**PostGIS:**

Dimension():Integer获取几何对象的几何维数

CoordinateDimension():Integer获取几何对象坐标维数

GeometryType():String获取几何对象的数据类型,如点、线、面等

SRID():Integer用于获取几何类型的空间参考系

Envelope():Geometry用于获取最小边界矩形

AsText():String返回WKT (Well-Known Text)的表达形式,不包含SRID元数据

AsBinary():String返WKB(Well-Known Binary)的表达形式,不包含SRID元数据

IsEmpty():Boolean判断几何类型是否为空

IsSimple():Boolean 判断几何类型是否是简单的

Is3D():Boolean判断是否有z坐标

IsMeasured():Boolean判断是否有M值

Boundary():Geometry 获取边界

Distance(another:Geometry):Distance求距离

Buffer(distance:Distance):Geometry求缓冲区

ConvexHull():Geometry 求本Geom的凸包

Intersection(another:Geometry):Geometry 求本Geom与另一个Geom的交

Union(another:Geometry):Geometry 求并

Difference(another:Geometry):Geometry求差

SymDifference(another:Geometry):Geometry求对称差

Equals(another:Geometry):Boolean判断是否相等

Disjoint(another:Geometry):Boolean是否相离

Intersects(another:Geometry):Boolean是否相交

Touches(another:Geometry):Boolean是否相接

Crosses(another:Geometry):Boolean是否穿越another

Within(another:Geometry):Boolean是否包含于another

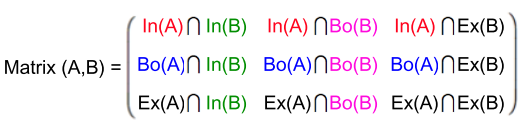
Contains(another:Geometry):Boolean是否包含another

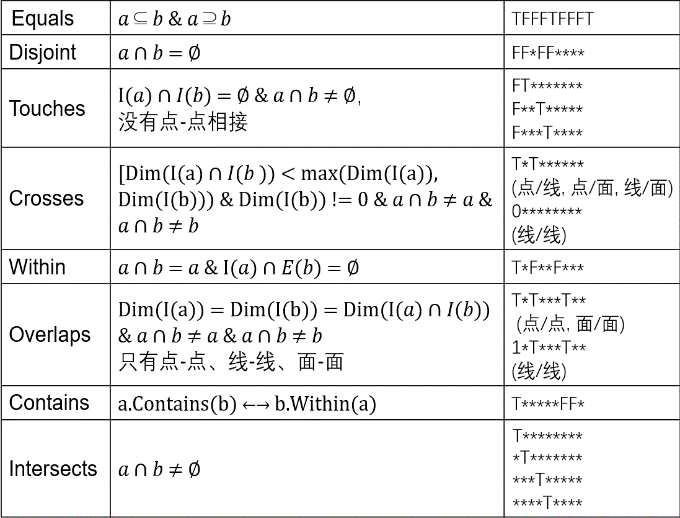
Overlaps(another:Geometry):Boolean是否交叠

Relates(another:Geometry,matrix:String):Boolean 是否符合给定的九交矩阵

LocateAlong(mValue:Double):Geometry选取M值为mValue的点,形成一个新的Geometry

LocateBetween(mStart:Double,mEnd:Double):Geometry 选取M值∈[mStart~mEnd]的点,形成新Geome

**空间关系** **- 九交矩阵**



类型转换

**CAST**(x AS typename) or x**::**typename

SELECT seq, node, edge, cost

FROM **pgr\_dijkstra**(

'SELECT LINK\_ID as id, START\_NODE\_ID as source, END\_NODE\_ID as target, COST FROM AIRPORT\_LINK',

(SELECT AIRPORT\_ID FROM AIRPORT\_LIST WHERE AIRPORT\_NAME='Dillingham, AK'),

(SELECT AIRPORT\_ID FROM AIRPORT\_LIST WHERE AIRPORT\_NAME='Gainesville, FL') );

更新road\_network中的source和target属性

varchar **pgr\_createTopology**(text edge\_table,  
 double precision tolerance, text the\_geom,  
 text id,text source, text target,  
 text rows\_where:=’true’, boolean clean:=false)

SELECT pgr\_createTopology('road\_network', 0.00001, 'geom', 'id', 'source','target', 'true');

分析网络边和顶点

SELECT **pgr\_analyzeGraph**('road\_network',0.00001, 'geom','id','source','target', 'true');

创建相交点修正网络

SELECT **pgr\_nodeNetwork**('road\_network', 0.00011, the\_geom:='geom',id:='id',table\_ending:='1');

**pgr\_aStar**(edges\_sql, starts\_vid, end\_vids, directed, heuristic, factor, epsilon)

RETURNS SET OF (seq, path\_seq [, start\_vid] [, end\_vid], node, edge, cost, agg\_cost) OR EMPTY SET

