town_count
  FROM foo
  GROUP BY cnty_code
)
SELECT max(c.city_name),sum(b.town_count)
FROM city c, bar b
WHERE c.city_code = b.city_code
GROUP BY c.city_code;
```

执行时长：**64ms**。

4.5.1 结构化查询优化

（二）SQL语句的优化(续)

- **方案4**：首先嵌套子查询块查询每个区县的办事处/乡镇数量，然后再与市界数据表连接，统计每个地市的办事处/乡镇数量。语句如下：

```
WITH foo as
(
  SELECT city_code,cnty_code,
                ( select count(0)
                  FROM town t
                  WHERE t.cnty_code = a.cnty_code
                 ) cnt
  FROM county a
)
SELECT max(c.city_name) ,sum(f.cnt)
FROM city c,foo f
WHERE c.city_code = f.city_code
GROUP BY c.city_code;
```

执行时长：**3470ms**。

4.5.1 结构化查询优化

（二）SQL语句的优化(续)

- 上述查询方案均使用了多表连接查询，执行效率相差较大，主要原因在于连接操作的执行顺序。因此，在编写SQL查询语句时，需要注意以下事项:
  - SELECT语句中尽量**避免使用“*”**（表示查询所有字段）
  - SELECT查询中尽量**不要使用字段或数据表别名**
  - 尽量**避免导致索引失效**而扫描全表的语句,如like、!=或< >、OR、NOT IN / IN、条件从句中“=”前使用字段表达式或函数运算、判断是否为空等。
  - 尽可能**限制使用连接查询**，或者尽量减少连接查询的数据表数量；在连接时要“以小驱大”。
  - 在复杂的SQL语句（如多重嵌套或多重连接）里，**尽可能使查询代价小的子查询块**先执行。

4.5.2 空间结构化查询优化

- **空间结构化查询优化**是指在空间数据库中针对空间查询进行优化的过程。与结构化查询一样，影响空间结构化查询的因素较多，一些常见的空间结构化查询优化方法包括：

（1）**创建空间索引**是最常用的优化方法之一。

（2）**空间分区**是将空间数据划分为不同的存储区域，每个区域包含一组或一类空间对象。通过将数据分布在不同的分区，能够减少查询时扫描的数据量。常见的空间分区方法包括网格分区和基于距离的分区。

（3）**空间聚集**是按照空间邻近规则将空间数据分簇存储，以减少查询的磁盘I/O操作。常见的空间聚集方法包括空间填充曲线（如Hilbert曲线）方法和空间网格方法。

（4）**查询优化器**能够根据查询的代价模型和统计信息选择最优的查询执行计划，这也是空间数据库管理系统提供的优化方法。

（5）**空间预处理和缓存技术**是在空间查询执行之前对空间数据进行预处理和暂存处理结果的方法，以减少空间查询时的计算开销。

4.5.3 空间结构化查询优化策略

- 空间结构化查询语句在优化时既需要遵循SQL语句优化的策略，还要考虑以下几种优化规则：
  - （1）对于复杂或嵌套空间查询语句，应尽可能先执行非空间运算
  - （2）尽量避免引起空间索引失效的空间查询语句
  - （3）在满足空间精度要求的前提下，尽可能将空间计算换成简单的代码运算或比较运算
  - （4）尽可减少空间计算的次数
  - 例如，查询与长沙市相邻且人口数大于200万的地级市

```
--方案1：SELECT a.city_name,a.population   FROM city a,city b
         WHERE b.city_name = '长沙市' AND st_touches(b.geom,a.geom) AND a.population > 2000000;

