## 一 地理空间数据库概论

空间分析: 度量、临近、拓扑、方向

概念数据模型:概念数据模型与 DBMS 无关,按用户的观点来对数据和信息建模

逻辑数据模型:直接面向数据库的逻辑结构,通常有一语法和语义的数据库语言,与 DBMS 有关,有

层次模型 网状模型 关系模型 面向对象模型 对象关系模型

物理数据模型:对数据最低层的抽象,描述数据在磁盘或磁带上的存储方式和存取方法

层次模型: 数据结构 - 树 优点 层次数据模型简单, 对具有一对多关系的部门描述清楚 性能优于关 系模型,不低于网状模型 提供了良好的完整性支持 缺点 多对多联系表示不自然 对插入和删除操作 的限制多 查询子结点须通过父结点 层次命令趋于程序化

网状模型: 数据结构 - 图、优点 能够更为直接地描述现实世界 具有良好的性能、存取效率较高、缺 点 结构比较复杂, 而且随着应用环境的扩大, 数据库的结构变得越来越复杂, 不利于最终用户掌握 DDL、DML 语言复杂,用户不容易使用

关系模型: 简单、易访问 关系数据库原理由埃德加·科德 1970 年提出

三级模式 模式定义或描述一个数据集合 逻辑模式用逻辑数据模型对数据库中全部数据的逻辑结构和 特性的描述 是数据库所有用户的公共数据视图



一个数据库(的关系)只有一个模式

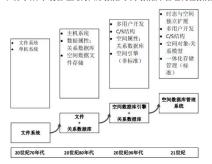
一个数据库(的关系)可有多个外模式 对用户所 用到的那部分数据的描述

内模式 (存储模式) 用物理数据模型对数据的 描述数据库(的关系)只有一个内模式 外模式/ 模式映象保证逻辑独立性 模式/内模式映象 保证数据的物理独立性

空间数据分为矢量数据和栅格数据 矢量数据 是一种用点、线、面等基本空间要素来表示自 然世界的数据 不可再分的最小单元现象称为

空间实体 空间数据主要具有以下重要特征 空间特征 非结构化特征 空间关系特征 时态特征 多尺度 特征 空间数据包括空间坐标和拓扑关系 多尺度包括时间和空间

空间数据库: 特点 数据量大 空间数据与属性数据的集合 应用广泛 SDBMS 功能 空间数据的定义与 操纵 空间数据的组织、存储和管理 后台的事务管理和运行管理 数据库的建立与维护 世界上第一个 **地理信息系统**——加拿大地理信息系统 (CGIS) 20 世纪 80 年代,关系数据库发展并成熟。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 文件管理系统功能较弱 数据安全性、一



致性、完整性、并发控制以及数据损坏 后 的恢复方面缺少基本的功能 空间数 据引擎 图形坐标数据作为一个二进制 数据类型, 由数据库管理系 统进行存储 SDE 提供一组空间数据的操作函数,完 成空间数据的转换, 以及数据的索引调 度和空间数据的存储管理 ESRI的 ArcSDE、SuperMap 的 SDX、中地的 MapGISSDE、开源的 TerraLib 对象关系 型数据库管理系统 支持 SOL. 具有良好 的通用性 具有面向对象特性, 支持复杂 对象及其行为 能够直接存储和管理非 结构化的空间数据 Oracle Spatial IBM 的 DB2 Spatial Extender 微软的 SQL Server Spatial 开源的 PostGIS 开放地理

空间信息协会 Open Geospatial Consortium, OGC SFA SQL 1999 年提出说明了简单地理要素(点、线, 多边形等)的对象模型接口标准

# 二 关系模型与关系代数

关系模型是一种数据模型 组成:关系数据结构 关系数据操作 关系完整性约束

关系模型采用关系 (Relation) 作为数据结构 关系就是简单的表 (Table), 由表名 表头 数据组成 域 (Domain) 是一组具有相同数据类型的值的集合 码:可以唯一确定一个元组的最小属性集合称为候选 码 (Candidate Key), 或简称为码 (Key) 一个关系由关系名、关系模式和关系实例组成, 分别对应于表 名、表头和表中的数据 关系模型中三类完整性约束 实体完整性 参照完整性 用户定义完整性 实体完 整性和参照完整性是关系模型必须满足的完整性约束条件,被称作是关系的两个不变性 关系代数运算 符打法 并 \cup 交\cap 笛卡尔积\times 连接\join 大于等于\geq 小于等于\leq 不等于\neq 除\div 选择\sigma 投影\pi 与\wedge 或\vee 重命名\rho