SELECT \* FROM pgr\_astar('SELECT id, source, target, cost, reverse\_cost, x1, y1, x2, y2 FROM edge\_table', 2, 12)

最少转机次数

**WITH RECURSIVE** X(end\_node,times) AS(

(SELECT END\_NODE\_ID, 0

FROM AIRPORT\_LINK,AIRPORT\_LIST

WHERE AIRPORT\_LINK.START\_NODE\_ID=AIRPORT\_LIST.AIRPORT\_ID AND AIRPORT\_LIST.AIRPORT\_NAME='Dillingham, AK') UNION

(SELECT AIRPORT\_LINK.END\_NODE\_ID, X.TIMES+1

FROM AIRPORT\_LINK, X

WHERE AIRPORT\_LINK.START\_NODE\_ID=X.end\_node AND X.TIMES<10))

SELECT min(times)

FROM X,AIRPORT\_LIST

WHERE X.end\_node=AIRPORT\_LIST.AIRPORT\_ID and AIRPORT\_LIST.AIRPORT\_NAME='Gainesville, FL'

所有可到达的地铁站:

WITH recursive X(node, depth, path, circle) as (

select fromstation, 0, array[fromstation], false from link,station where station.id=link.fromstation and station.**name like '%世界之窗%'**

UNION

select tostation, depth + 1, path || tostation, tostation = any(path)

from link, X

where fromstation = node and not circle and depth<30)

SELECT min(station.id) as id,station.name,min(depth) as depth,min(path) as path

FROM (SELECT node,min(depth) FROM X GROUP BY node)M, X, station

WHERE X.node=M.node and X.depth=M.min and X.node=station.id

GROUP BY name ORDER BY depth

增加几何属性列

select **AddGeometryColumn**('uscities', 'geom', 4326, 'POINT', 2);

UPDATE uscities SET geom = ST\_**SetSRID**(ST\_**Makepoint**(longitude,latitude),4326);

数据插入(最大id+1)

**INSERT INTO** trip (id, start\_time, start\_station\_id, end\_station\_id)

VALUES ((SELECT max(id) from trip)+1, '2015-8-31 23:26', 50, 70);

空间关联查询(事故最多的公路)

SELECT ushighways.gid, ushighways.full\_name as name, ushighways.geom as geom, B.max\_count as accidents\_count

FROM(SELECT max(accidents\_count) as max\_count

FROM(SELECT ushighways.gid as gid, COUNT(\*) as accidents\_count

FROM usaccidents, ushighways

WHERE ST\_DWithin(usaccidents.geom::geography, ushighways.geom::geography, 500) AND (usaccidents.month=7 OR usaccidents.month=8)

GROUP BY ushighways.gid

)A)B,

(SELECT ushighways.gid as gid, COUNT(\*) as accidents\_count

FROM usaccidents, ushighways

WHERE ST\_DWithin(usaccidents.geom::geography, ushighways.geom::geography, 500) AND (usaccidents.month=7 OR usaccidents.month=8)

GROUP BY ushighways.gid)C,

ushighways

WHERE C.accidents\_count=B.max\_count AND C.gid=ushighways.gid

DROP TABLE if exists trip;

CREATE TABLE trip(

id int **not null**,

start\_station\_id smallint,

bike\_id smallint,

**PRIMARY KEY**(id),

**FOREIGN KEY**(start\_station\_id) **REFERENCES** station(station\_id) );

**timestamp** (strat\_time)的转化(to date type)与截取:

date\_part('year', start\_time)

GROUP BY start\_time::date

最新的记录, **DESC**降序排序

SELECT B1.endstationid as stationid, COUNT(\*)

FROM (SELECT bikeid,MAX(endTime) as latestTime

FROM trip

GROUP BY bikeId )A,

(SELECT bikeId, endTime, endStationId

FROM trip )B

WHERE A.bikeId=B.bikeId AND A.lastTime=B.endTime

GROUP BY stationId

ORDER BY count DESC, stationId

从磁盘导入数据

**copy** uscity (gid,name,state,latitude,longitude) from 'E:\\usdata\\uscity.txt' (DELIMITER '#')

触发器

参考完整性(R.A references S.B, cascaded **delete**)

CREATE TRIGGER Trigger1\AFTER Delete ON S REFERENCING Old Row As O\For Each Row\Delete From R where A = O.B

< statement level> Create Trigger Trigger2\After Delete On S\Referencing Old Table As OT\Delete From R where A in (select B from OT)

参考完整性(R.A references S.B, cascaded **update**)

Create Trigger Trigger3\After Update of B On S\Referencing Old Row As O, New Row As N\For Each Row\Update R Set A=N.B Where A=O.B;

实体完整性(主码为属性A)

Create Trigger Trigger4\Before Insert On R\Referencing New Row As N\For Each Row\When exists (select\* from R whereA=N.A)\select raise(ignore);