--方案2：SELECT b.city_name,b.population
        FROM
            (SELECT geom FROM city WHERE city_name = '长沙市') a,
           (SELECT city_name,population,geom FROM city WHERE population>2000000) b
         WHERE st_touches(a.geom,b.geom);
```

4.6 空间索引技术

- 4.6.1 数据库索引
- 4.6.2 空间索引概念
- 4.6.3 空间索引方法
- 4.6.4 PostgreSQL管理索引
- 4.6.5 PostGIS管理空间索引

4.6.1 数据库索引

- **索引**是一种加快数据访问速度的特定数据结构，由一组指向存储在计算机内存或外存的数据指针组成，有助于数据快速定位查询所需的数据。
- **数据库索引**是在数据库里存储的一种加快数据查询速度的特殊数据表，由一系列指向数据库表记录的指针组成，能够在查询记录时快速定位所需的记录。
- 索引技术可以帮助数据库系统更快地**查找和定位所需的记录**，避免全表扫描操作，从而提高了数据查询操作的性能。
- 数据库索引技术有多种类型，常用的索引类型包括：
  （1）**B树索引**
  （2）**哈希索引**
  （3）**全文索引**
  （4）**空间索引**

4.6.2 空间索引概念

- 由于空间数据具有空间位置和空间形态等特征，所以空间数据索引技术有别于结构化数据索引技术，原因包括：
  ① 空间数据存储结构不同于传统数据，结构化特性不明显，甚至没有结构化
  ② 空间数据的查询不仅包括结构化属性数据的筛选，也包括空间几何图形的查询和空间运算
  ③ 空间关系计算通常与空间要素的空间范围相关，按照空间位置建立空间索引也是空间数据索引的重要特点
- **空间数据索引**简称空间索引，是一种用于处理具有地理空间位置信息相关的数据索引技术，通过构建面向空间查询和运算操作的特定数据结构，以便在查询和操作空间数据时，能够快速定位所需的地理空间数据。
- **空间索引的核心思想**是将空间数据**划分为多个空间区域**，然后根据空间区域构建数据结构，从而在查询或空间关系计算时，能够根据空间索引快速读取所需的空间要素，缩小空间计算的数据量，避免全表搜索，降低磁盘I/O操作

4.6.3 空间索引方法

- (1)**R(R+)树空间索引**是将空间范围递归地**划分为多个矩形子区域**，并将这些子区域构建为一棵**树状结构**。树的每个节点代表一个矩形区域，矩形区域包含一个或多个空间要素。
- (2)**网格（grid）索引**将空间范围**划分为多个网格**，并为每个网格分配一个唯一标识符。进而，将每个网格覆盖的空间要素指针存储在网格结点中。
- (3)**KD（K-Dimensional tree）树空间索引**将空间范围划分为多个超矩形区域，并将这些区域构建为一棵二叉树。树的每个节点代表一个超矩形区域，在每个节点上存储超矩形区域覆盖空间要素指针。
- (4)**四叉树（qsuadtree）空间索引**使用**层次递归四分的方式划分空间范围**，即四叉树索引首先将空间区域平均划分四个正方形区域，然后对四分的每个区域再平均划分为四个正方形区域，依此递归划分，直到每个子区域中只包含一个空间要素为止。
- (5)**BSP树（binary space partitioning tree）空间索引**使用**层次递归二分方式划分空间范围**，然后将划分的子区域构建为一棵二叉树。树的每个节点代表一个子区域，节点存储子区域覆盖的空间要素指针。
- (6)**填充曲线（peano curve）空间索引**将**多维空间数据映射到一维空间**，通过填充曲线将多维空间中的点映射到一维空间中的点。
- (7)**GiST（Generalized Search Tree）**是一种通用的索引结构，适用于各种类型的数据。GiST索引通过构建一棵多叉树表示空间数据，每个节点包含一个空间要素和对应的BBox（Bounding Box）。
- (8) **BRIN（block range index）**是一种基于块的索引方式，BRIN将相邻块中空间要素的范围信息存储在索引中，而不是存储每个空间要素的位置信息。BRIN索引将整个**空间范围分成若干个块**，然后记录每个块中要素的空间范围，

4.6.4 PostgreSQL管理索引

（一）创建索引

- PostgreSQL创建索引的基本语法如下：

```
CREATE [UNIQURE] INDEX [IF NOT EXISTS]<index_name>
ON <table_name>
[ USING <method> ] (<column_name | expression> [ASC|DESC] [,...]);
```

- 参数说明：
  - ① UNIQURE：可选项，创建唯一值索引，不允许索引列存在重复值。
  - ② index_name：索引名称，在同一数据库中不允许重名。
  - ③ table_name：创建索引的数据表名称。
  - ④ USING method：指定使用的索引方法，可选的方法有btree、hash、gist、spgist 和gin，默认方法是btree。
  - ⑤ column_name | expression：要创建索引的字段名称或字段表达式。
  - ⑥ ASC | DESC：索引列使用升或降序排序。

- 例4-38. 为数据表city的city_name列创建索引，语句如下：

```
CREATE INDEX idx_city_name ON city(city_name);
```

4.6.4 PostgreSQL管理索引

（二）查看索引

- 在PSQL里执行下列语句查看数据表索引：

```
SELECT indexname, indexdef
FROM  pg_indexes
WHERE tablename = '<table_name>';
```

- 其中**table_name**为要查看的数据表名称。

- 例4-38. 使用SQL语句查看city数据表的索引，语句如下：

```
SELECT
        indexname, indexdef
FROM
        pg_indexes
WHERE
        tablename = 'city';
```

4.6.4 PostgreSQL管理索引

（三）修改索引

- PostgreSQL修改索引操作包括**修改索引名称、表空间、存储参数**等，其中修改索引名称的语法如下：

  ALTER INDEX <index_name> **RENAME TO** <new_index_name>

- 其中，index_name原索引名称，new_index_name为索引的新名称。

- 例4-39. 将city的idx_city_name索引名称修改为idx_cityname，语句如下：

  ALTER INDEX idx_city_name **RENAME TO** idx_cityname;

4.6.4 PostgreSQL管理索引

- （四）删除索引
  - 删除索引的语法：

    DROP INDEX [IF EXISTS] <index_name> [, ...] [CASCADE | RESTRICT];
  - 其中，index_name为要删除的索引名称。

  - 例4-40. 删除city数据表的idx_cityname索引，语句如下：

    DROP INDEX idx_cityname;

4.6.5 PostGIS管理空间索引

- PostGIS空间数据库**支持多种空间索引方法**，包括常用的**R树索引、GiST索引、四叉树索引、KD树索引、BRIN索引**等。
- PostGIS创建空间索引的语法如下：

```
CREATE INDEX [IF NOT EXISTS] <index_name>
    ON <table_name>
    USING <method>(<geometry_column>);
```

  - 其中，index_name为索引名称，table_name为数据表名称，method是索引方法，geometry_column为要索引的空间几何字段。PostGIS使用GiST索引取代了其它索引方法。
- 例4-41. 为city数据表的geom字段创建GiST空间索引，语句如下：

```
CREATE INDEX IF NOT EXISTS gidx_citygeom
ON city
USING gist(geom);
```

- 空间数据索引的修改和删除语与和PostgreSQL相同

中南大學
CENTRAL SOUTH UNIVERSITY

感谢您的观看!