## 三 关系数据库标准语言

SOL 的产生与发展 1974 年由 IBM 公司的 Boyce 和 Chamberlin 提出 1986 年 10 月美国国家标准局 (ANSI)的数据库委员会批准了 SOL 作为关系数据库语言的美国标准。同年公布了 SOL 标准文本 (简称 SOL-86) 1987 年国际标准化组织 (ISO)也通过了这一标准 1989 年公布了 SOL-89 标准 1992 年公布了 SOL-92 标准(又称 SOL2)(is a basic set) 1999 年公布了 SOL-99 标准(SOL3)(Introduced "Object-Relational" concepts) 当前版本 SOL-2016(2003 was last major update; XML, window functions. sequences, auto-generated IDs 2008 added TRUNCATE, x-query stuff, new triggers (instead of) 2011



temporal data definition. manipulation 2016 added JSON, polymorphic tables) SQL特点 综合统一 高度非 过程化 面向集合的操作方 式 以同一种语法结构提供 两种使用方式 语言简捷, 易学易用

## 四 几何对象模型与查询

空间数据模型是空间信息的一种数据组织方式 有对象模型、场模型、网络模型 注记文字模型 分为以 下 3 类 注记标签 注记文本 注记尺寸注记标签中的文字是要素的某个字段属性、其显示与该层的本文 风格一致, 因此无需额外对其进行定义 标记文本具有自己地理位置和属性 地理要素(feature)对现实 世界空间现象的抽象 由几何、属性、行为三类信息构成地理要素的属性和行为等信息的建模 是由应 用系统的设计者,根据实际应用需求进行建模 几何的建模是数据库管理系统关心的基础问题 关键是 选择一组基本空间数据类型来满足地图常用几何信息的建模要求 OGC 模型规范已逐渐得到认可 一个 依赖干空间参考系和测量参考系的几何类派生出点、线、面、多点、多线、多面等类型 空间数据类型 点: 零维 曲线: 一维 折线 曲线的子类, 采用线性插值 线段 折线的特例, 只有两个

点的线串 环线 由折线派生而来。闭合的、不自相交或相切的折线 多边形二维坐标空间中由一个外边 界、零到多个内边界定义的平坦表面,由一个或一个以上的线环聚合而成 几何集合由一个或多个几何 对象组成的集合,其中的元素必须具有相同的空间参考系和测量参考系 坐标维数和几何维数 坐标维 数是指在一个坐标系统描述一个位置所需的测量或坐标轴的个数(空间维数) 几何维度是在一定前提下 描述一个几何对象所需的参数个数 任何几何模型都有其边界、内部和外部 边界: 一个几何实体界限 的集合,几何维数是其本身几何形状的维数减一 任意几何对象外部的维数总是 2 由于体表面违反了 "多边形元素只能相交在有限数量的点上",所以体表面不是多多边形

Pagis Experience Continue Continue () Deturns the underlying coordinate system of the go

Basic Functions	SpatialReference ()	Returns the underlying coordinate system of the geometry	
	Envelop ()	Returns the minimum orthogonal bounding rectangle of the geometry	
	Export ()	Returns the geometry in a different representation	
	IsEmpty ()	Returns true if the geometry is a null set	
	IsSimple ()	Returns true if the geometry is a simple (no self-intersection)	
	Boundary ()	Returns the boundary of the geometry	
Topological / Set	Equal	Returns true if the interior and boundary of the two geometries are spatially equal	
Operators	Disjoint	Returns true if the boundaries and interior do not intersect	
•	Intersect	Returns true if the geometries are not disjoint	
	Touch	Returns true if the boundaries of two surfaces intersect but the interiors do not	
	Cross	Returns true if the interior of a surface intersects with a curve	
	Within	Returns true if the interior of the given geometry does not intersect with the exterior of another geometry	
	Contains	Tests if the given geometry contains another given geometry	
	Overlap	Returns true if the interiors of two geometries have nonempty intersection	
Spatial Analysis	Distance	Returns the shortest distance between two geometries	
	Buffer	Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance	
	ConvexHull	Returns the smallest convex geometric set enclosing the geometry	
	Intersection	Returns the geometric intersection of two geometries	
	Union	Returns the geometric union of two geometries	
	Difference	Returns the portion of a geometry that does not intersect with another given geometry	
	DymmDiff	Returns the portions of two geometries that do not intersect with each other	
Andread and and	At .	I. the laterale	