**视图:**虚表,是从一个或几个基本表(或视图)导出的表;只存放视图的定义,不会出现数据冗余;基表中的数据发生变化,从视图中查询出的数据也随之改变

CREATE **VIEW** current\_employees AS

SELECT NAME, ID, SALARY

FROM EMPLOYEES;

//DROP VIEW current\_employees;

**INSTEAD OF触发器**

CREATE or REPLACE function insert\_value()

returns trigger as $$

BEGIN

insert into track(carid,position) values(new.carid, new.position); \return new;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trig\_insert \INSTEAD OF INSERT ON currenttrack \FOR each row \EXECUTE procedure insert\_value()

**GRANT** SELECT(,INSERT,UPDATE,DELETE,RULE,ALL)

ON Table\_A TO UserX with grant option;

**REVOKE** SELECT on Table\_A from UserX (restrict, cascade) //**restrict**,如果下属有派发过,不允许删除;如果是**cascade**,下属的引用行一起被删除

**count不返回空值,union 0;** SELECT \* FROM Student WHERE GPA >= 3.5 or GPA < 3.5 **or GPA is NULL;ER图key加下划线;**

**ACID**:原子**A**tomicity(恢复机制);一致**C**onsistency(并发控制&恢复);隔离**I**solation(并发控制);持久**D**urability(恢复);**CAP**: Consistency一致Availability可用性Partition tolerance分区容错性;多个**外模式**逻辑独立性,一个**内模式**物理独立性;**外模式/模式映象**:保证数据的逻辑独立性;**模式/内模式映象**:保证数据的物理独立性 (\_将数据操作从具体的计算软件模型和物理存储模式中独立\_)**空间应用系统**:文件系统,混合管理系统,空间数据引擎,对象关系型数据库管理系统;**SFASQL**:PostGIS,标记文本类型、空间数据存储;**SQL/MM**:Oracle Spatial,拓扑数据结构、网络模型,定义了几何拓扑模型;Oracle Spatial, Spatialite, PostGIS是关系/对象关系数据库的**空间扩展数据库模块**

(实体)**主属性**不能取空值,(参照)**外码**取NULL或参照关系中的主码或Unique值;natural属性名字相同 full outer joint不满足结合律; **unique**可有多个null,**PRIMARY KEY**自动获得UNIQUE;可以**无PRIMARY KEY;** A left outer join B最多包含mn个元组

**HAVING子句**:解决WHERE 关键字无法与聚合函数一起使用的问题,用于筛选分组后的各组数据  
SELECT Websites.name, SUM(access\_log.count) AS nums FROM Websites  
INNER JOIN access\_log  
ON Websites.id=access\_log.site\_id  
WHERE Websites.alexa < 200   
GROUP BY Websites.name  
HAVING SUM(access\_log.count) > 200;

**ALTER TABLE** Enrolled **ADD CONSTRAINT** grade\_check CHECK(grade >= 0);**Insert into** Students(sid, age, name) **Values**(‘200012’, 20, ‘李四’);**Update** Students **Set** age = 18 **where** sid = ‘200011’; select id from t1 where **not exists**(select id from t2 where t2.id = t1.id);select id from t1 where **id not in** (select id from t2);not in后面的结果集如果有null,主查询就查不到记录; select **extract**(**dow from** '2019-06-29 10:20:28'::timestamp)返回星期几; **‘’||**direction 转字符串;

**On delete**:(默认set null)**set null**: b删除,A中相关记录的a=null;**cascade**: A中相关记录也全部删除;**restrict**: A中存在对应记录,则无法操作 b 的删除

**SELECT**:笛卡尔积->选择->投影; **LEFT JOIN** 关键字会从左表 (table\_name1) 那里返回所有的行; **NULL**在ORDER BY时默认排序最前面; **COUNT(\*)**,NULL的记录参与计算,**COUNT属性**,NULL的记录忽略;**BETWEEN AND**包括这两个值; **null**满足**check**约束;

**CREATE DOMAIN** GenderDomain CHAR(2) CHECK (VALUE IN (‘男’, ’女’) );

**CREATE TABLE** Persons

(Id\_P int **NOT NULL**,

b INT, **CHECK(b IN (SELECT c FROM S)**,

Ssex **GenderDomain**,

GPA REAL **CHECK(GPA is NOT NULL)**,

**UNIQUE** (Id\_P),

**CONSTRAINT** pk\_En **PRIMARY KEY**(student\_id, cid),

**CONSTRAINT** fk\_En **FOREIGN KEY** (student\_id) **REFERENCES** Students(sid) **on delete/update restrict**)

**空间数据模型**-矢量模型{几何{对象,拓扑},网络},栅格模型,注记文本模型;边界**维数**-1(点空),内部维数一样;Touches+Overlaps==Intersects;**预定义数据类型的用户表**FEATURE,GEOMETRY;**系统表**GEOMETRY\_COLUMNS和SPATIAL\_REF\_SYS;三维空间点线面**坐标维数**333,**几何维数**012;ST\_Distance(C1.geom, C2.geom, true/false),ST\_Distance(C1.geom::geography, C2.geom::geography)返回单位都是**米;预定义数据类**numeric,BLOB**扩展几何类型**Geometry;概念模型->逻辑模型->物理模型;

**地理要素信息构成**:几何,属性,行为;  
**Polygon**是模型层次关系**Surface**的直接子类

**z-order**:x(0,0)y(1,1)->(0101)=5;

**NESTED LOO:**内表有索引,CPU&磁盘I/O;一张表比较小(<10000),作为驱动表;另一张表有索引(且索引选择性较好),作为inner表  
**HASH JOIN**小表构建HASH,内存、临时空间;  
**MERGE JOIN:**两个表差别很大,且较小表(HASH表)可以完全放入内存;或者两个表都可以放入内存;没有索引,并且数据已经排序;不等价(＞, ＜, ≤, ≥, <>)关联;内存、临时空间;  
(就结果来说以上3种join都是一样的) **ST\_Distance**,ST\_Disjoint不能利用空间索引; **R-tree** envelope会重叠,一个object属于一个leaf; **R+tree** envelope不重叠,一个object属于很多leaf; **R-link tree**有最高的并发度 **Hash file** efficient for **point queries** not range queries

**Spatial Queries** CPU and I/O, **traditional** queriesI/O;  
Space partitioning两个表都没有index,Nestedloop with index一个表有,Tree matching两个表都有;物理结构设计阶段与具体的DBMS有关;

**SDBMS与DBMS的不同:**- CPU-cost相比I/O-cost显得不可忽略  
- keys经过了排序以便于分类

**设计异常:**

▬数据冗余:例如,每个系主任的姓名重复出现

▬更新异常 :例如,某系更换系主任后,系统必须修改与该系学生有关的每个元组

▬插入异常:例如,如果一个系刚成立,尚无学生,就无法把这个系及其系主任的信息存入数据库

▬删除异常:如果某个系的学生全部毕业了,我们在删除该系学生信息的同时,把这个系及其系主任的信息也丢掉了

如果A的闭包包含了所有属性,则A是码

Create Trigger Trigger4

Before Insert On R

Referencing New Row As N

For Each Row

When exists (select \* from R where A = N.A)

select raise(ignore);

视图插入数据

create or replace function insert\_values()

**returns trigger** as $$

**DECLARE** a int; b int;

Begin

insert into track(carid, position) values(new.carid, new.position);

**return new**;

end;

**$$ LANGUAGE plpgsql;**

create trigger insertion

**instead of** insert on currenttrack

for each row

**execute procedure insert\_values()**

**丢失修改:**事务2的提交结果破坏了事务1提交的结果, **不可重复读:** 事务2执行更新操作,使事务1无法再现前一次读取结果,**脏读**:事务1修改数据后回滚, 事务2读到的数据就与数据库中的数据不一致;幻读:事务中能添加新元组

**Read Uncommitted**: 一级封锁协议在修改数据前加X锁,读数据不加锁|DirtyReads,NonrepeatableReads,

**Read Committed**: 二级封锁协议在修改数据前加X锁,读数据加S锁,读完释放| NonrepeatableReads,Phantoms

**Repeatable Read**: 三级封锁协议,事务A修改数据前加X锁,读数据加S锁,直到事务结束释放,事务B新增加的数据,不需要获得已有数据的任何锁,因为不做修改,事务B只对新加的数据加X锁,commit之后就释放新数据的X锁 | Phantoms

**Serializable**: 严格按照两段协议对数据加锁

If...then ;elsif then;end if;

for i in **1..(array\_length(ids, 1)-1)** loop

declare g geometry[] = '{}';

for i in 1..ST\_NumPoints(geom) loop

g = **array\_append**(g, ST\_PointN(geom, i));

return ST\_Collect(g);

returns table(id integer) as $$

return query(select...)