Equals	$a \subseteq b, a \supseteq b$	TFFFTFFT (同类:点/点,线/线,面/面)
Overlaps	$Dim(I(a)) = Dim(I(b)) = Dim(I(a) \cap I(b)), \ a \cap b \neq a, \ a \cap b \neq b$	T*T***T** (点/点, 面/面) 1*T***T** (线/线) (同类)
Disjoint	$a \cap b = \emptyset$	FF*FF**** (点线面所有组合)
Intersects	a.Intersects(b) ←→ !b.Disjoint(b)	?(点线面所有组合)
Within	$a \cap b = a$ , $I(a) \cap E(b) = \emptyset$	T*F**F*** (除点/点,点线面所有组合)
Contains	a.Contains(b) ←→ b.Within(a)	? (除点/点,点线面所有组合)
Touches	$I(a) \cap I(b) = \emptyset, \ a \cap b \neq \emptyset$	FT*******, F**T******, F***T***** (除点/点、点/多点、多点/多 点,点线面所有组合)
Crosses	$I(a) \cap I(b) \neq \emptyset$ , $a \cap b \neq a$ , $a \cap b \neq b$	T*T****** (点/线, 点/面, 线/面) 0******** (线/线)

## 五 空间拓展 E/R 图

数据库设计基本阶段:需求分析阶段 概念结构设计阶段 逻辑结构设计阶段 物理结构设计阶段 数据 库实施阶段 数据库运行和维护阶段 数据流图 用于表达和描述系统的数据流向和对数据的处理功能 是现行系统的一种逻辑抽象,独立于系统的实现,数据字典通过对数据项和数据结构的定义来描述数据 流、数据存储的逻辑内容 是各类数据描述的集合 是进行详细的数据收集和数据分析结果

#### 六 关系数据库设计理论

第一范式 每个属性都具有原子性,是不可再被细分的,第二范式定义:如果关系模式 R 是第一范式的 而且关系中每一个非主属性不部分依赖于主键这就是一个部分函数依赖 3.第三范式在第二范式的基础 上,不存在非主属性对码的传递性依赖 4.BC 范式对于所有的非平凡依赖,左边必须是一个超码 即在 第三范式的基础上,消除了对主属性对码的部分和传递依赖 5.第四范式 和 BC 范式非常相像, 对于所 有的非平凡多值依赖,左边必须是超码 无损连接 把 R 分解为 R1 和 R2,此分解是无损的必须满足: α=R1∩R2 是 R1 或者是 R2 的超码。 依赖保持 需要验证 {F1 并 F2 并 F3} += F+, 即每一个分解出的 R 包含的函数依赖的并集的闭包和原集 F 的闭集相等 假设:需要判断是否蕴含了 $\alpha$ -> $\beta$ 函数依赖;令 Result=α开始扫描 Ri T=(Result ∩ Ri) + ∩ Ri 令 Result=Result UT Result 包含了 β,则α->β成立,算法 终止 Result 有变化、但还没包含β、继续下一轮 Result 无变化、目不包含β、则α->β不成立、分解不覆 盖 α->β. 说明不保持函数依赖。

## 七 空间存储与索引

Common Queries Point query: Find all rectangles containing a given point Range query: Find all objects within a query rectangle Nearest neighbor; Find the point closest to a query point Intersection query: Find all the rectangles intersecting a query rectangle 磁盘存储速度顺序扫描远大于随机读取 I/O cost: Number of disk sectors retrieved from secondary storage CPU cost: Number of CPU instruction used **Total cost** = sum of I/O cost and CPU cost 文件结构有 堆 无序存储。插入到最后 很快, 查找和 findnext 扫描整个文件 顺序存储 记录被按某种顺序存储 Findnext 很快, Find Insert delete 使用二分搜索 最邻近查询使用范围查询实现 哈希方法 find insert delet 很快 findnext 最邻近 慢 聚集与非聚集索引 聚集索引存储记录是物理上连续存在,而非聚集索引是逻辑上的连续、物理存 储并不连续 聚集索引插入数据时速度要慢 查询数据比非聚集数据的速度快。



cells by a grid Efficient for find, insert, nearest neighbor But may have wastage of disk storage space 缺点 索引数据冗余 网格的大小难以确定 很多网格没有数据

四叉树优缺点 与网格索引相比,四叉树保证了桶内要素不超过某个量,提高了检索效率 对于海量数 据, 四叉树的深度会很深, 影响查询效率 可扩展性不如网格索引

PostGIS 中的索引 B-trees GiST( R-Tree) BRIN(Store only the bounding box) SP-GiST

#### 八 空间查询处理与优化

DBMS 接收到 SQL 查询后一般分为四个步骤 查询分析 查询检查 查询优化 查询执行 空间数据库中 空间查询操作一般分为过滤和精炼两步 Postgre 信息输出中, 越靠右越先执行 关系R(A, B), S(B, C)