表BEFORE INSERT/UPDATE/DELETE返回NULL处理,AFTER抛出异常终止事务,视图INSTEAD OF;TRUNCATE只能用于FOR EACH STATEMENT,不会触发DELETE抛出异常终止事务,元数据不能回滚,删除表中的所有行;

AFTER **UPDATE OF id ON** table\_with\_pk\_id

with **recursive** X(node, depth, path, circle) as (

select start, 0, **array[start]**, false from edges where start = A

union

select end,depth + 1,**path || end,end = any(path)**

from edges, X

where start = node and **not circle**)

SELECT source Table: Falls\_Into FROM Falls\_Into

**CONNECT BY** **PRIOR** source = dest **START WITH** dest = “Miss”

**Main Memory Data-Structures:**Adjacency matrix,list; Breadth first search, Depth first search; Dijkstra’s algorithm, A\*;**Disk-based Tables:**Normalized,Denormalized;Hierarchical routing algorithm;

select **pgr\_nodeNetwork**('roads', 0.001, 'id', 'geom'); 执行后,dead ends数目通常会变少; nodes a network edge table edge\_table\_noded

select **pgr\_createTopology**('roads\_noded', 0.001, 'geom', 'id');

创建节点表,填充id&the\_geom;填充边表target&source(改变)

select **pgr\_analyzeGraph**('roads\_noded', 0.001, 'geom', 'id', 'source', 'target');

填充节点表cnt and chk

Dead ends: 仅有1个相邻节点;Intersections detected: 交点;

gaps的距离小于0.001;

select seq, node, edge, cost from **pgr\_dijkstra**(

'SELECT id, source, target, cost FROM edge\_table',

7, 12, false, false);

pgr\_createTopology,prg\_createVerticesTable会生成网络节点的顶点表;

**OLAP** – Online Analytical Processing

Long transactions,Complex queries,Touch large portions of data,Infrequent updates

**MOLAP**多维数组,**ROLAP**星型&雪片

**with cube**,所有属性都可以取NULL值;A1, A2, ..., An **with rollup**,当Ai是NULL值,Ai+1-An都是NULL值

**MongoDB**属于Document stores,**Redis**属于Key-value stores

**NoSQL**特点

Performance,Scalability可扩展性,Schema模式 flexibility,Analytics capabilities,Shared nothing architecture,Sharing as part of the data storage model,Asynchronous replication异步复制,BASE instead of ACID transactions

**NoSQL以BASE来代替ACID**:

Basic Availability: 基本可用

Soft-state: 软状态/柔性事务,“无连接”

Eventual consistency: 最终一致性

**传统的DBMS在需要ACID的数据中仍有广泛的作用,但是NoSQL更适用于**:

在现实中以无结构或半结构化的方式存储的数据

处理数据的程序需要有一定的性能或扩展性时

处理数据的程序本身已经能保证结果的最终一致性(故不再需要ACID约束)

**当前的NoSQL数据库系统的分类**:

MapReduce framework:分布式计算框架

Key-value stores:通过key快速查询到其value,存储不用考虑value的格式

Document stores:一般用类似json的格式存储,存储的内容是文档型的,有机会对某些字段建立索引,实现关系数据库的某些功能

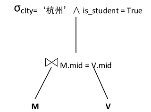
Graph database systems:图形关系的最佳存储。使用传统关系数据库来解决的话性能低下,而且设计使用不方便

(postgres)DROP **INDEX** public.ir; CREATE [UNIQUE] [CLUSTERED] INDEX ir ON public.r USING btree (a);

(postgis)Create Index [indexname] On [tablename] Using GiST ([geometryfield]) //空间索引基本上都是R树索引,非常耗时

非聚集索引相比聚集索引,cost会迅速增加

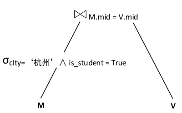
T=关系数;=,B=磁盘页或Block数,V=属性不同取值数

无索引:cost=B(Member);city上有非聚集索引cost=T(Mem)/V(Mem,city);有聚集索引cost=B(Mem)/V(Mem,city);(city,is\_stu)属性上有非聚集索引cost=T(Mem)/V(Mem,city)/V(Mem,is\_stu)//根据实际情况取最小值

nest-loop的cost=

没有索引:if B(Mem)<B(Visi)then B(Mem)+B(Mem)\*B(Visi)

Visi的mid有聚集:B(Mem)+B(Mem)\*T(Visi)/V(Visi, mid)

Visi的mid属有非聚集B(Mem)+B(Mem)\*B(Visi)/V(Visi,mid)

Member的(city,is\_stu)有非聚集:cost1=T(M)/V(M,city)/V(M,is\_stu),cost=cost1+cost1\*B(Visi)

Member的(city,is\_stu)有非聚集索引,Visi的mid上有非聚集: cost1=T(M)/V(M, city)/V(M,is\_stu)=25,cost2=T(V)/V(V,mid)=10,cost=cost2+cost2\*cost1=10+10\*25=260

**空间填充曲线**:点查询二分法查找H-curve值,rangeQ二分法查找每个区域的开始值( (0+7)/2=Block4->H=(9,10)),最邻近,两端二分然后连接中间,再两端…然后单独计算比较。具体读取次数要看内存够不够

**ST\_GeomFromText**('MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))',4326)

**ST\_GeomFromText**('POLYGON((0 0,1 1,1 2,0 0))',4326)

数据库建立与具体DBMS相关:物理设计、实施、运行和维护三个阶段

**BCNF成立:**α→β是平凡的(β∈α)或α是模式R的一个超码

分解BCNF:将R分解为{α∪β}, {R-β}(需要再次检查还有无BCNF)

**数据库系统的安全特性**:数据独立性,数据安全性,数据完整性,并发控制,故障

恢复

**视图可更新标**

**SQL**>视图来源(Select without distinct)于单一的表T;视图外的变量可以null或者有缺省值;子查询与T无关;没有经历GROUP BY或是集聚(以SELECT \* 方式创建的视图可扩充性差,应当尽可能避免)

**PostgreSQL**>**T**he view must have exactly one entry in its FROM list, which must be a table or another updatable view;**T**he view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level;**T**he view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level;**A**ll columns in the view's select list must be simple references to columns of the underlying relation. They cannot be expressions, literals or functions. System columns cannot be referenced, either;**N**o column of the underlying relation can appear more than once in the view's select list;**T**he view must not have the security\_barrier property

故障:事务(Undo)故障;系统(Redo/Undo)故障;介质故障(Redo);计算机病毒——先写日志文件后写数据库

**最近的电影院:**

select id, name, ST\_AsText(geom) from poi where name like '%%影院%%' or name like '%%影城%%' order by ST\_Distance(ST\_SetSRID(ST\_MakePoint(%d,%d),4326), geom) LIMIT 1""".decode('utf-8') % (point[1], point[0])

**车速**

CREATE OR sREPLACE FUNCTION cesu() RETURNS trigger AS $speedNotifyTrigger$

DECLARE

lastpos geometry; GP geometry; speed double precision; timelag integer; limitspeed integer;

BEGIN

IF NOT EXISTS(SELECT \* FROM track WHERE track.carid = NEW.carid)

THEN RETURN NEW;

ELSE

SELECT position INTO lastpos FROM track WHERE carid=new.carid ORDER BY time DESC LIMIT 1;

timelag:= extract (second from (NEW.time - (SELECT max(time) FROM track WHERE carid=new.carid))) + 60\*extract (minute from (NEW.time - (SELECT time FROM track ORDER BY time DESC LIMIT 1)));

FOR GP IN SELECT geom FROM guidepoints LOOP

IF ST\_Distance(lastpos, GP, false) > 100 AND ST\_Distance(NEW.position, GP, false) <100 THEN

speed:= ST\_Distance(lastpos, NEW.position, false)\*3.6/timelag;

SELECT velocity INTO limitspeed FROM guidepoints WHERE GP = geom;

<判断车速,输出警告INSERT INTO notifymessage>

END IF;

END IF;

END LOOP;

END IF;

RETURN NEW;

END;

$speedNotifyTrigger$ LANGUAGE plpgsql;

drop trigger if exists speedNotifyTrigger ON track;

CREATE TRIGGER speedNotifyTrigger BEFORE INSERT ON track

FOR EACH ROW EXECUTE PROCEDURE cesu();

**UPDATE** Person SET FirstName = 'Fred' WHERE LastName = 'Wilson'

**INSERT** INTO Persons VALUES ('Gates', 'Bill', 'Xuanwumen 10')

FROM Persons **INNER JOIN** Orders == FROM Persons, Orders;

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons **LEFT JOIN** Orders --Orders中可能出现null(没有匹配行

ON Persons.Id\_P=Orders.Id\_P;

(FROM Persons **LEFT JOIN** Orders --Persons中可能出现null

(FROM Persons **FULL JOIN** Orders --两边都可能出现null

**SELECT** LastName,FirstName **INTO** P\_backup FROM Persons -–备份

数据库运行和维护阶段->数据流图和数据字典 (软工)

数据库实施阶段-> E-R图/UML图

物理结构设计阶段->关系数据库模式 (规范化)

逻辑结构设计阶段->存储方式、索引和用户权限

概念结构设计阶段

需求分析阶段(从下往上,加粗的前三个与具体的DBMS有关)

NNQuery最近几何对象基于R树的查询流程:先确定该位置所在节点a,遍历a中的几何对象距离p点距离的最大值,从中选出最小值minDist,以minDist为半径画圆,查找与圆相交的对象,在这些对象中选取与p最近的对象

空间数据库中空间查询操作一般分为过滤和精炼两步,空间索引主要用于空间查询执行的过滤步

逻辑数据模型

▬ 层次模型 (Hierarchical Model)

▬ 网状模型 (Network Model)

▬ 关系模型 (Relational Model)

▬ 面向对象模型 (Object Oriented Model)

▬ 对象关系模型 (Object Relational Model)

第一代空间应用系统:文件系统

第二代空间应用系统:混合管理系统

▬ 文件系统管理几何图形数据

▬ 关系数据库管理属性数据

▬ 之间的联系通过目标标识或内部连接码进行,对象唯一标识符OID

第三代空间应用系统:空间数据引擎,图形坐标数据作为一个二进制数据类型,中间件技术

第四代空间应用系统:对象关系型数据库管理系统,数据库技术

ORDBMS提供

▬ 类、继承

▬ 用户自定义类型、函数、索引和规则

An **execution plan** has 3 components

▬ A query tree

▬ A strategy selected for each non-leaf node

▬ An ordering of evaluation of non-leaf nodes

**一次查询优化**：

起始：一个SQL查询语句

终点：一个执行计划（execution plan）

中间步骤：

查询树

逻辑树转换（Logical tree transforms）

· 策略选择（Strategy selection）

RDBMS query optimization ideas needs reconsideration in SDBMS?