- · 关系数据库仅关注I/O cost, 假设读取一个数据块的 cost为1,内存只能缓存每个关系一个数据块
- 关系R(A, B)
- 数据元组数/行数: T(R)
- 数据块: B(R) ■ 每个数据块存储的行数: T(R)/B(R)
- 属性可取的数值个数: V(R, A), V(R, B) ■ 如果A是key, V(R, A) = ?
- σ<sub>A=a</sub>(R) (1 data block)
- 返回的行数估计为T(R) \* 1 / V(R, A)
- Hean file, 无索引·R(R)
- 聚集委引(除含sorted file), R(R) \* 1 / V(R Δ)
- 非聚集索引(文件组织?).T(R)\*1/V(R,A)

## 九 空间网络模型

维数扩展-9 交模

型(DF-9IM )不同

空间函数下的几

何关系 dim 函

数返回值 -1、0、

1 或 2 -1 表示空

- pgr createTopology
- Builds a network topology based on the geometry
- The edge\_table will be affected ■ The source column values will change
- The target column values will change
- An index will be created, if it doesn't exists, to speed up the process to the following columns
- > id, the\_geom, source, target
- OK after the network topology has been built Creates a vertices table: <edge\_table> vertices\_pgr
- Fills id and the geom columns of the vertices table
- Fills the source and target columns of the edge table referencing the id of the vertices table

## pgr\_analyzeGraph

- Analyze the network topology
- The edge table to be analyzed must contain
- a source column and a target column filled with id's of the vertices of the seaments
- the corresponding vertices table <edge\_table>\_vertices\_pgr that stores the vertices information.

varchar nor analyzeGranh(text edge table, double precision tolerand text the geom; "'the geom', text id; "'id', text source: "'source', text target: "'target', text rows\_where: "'true'

#### pgr analyzeOneway

- Analyzes oneway streets and identifies flipped segments

# pgr createTopology - The Vertices Table

- id bigint Identifier of the vertex.
- cnt integer Number of vertices in the edge table that reference

- 数据元组数/行数: T(R), T(S)

R ⋈ S (nested loop join, 2 data blocks)

B(R) + T(R) \* B(S) \* 1 / V(S, B)

B(R) + T(R) \* T(S) \* 1 / V(S, B)

B(S) + T(S) \* T(R) \* 1 / V(R, B)

\_ 数据性, B(R) B(S)

B(R) + B(R) \* B(S)

B(S) + B(S) \* B(R)

- 无索引

聚集索引

- 非聚集索引

chk integer - Indicator that the vertex might have a problem

- 属性可取的数值个数: V(R, A), V(R, B), V(S, B), V(S, C)

■ B(S) + T(S) \* B(R) \* 1 / V(R, B) Merge Join: B(R) + D(S) <= M

What if there are M data blocks

(block nested loop join)

Hash Join: B(R) + B(S)

Merge Join: B(R) + B(S)

- in integer Number of vertices in the edge\_table that reference
- this vertex AS incoming ■ eout integer - Number of vertices in the edge table that
- reference this vertex AS outgoing the\_geom geometry - Point geometry of the vertex
- prg\_createVerticesTable

- Reconstructs the vertices table based on the source and pgr analyzeGraph

# - Analyzes the network topology

- edge\_table: text Network table name. (may contain the schema name as well) ■ tolerance: float8 Snapping tolerance of disconnected edges. (in
- projection unit) the geom; text Geometry column name of the network table.
- Default value is the geom id: text Primary key column name of the network table. Default
- value is id source: text Source column name of the network table. Default
- value is source target: text Target column name of the network table. Defaul
- rows where: text Condition to select a subset or rows. Default
- value is true to indicate all rows

#### 十 数据库安全性与完整性

SQL 权限控制 在指定的表、视图、存储过程或函数上执行特殊动作的权利。对于不同类型的对象,有 不同类型的对象特权 对于有些对象,如索引、触发器等,没有相关的对象特权 它们由系统特权控制 假设用户王平创建了基本表 S、C 和 SC,则他自动获得对这些表的所有权限(包括将这些权限传播给其 他用户的权力) GRANT INSERT, DELETE ON SC TO 李霞 WITH GRANT OPTION 执行此 SQL 语句后,用 户李霞不仅拥有了对表 SC 的 INSERT 和 DELETE 权限, 还可以传播这些权限

触发器 Trigger SQL 标准写法 Create Trigger name Before | After | Instead Of events [referencing-variables] [For Each Row] When (condition) action Create Trigger Trigger1 After Delete On S Referencing Old Row As O For Each Row [No condition] Delete From R where A = O.B