▬ **CPU-cost** of an operation is much lower than its

I/O cost

▬ There are multiple **strategies** for each **building block**

▬ Pushing **selection below join** in query-tree always speeds up execution

▬ **Algeraic cost-models** are available to rank alternative strategies

LinearSearch is faster than indexSearch for some range queries

create table t1(c int references t(a), d int);

如果t中的a不是primary key或unique,报错

T1的c属性可以插入null

空间数据库 = 对象关系数据库 + 空间扩展

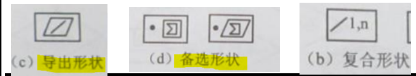
对象关系数据库与关系数据库相比

▬ 扩充了系统类型(结构类型,数组类型,多重集合类型和参照类型,支持继承,特别是方法)

▬ 关系不仅是元组的集合,也可以是对象的集合

通常以中层数据流图作为设计分E/R图的依据

复合形状的“1,n”表示几何形状至少有1个,可以有n个;“0,n”表示可以有0个或n个



a

驾驶距离1km内的商家

Create or replace function ST\_Nearest(position geometry, range float)

Returns setof integer AS $$

declare dist float;

Begin

For res in select bid, position from Business B where

ST\_DWithin(B position, position, range, true) Loop

select sum(cost) into dist

from pgr\_ dijkstra( 'select id, source, target, len as cost from road', ST\_Position2Node(position), ST\_Position2Node(res.position), false);

If dist <= range then

Return next res.bid;

Endif;

End Loop;

End;

$$ language plpgsql;

create tigger NotWithinTrigger

before update or insert on buildings

for each row

execute procedure NotIntersect();

create or replace function NotIntersect()

returns trigger as $$

begin

if exists (select \* from lakes where ST\_Intresects(shore, New.footprint)

then raise notice '建筑物不能位于湖上';

return null;

elsif exists (select \* from streams where ST\_Intersects(centerline, New.footprint))

then raise notice '建筑物不能位于河流上';

return null;

else

returm new;

endif,

end;

$$ LANGUAGE plpgsql;

一个表可以有一个聚集索引和多个非聚集索引

查询在杭州市内的道路

ST\_Within(R.geom, C.geom) or ST\_Crosses(R.geom, C.geom)

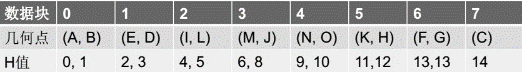
select date(CURRENT\_TIMESTAMP)

select extract(dow from '2019-06-29 10:20:28'::timestamp) 返回星期几; select date\_part('minute', timestamp'2019-03-05 20:38:40');

select CURRENT\_DATE; select CURRENT\_TIMESTAMP;

WHERE current\_timestamp - time <= interval '2 months'

GROUP BY start\_time::date



查询区域为H11-13,则查到11所在数据块5;查询点7,查到数据块3

▬ Block ceil((0 + 7)/2) = 4 -> H = (9, 10)

▬ Block ceil((0 + 4)/2) = 2 -> H = (4, 5)

▬ Block ceil((2 + 4)/2) = 3 -> H = (6, 8)

(分内存够大和内存不够大,不够大数据块可能要读多次)



索引1(city, is\_student),索引2(city, is\_student, birthday),有了索引2之后索引1是冗余的

A->B

select \* from (select count(\*) from sales s1, sales s2 where s1.A = s2.A and s1.B != s2.B) foo where count != 0