视图的特点 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 视图的作用 视图能够简化用户的操作 视图 使用户能以多种角度看待数据 视图对重构数据库提供逻辑独立性 视图能够对机密数据提供安全保护 撤销权限写法 Revoke privs On R From users [Cascade | Restrict]

- Nodes an network edge table

pgr createTopology function

pgr\_createTopology function

PostgreSQL可更新视图标准

the view's select list.

be a table or another updatable view

INTERSECT or EXCEPT) at the top level

network

- id: bigint Unique identifier for the table

- old\_id: bigint Identifier of the edge in original table

- sub id: integer Segment number of the original edge

- source: integer Empty source column to be used with

- target: integer Empty target column to be used with

- the geom: geometry Geometry column of the noded

HAVING, LIMIT, or OFFSET clauses at the top level.

- The view must not have the security\_barrier property

The view definition must not contain set operations (UNION

- All 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.

No column of the underlying relation can appear more than once in

The view must have exactly one entry in its FROM list, which must

The view definition must not contain WITH, DISTINCT, GROUP BY

视图可以建立在 单个表 多个表 (基于多个基表的视图) 一个或多个视图 (基于视图的视图)表和视图 pgr nodeNetwork

# pgr\_nodeNetwork

- Node an network edge table
- edge\_table: text Network table name. (may contain the The output table will have for edge\_table noded schema name as well)
- tolerance: float8 tolerance for coincident points (in projection unit)
- id: text Primary key column name of the network table. Default value is id
- the\_geom: text Geometry column name of the network table. Default value is the geom
- table ending; text Suffix for the new table's. Default value is noded

pgr\_dijkstra(text edges\_sql, bigint start\_vid, bigint end\_vid, boolean directed:=true) SQL Server可更新视图标准

- 任何修改(包括 UPDATE、INSERT 和 DELETE 语句) 都只能引用一个基表的列
- 视图中被修改的列必须直接引用表列中的基础数据。不 能通过任何其他方式对这些列进行派生, 如通过以下方式 ■ 聚合函数: AVG、COUNT、SUM、MIN、MAX、GROUPING
- STDEV, STDEVP, VAR RI VARP. 计算。不能从使用其他列的表达式中计算该列。使用集合运算 UNION, UNION ALL, CROSSJOIN, EXCEPT #
- INTERSECT 形成的列格计入计算结果, 日不可更新 被修改的列不受 GROUP BY、HAVING 或 DISTINCT 子
- TOP 在视图的 select statement 中的任何位置都不会与
- WITH CHECK OPTION 子句一起使用。

## 十二 事务处理

### 调度S是可串行化的如果S和某些串行调度是冲突等价的

冲突图 考虑节点为 TXN 的图形,考虑事务 Ti,Tj,则如果 Ti 的任何操作在 Tj 中的任何动作之前而且 发生冲突,则具有边缘 Ti →Ti 当且仅当其冲突图为无环时,调度是可以序列化的

两段锁协议(2PL) 算法: 严格的两相锁定 - 作为处理并发的一种方式 保证冲突序列化(从概念上) 简单地实现, 并对用户诱明

**封锁** 所谓封锁就是事务 T 在对某个数据对象例如表、记录等操作之前,先向系统发出请求,对其加锁。 加锁后事务T就对该数据对象有了一定的控制,在事务T释放它的锁之前,其他的事务不能更新此数 据对象。基本的封锁类型有两种: 1.排它锁 (Exclusive Locks),简称 X 锁 2.共享锁 (Share Locks),

简称S锁 排它锁又称写锁。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,其他任何事务都不能

共享锁又称读锁。若事务 T 对数据对象 A 加上 S 锁,则事务 T 可以读 A 但是不能修改 A,其他事务只 能再对A加S锁,而不能加X锁,直到T释放A上的S锁。

# 两段锁协议

在运用封锁方法时,对数据对象加锁时需要约定一些规则,例如何时申请封锁、持所时间、何时释放 封锁等。这些规则被称为封锁协议。

所谓两段锁协议是指所有事务必须分两个阶段对数据项加锁和解锁

1.在对任何数据进行读、写操作之前,首先要申请并获得对该数据的加锁

## 2.在释放一个封锁之后,事务不再申请和获得任何其他封锁

事务遵守两段锁协议是可串行化的充分条件,而不是必要条件、遵守两段锁协议的事务可能发生死锁、 严格两段锁协议: 只能在事务提交或者取消的时候才能释放锁

#### 如果调度是严格两段锁的,则其是冲突可串行化的

再对 A 加任何类型的锁, 直到 T 释放 A 上的锁。

死锁 事务对锁释放的循环等待 处理死锁的两种方式 死锁预防 死锁检测

等待图 节点是事务 如果 Ti 在等待 TJ 释放锁,则有 Ti→Tj

脏读: 也就是当数据库的一个事务 A 正在使用一个数据但还没有提交, 另外一个事务 B 也访问到了这 个数据, 还使用了这个数据, 这就会导致事务 B 使用了事务 A 没有提交之前的数据。

不可重复读: 在一个事务 A 中多次操作一个数据, 在这两次或多次访问这个数据的中间, 事务 B 也操 作此数据, 并使其值发生了改变, 这就导致同一个事务 A 在两次操作这个数据的时候值不一样, 这就 是不可重复读。

幻读: 是指事务不独立执行产生的一种现象。事务 A 读取与搜索条件相匹配的若干行。事务 B 以插入 或删除行等方式来修改事务 A 的结果集,然后再提交。这样就会导致当 A 本来执行的结果包含 B 执行 的结果,这两个本来是不相关的,对于 A 来说就相当于产生了"幻觉"。

#### 数据库的四种隔离级别 (并发事务)

## (一) 可读取未提交 (Read uncommitted

写事务阻止其他写事务、避免了更新遗失。但是没有阻止其他读事务。

存在的问题: 脏读。即读取到不正确的数据,因为另一个事务可能还没提交最终数据,这个读事务就 读取了中途的数据。这个数据可能是不正确的。

解决办法就是下面的"可读取确认"。

#### (二) 可读已提交 (Read committed)

Sql Server, Oracle 的默认隔离级别

写事务会阻止其他读写事务。读事务不会阻止其他任何事务。 存在的问题:不可重复读。即在一次事务之间,进行了两次读取,但是结果不一样,可能第一次 id 为 1的人叫"李三",第二次读 id 为 1 的人就叫了"李四"。因为读取操作不会阻止其他事务。 解决办法就是下面的"可重复读"。

## (三) 可重复读(Repeatable read

#### MySOL 的默认隔离级别

读事务会阻止其他写事务、但是不会阻止其他读事务。

存在的问题: 幻读。可重复读阻止的写事务包括 update 和 delete (只给存在的表加上了锁), 但是不 包括 insert (新行不存在, 所以没有办法加锁), 所以一个事务第一次读取可能读取到了 10 条记录 但是第二次可能读取到 11 条、这就是幻读。解决办法就是下面的"串行化"。

#### (四) 序列化 (Serializable)

可避免幻读。读加共享锁,写加排他锁。这样读取事务可以并发,但是读写,写写事务之间都是互斥 的,基本上就是一个个执行事务,所以叫序列化。

WAL 预写日志,是数据库系统中常见的一种手段,用于保证数据操作的原子性和持久性。

#### 关系代数相关

2.1 查找至少选修一门4学分及以上课程的学生学号(2分)

答案:  $\Pi_{sid}(\sigma_{credit>4}(SC \bowtie Course))$ 

2.2 查找学生'张三'或'李四'洗修过的课程名(2分)

答案:  $\Pi_{C.name}(\sigma_{(Student.name='**E') \land Student.name='**E') \land Student.sid=C.sid}(\rho_C(SC \bowtie Course) \times Student))$ 

2.3 查找学生'张三'和'李四'都选修过的课程名 (2分)

答案:  $\Pi_{B.cid}(\sigma_{B.name-'}$ 季西 $(\sigma_{B.cid-A.cid}(\rho_B(SC \bowtie Student) \times \rho_A(\sigma_{name-'} \otimes \Xi'(SC \bowtie Student)))))$ 

2.4 直找所有学生成绩都大于等于80分的课程号,注意有的课程可能没有学生选修,如刚开设的课程(2分)

答案:  $\Pi_{cid}(Course \bowtie \rho_A(Course - \Pi_{cid,name,credit}(Course \bowtie (\sigma_{grade < 80}(SC)))))$ 

2.5 查找:地理空间数据库(cid='06122870')成绩最高的学生学号、假设该课程成绩都不相同(2分)

答案:  $\Pi_{sid}(SC) - \Pi_{A.sid}(\rho_A(\sigma_{cid='06122870'}(SC))) \bowtie_{A.grade} < B.grade \rho_B(\sigma_{cid='06122870'}(SC)))$ 

## 复合主键创建

result1 = %sql #query1

query2 = """
SELECT gid, name, g
FROM uslakes as Li
WHERE EXISTS(
SELECT "

- R1(C, D, E)

FROM uslakes AS L2, ushighways WHERE L1.gid = L2.gid AND ST\_Crosses(L2.geom, ushighways.geom)