已知A->B, C->D: AB->C相当于A->C; AD->B相当于AC->B

封锁技术有S锁和X锁

**S锁(共享锁/读锁):**若事务T对数据对象A加上S锁,则事务T可以读A但不能修改A,其它事务只能再对A加S锁,而不能加X锁,直到T释放A上的S锁。

保证了其它事务可以读A,但在T释放A上的S锁之前不能对A做任何修改

**X锁(排他锁/写锁):**若事务T对数据对象A加上X锁,则只允许T读取和修改A,其它任何事务都不能再对A加任何类型的锁,直到T释放A上的锁。

保证了其它事务在T释放A上的锁之前不能读取和修改A

**一级封锁协议:**

事务T在修改数据R之前必须先对其加X锁,直到事务结束才释放(事务结束包括正常结束COMMIT和非正常结束ROLLBACK)。

可防止丢失修改,并保证事务T是可恢复的

不能保证可重复读和不读“脏”数 据

**二级封锁协议:**

一级封锁协议 + 事务T在读取数据R之前必须先对其加S锁,读完后即可释放S锁。

进一步防止读“脏”数据

由于读完数据后即可释放S锁,不能保证可重复读

**三级封锁协议:**

一级封锁协议 + 事务T在读取数据R之前必须先对其加S锁,直到事务结束才释放。

除了防止了丢失修 改和不读“脏”数据外,还进一步防止了不可重复读

**两段封锁协议:**

每个事务的执行可以分为两个阶段:生长阶段(加锁阶段)和衰退阶段(解锁阶段)

**加锁阶段:**在该阶段可以进行加锁操作。在对任何数据进行读操作之前要申请并获得S锁,在进行写操作之前要申请并获得X锁。加锁不成功,则事务进入等待状态,直到 加锁成功才继续执行

**解锁阶段:**当事务释放了一个封锁以后,事务进入解锁阶段,在该阶段只能进行解锁操作不能再进行加锁操作

一个事务开始后就处于加锁阶段,一直到执行ROLLBACK和COMMIT之前都是加锁阶段

**具体内容:**在事务T的R(A)操作之前,先对A加S锁,如果加锁成功,则执行操作R(A),否则,将R(A)加入A的等待队列

在事务T的W(A)操作之前,先对A加X锁,如果加锁成功 则执行操作W(A),否则,将W(A)加入A的等待队列;在收到事务的Abort或Commit请求后,释放T在每个数据上所加的锁,如果在数据A的等待队列中不空,即有其它的事务等待对A进行操作,则从队列中取出第一个操作,完成加锁,然后执行该操作;执行Abort和Commit请求后,不再接收该事务的读写操作

count不返回空值,union 0; SELECT \* FROM Student WHERE GPA >= 3.5 or GPA < 3.5 or GPA is NULL;ER图key加下划线,MULTI类型图后加1,n;DISTINCT去重; 边界维数-1(点空),内部维数一样;空间参考系

TRUE AND NULL = NULL

FALSE AND NULL = FALSE

TRUE OR NULL = TRUE

FALSE OR NULL = NULL

不能取NULL的完整性约束属性A: primary key; NOT NULL

SQL中所直接有的关系操作:选择、投影、连接、并、交、差(没有除)

排序操作的最佳处理时间是在投影之后

OLTP:Online Transaction Processing(联机交易处理)

短处理;简单查询;使用小部分数据;频繁更新

OLAP:Online Analytical Processing(联机分析处理)

长处理;复杂查询;使用大部分数据;不频繁更新

**WITH ROLLUP**

SELECT dimension-attrs, aggregates

FROM tables Where conditions

GROUP BY dimension-attrs

WITH ROLLUP

--dimension-attrs内部有划分层次的时候,使用WITH ROLLUP可以按照维度的每一层次进行集聚

**WITH CUBE**

SELECT dimension-attrs, aggregates

FROM tables Where conditions

GROUP BY dimension-attrs

WITH CUBE

--当dimension-attrs内部没有划分层次(或不需要考虑层次)时,使用CUBE对维度内的任意层次集聚

**空间索引加速**：

空间函数通常计算量大，扫描全表两两计算比较耗时，先用Envelope判断两个几何是否有关系，如果有，再进行空间函数

对于8种空间查询方法，除Disjoint和Relate外，可以使用索引：

1这些函数只需返回True/False

2可以利用索引快速判断获得False

**R树**

使用分等级的矩形（最小外包矩形，MBR）来组织数据

是B树在k维上的自然扩展

特点：

每个叶结点包含m至M条索引记录，（m≤M/2），除非他是根节点

一个叶结点上的每条索引记录了（I，元组标识符），I是MBR在空间上包含了所指元组表达的k维数据对象

每个非叶结点都有m至M个子结点，除非他是根结点

对于非叶结点中的每个项（I，子结点指针），I是在空间上包含其子结点中矩形的最小外包矩形

根结点至少有两个子结点，除非他是叶结点

所有叶结点出现在同一层

所有MBR的边与一个全局坐标系的轴平行

插入与分裂：

当插入一个元素从而结点需要分裂时，遵循以下规则：

选取两个MBR作为“seeds”

从两个“seeds”开始，不断包括其他MBR，分配的原则为：分配后增长的面积越小越好

**R+树**

MBR可能会被树中非叶结点的矩形分割：

特点：

对于中间结点的每个项（I，子结点指针），当且仅当R被I覆盖时，以子结点指针指向的结点为根的子树包括一个矩形R，唯一的例外是当I是一个叶结点的矩形，这种情况下，两者只交叠

对中间结点的任何个项，I1和I2之间的交叠是0