选择违背BC范式的函数依赖CD→E进行分解

■ R2基于油費RC指式的CD→A进行分類

> R3(C, D, A)和R4(C, D, B)

■ R1的函数依赖为CD→E, 且码为CD, R1属于BCNF

R2(C, D, A, B)
 R2不能选择BC→D继续分解。因为(BC)\* = ABCD

■ R2的函数依赖为AB→C, BC→D, CD→A, R2的码AB和BC ➤ CD → A是基于R的函数依赖获得,只和ACD相关,与E无关

➤ R3的函数依赖CD→A,且码为CD,R3属于BCNF

> R4的函数依赖BC→D。且码为BC。R4属于BCNF

➤ 例如id → sno; sno → name。R'(id, name)存在id → name

直向被公司穿越的湖

```
%%sql drop table if exists weather;
                                                                                                                SELECT DISTINCT uscities.name // 'in ' // uscities.state AS nam
CREATE TABLE weather (
date date not null,
max_temp real,
mean_temp real,
                                                                                                                    uscities,
      min_temp real,
                                                                                                                        FROM usaccident
      max visibility miles real,
     max_visibility_miles real,
mean_visibility_miles real,
min_visibility_miles real,
max_wind_speed_mph real,
max_gust_speed_mph real,
cloud_cover_real,
events text,
wind directors real
                                                                                                                    WHERE ST_CASE = 10012
) AS accident
                                                                                                                 WHERE ST_Distance(uscities.geom::geography, accident.geom::geography) >= all
                                                                                                                    SELECT ST Distance(uscities.geom::geography, accident.geom::geography
                                                                                                                        uscities
                                                                                                                         SELECT *
FROM usaccidents
WHERE ST_CASE = 10012
) AS accident
     wind_dir_degrees real,
zip_code text not null,
constraint pk_weather p
                                          primary key (date, zip code
                                                                                                                         SELECT A.gid AS gid, A.full name AS name, A.geom AS geom
                                                                                                                        FROM ushighways AS A, ushighways AS B
WHERE A.gid \leftrightarrow 94 AMD B.gid = 94 AMD ST_Intersects(A.geom, B.geom);
                                                                                                                     result1 = \S eq1 \S equery result1 = \S eq1 select gid, geom, full_name as name from ushighways where gid = 94
  (SELECT
EXTRACT(MONTH FROM date) AS month, avg(number) AS number
FROM(
                                                                                                                    display/[result] result2] "man5" 51
                                                                                                                    * 亚邦拉长度
          SELECT
DATE(start_time) AS date, count(*) AS number
        read (SELECT + FMON trip, station WHERE EXTRACTIVEAR FROM start_time) = 2014 AND station_id = start_s AND sta_code = "941807" AD T GUIDS TO #441) AS T
                                                                                                                     SELECT DISTINCT SUM(ST Length(geom::geography) * 1E-3) AS length
                                                                                                                         SELECT Augeon AS geon
                                                                                                                         WHERE & gid () 94 AND R gid = 94 AND ST Tetarcorto(& garm. R garm.
                                                                                                                    %sql $query2

    关系R(A, B, C, D, E)具有以下函数依赖: AB→C.

# STREET WASHINGTON
                                                                               BC→D, CD→E, DE→A。将关系R分解为BC范式,
query1 = """
SELECT gid, full_name AS name, geom
FROM ushighways as Hi
                                                                               需先给出关系R的码,哪些函数依赖违背了BC范式
                                                                               , 基于哪个函数依赖将哪个关系进行分解。是否存
        PROM uslakes, ushighways as H2
WHERE H1.gid = H2.gid AND ST_Crosses(H2.geom, uslakes.geom)
                                                                               在不同的BC范式分解,若存在,请给出其分解。
                                                                           关系R的key: AB和BC
```

```
    关系R(A, B, C, D, E)具有以下函数依赖: AB→C.

    R的码: AB和BC, 违背BCNF: CD → E, DE → A

                                        BC→D, CD→E, DE→A
```

■ AB, ABC, ABD, ABE, ABCD, ABDE, ABCDE是superkey

一种BCNF分解

- (AB)+ = ABCDE

- (BC)+ = ABCDE

– (BDE) \* = ABCDE

\_ (CD)+ = CDEA

- (DE)+ = DEA

- R1(C, D, E), R3(C, D, A)和R4(C, D, B)
- 丢失了函数依赖AB→C, DE→A
- 另一种BCNF分解
- R1(D, E, A), R3(C, D, E)和R4(C, D, B)
- 丢失了函数依赖AB→C
- 第六章 BCNF分解 选择不同R'和FD进行分解,BC 范式分解结果可能不同

```
Wsql
                                          WITH RECURSIVE
                                             link with name AS
                                                 SELECT start_mode_id AS start_mode_id, start_mame, end_mode_id, airport_mame AS end_mame
                                                FR0M(
                                                   SELECT start_node_id, airport_name AS start_name, end_node_id
                                                   FROM airport_link , airport_list
                                                   NHERE start_node_id = airport_id) AS link_start,
create table AIRPORT_NODE
                                                   airport_list
  NODE_ID INT PRIMARY KE
NODE_NAME VARCHAR(200),
NODE_TYPE VARCHAR(200),
ACTIVE VARCHAR(1),
                                                WHERE end node id = airport id
                                             all airport(start mode id, start mame, depth) AS(
  GEOMETRY geometry(POINT, 4326)
                                                FROM link with name
                                                UNION ALL
create table AIRPORT_LINK
                                                SELECT DISTINCT link_with_name.end_node_id AS end_node_id, link_with_name.end_name AS end_name, depth + 1 AS
  LIMK_ID
 LINK_LAME VARKHR(200),
START_MODE_ID INT NOT NULL,
END_MODE_ID INT NOT NULL,
LINK_TYPE VARKHR(200),
ACTIVE VARKHR(1),
LINK_LEVEL INT,
                                                FROM all_airport INNER JOIN link_with_name ON link_with_name.start_node_id = all_airport.start_node_i
                                                WHERE depth < 1
              geometry(MultiLineString, 4326),
                                          SELECT DISTINCT start node id. start name AS airport name
                                           FROM all_airport;
  BIDIRECTED
              VARCHAR(1)
2.3 查询哪些机场最多一次转机能够达到"Bethel, AK"机场的机场名称和AIRPORT_ID,使用With Recursive实现。(2分)
    WITH RECURSIVE
        link_with_name AS(
            SELECT start_node_id AS start_node_id, start_name, end_node_id, airport_name AS end_name
                SELECT start_node_id, airport_name AS start_name, end_node_id
                FROM airport_link , airport_list
                WHERE start_node_id = airport_id) AS link_start,
                airport_list
            WHERE end_node_id = airport_id
        all_airport(end_node_id, end_name, depth) AS(
            FROM link_with_name
            WHERE end_name = 'Bethel, AK'
            UNION ALL
            SELECT DISTINCT link_with_name.start_node_id AS start_node_id, link_with_name.start_name AS start_name, depth
            + 1 AS depth
            FROM all_airport INNER JOIN link_with_name ON link_with_name.end_node_id = all_airport.end_node_id
            WHERE denth < 2
    SELECT DISTINCT end_node_id, end_name AS airport_name
    FROM all_airport;
   %%sal
   INSERT INTO edges(id, name, geom, len)
   SELECT id, name, geom, ST length(geom)
   FROM roads;
    SELECT pgr_createTopology('edges', 0.0001, 'geom', 'id', 'source', 'target', 'true');
   SELECT pgr analyzeGraph('edges',0.0001, 'geom', 'id', 'source', 'target', 'true');
    SELECT pgr_nodeNetwork('edges',0.0001,'id','geom');
    SELECT pgr_createTopology('edges_noded', 0.0001, 'geom', 'id', 'source', 'target', 'true');
    SELECT pgr_analyzeGraph('edges_noded', 0.0001, 'geom', 'id', 'source', 'target', 'true');
      FROM pgr dijkstra(
                 'SELECT id AS id, source AS source, target AS target, len AS cost FROM edges'
                 422.
                 1944
                FALSE
%%sal
drop view if exists currenttrack:
create view CurrentTrack
        SELECT DISTINCT A.carID AS carID, A.position AS position, edges.id AS roadID
            SELECT track.carID, track.position, min(ST_DistanceSphere(position,edges.geom)) AS distance
            FROM (
                 SELECT carID, max(time) AS time
                 FROM track
                 GROUP BY carID
            ) AS B INNER JOIN track ON track.time = B.time AND track.carID = B.carID
            GROUP BY track.carID, track.position
        WHERE A.distance = ST_DistanceSphere(A.position,edges.geom)
CREATE OR REPLACE FUNCTION auditlogfunc() RETURNS TRIGGER AS Sexample tableS
          Insert INTO track(carID, time, position, username
           SELECT distinct new.carID, CURRENT_TIMESTAMP(0), new.position, track.username
           FROM track
           WHERE new.carID = track.carID;
           RETURN NEW;
Sexample tables LANGUAGE plpgsql:
 Create Trigger Trigger1
 INSTEAD OF Insert On CurrentTrack
For Each Row
EXECUTE PROCEDURE auditlogfunc():
```