数据库整理

lyl

- 1 intro: 数据库的作用/DBMS的定义/两种data model/数据独立性data independence/concurrency control并发控制(使用加锁协议)/crash recovery(存储备份)/数据库的分层架构以及其应用
  - 数据库的作用
  - DBMS定义
  - 两种data model
  - 数据独立性data indenpendence
  - 并发控制
  - crash recorvery
  - 数据库的分层架构
- 2 E/R: 实体,实体集,属性/ER图的实体,属性,关系(多对多,多对一,一对一),关系集,关系的属性,弱实体集,键,子类/关系的角色roles/子类,ER子类与OO子类的区别/键keys/弱实体集定义,两个约定/数据库设计的三个法则,数据冗余,使用实体而非属性的两种情况/关系可合并的两种情况,RE图转关系模型:实体的变换,属性的变换,关系的变换,弱实体集的变换,带有子类的ER图的三种转换法
  - 实体,实体集,属性
  - E/R图的实体,属性,关系(多对多,多对一,一对一),关系集reletionship set
  - 关系的角色role
  - 子类,ER子类与OO子类的区别
  - 键
  - 弱实体集weak entity定义, 两个约定
  - 数据库设计的三个法则, 数据冗余,使用实体而非属性的两种情况
  - 关系可合并的两种情况,RE图转关系模型:实体的变换,属性的变换,关系的变换,弱实体集的变换,带有子类的ER图的三种转换法
- 3 fd: 函数依赖functional dependence的定义,右部分解/键与超键/依赖的推导,闭包closure的推导,利用closure判断依赖是否存在,隐含依赖的推导,规范化normalization, nontrival fd非平凡函数依赖/依赖的投影projected FD,求依赖投影的简单指数算法exponential algorithm,两个技巧/3种异常, 判断无损分解, 分解的评判标准与chase检测/BCNF的定义,BC的分解/minimal basis最小函数依赖集的求法,prime主属性3NF的定义,3NF的合成(3NF synthesis),BC和3NF的比较/完全函数依赖(full function dependence)/2NF,1NF的定义与判断
  - 函数依赖functional dependence的定义,右部分解
  - 键与超键
  - 依赖的推导,闭包closure的推导,利用closure判断依赖是否存在,隐含依赖的推导, nontrival fd非平凡函数依赖
  - 依赖的投影projected FD,求依赖投影的简单指数算法exponential algorithm,两个技巧
  - 3种异常, 判断无损分解, 分解的评判标准与chase检测
  - BCNF的定义,BC的分解
  - minimal basis的求法,prime,3NF的定义,3NF的合成(3NF synthesis)
  - 完全函数依赖(full function dependence)
  - 2NF,1NF的定义与判断
- 3_2 fd的公理系统: 基本符号:关系模式,函数依赖的闭包,属性集的闭包,最小函数依赖集/Armstrong的3条规则和3个推理/闭包的计算,求闭包算法的循环次数/FD等价与FD覆盖
  - 基本符号: 关系模式,函数依赖的闭包,属性集的闭包
  - Armstrong的3条规则和3个推理

    - 闭包的计算,求闭包算法的循环次数
    - FD等价与覆盖
- 4 MVD多值依赖: MVD的定义,平凡多值依赖与非平凡多值依赖,MVD的6个性质,MVD与FD的关系/4NF的定义,4NF与BCNF的对比,4NF的分解/MVD或FD是否存在的判断(使用表格,或利用传递性)
    - MVD的定义,平凡多值依赖与非平凡多值依赖,MVD的6个性质,MVD与FD的关系
    - 4NF的定义,4NF与BCNF的对比,4NF的分解
    - MVD或FD是否存在的判断(使用表格,或利用传递性),规律小结
- 5 Relational Algebra (ra 关系代数) : relational algebra的定义,操作数operand与算子operation/核心关系代数:selection,projection,product and join,rename,执行结果/表达式的构建,表达式的三种写法,算符优先级/包bag定义,与集合的区别,包与集合上操作的不同之处
    - relational algebra关系代数的定义,操作数operand与算子operation
    - 核心关系代数:selection,projection, product and join, rename,执行结果
    - 表达式的构建,表达式的三种写法,算符优先级
    - 包bag定义,与集合set的区别,包与集合操作的不同之处
- 6 数据库-连接库: SQL注入/基于库libraries的SQL接口/三层架构/php游标cursors
    - 基于libraries的SQL接口
    - 数据库环境的三层结构:
    - 基于libraries的SQL接口
    - php游标cursors
- 7 SQL intro: select-from-where/使用as重命名/模糊查询的两种模式/空值null的两种含义/三值逻辑/多重关系查询:定义,方法/self-join查询同一关系中的两个不同对象/子查询描述,返回值/in(与普通query的区别),exists,any,all,union,intersect,except用法,语句的值,用在哪里/基于包执行的操作,基于集合执行的操作/消除重复项duplicate elimination--distinct/join操作: 笛卡尔积,自然连接,theta连接
    - 模糊查询的两种模式
    - 空值null的两种含义
    - 三值逻辑
    - 多重关系查询Multirelation query定义,方法
    - self-join查询同一关系中不同的两个对象
    - 子查询subquery描述,返回值
    - in,exists,any,all,union,intersect,except用法,语句的值,用在哪里
    - 基于包执行的操作,基于集合执行的操作
    - 消除重复项duplicate elimination--distinct
    - join操作: 笛卡尔积,自然连接,theta连接
- 8 ra-sql: 拓展关系代数:包去重,排序/悬挂元组dangling,内连接,外连接:左外连接,右外连接,全连接/聚集aggregation:sum,max,min,avg,count,以及与distinct的结合,聚集中的空值/分组,使用分组对select进行限制,分组中的聚集函数,使用分组的两个必须条件/having的使用以及规则,后面跟随的条件/数据库插入(普通插入,批量插入,利用子查询插入),删除(全部删除,部分删除),更新(全部更新,条件更新),缺省
    - 拓展RA:包去重,排序,分组聚集
    - 悬挂元组dangling tuple,内连接,外连接中悬挂元组的表示,外连接:left outer join, right outer join, full outer join
    - aggregation:sum,max,min,avg,count,在聚集函数中使用distinct,聚集函数中元组null的情况
    - 分组,使用分组对select进行限制,分组中的聚集函数,使用分组的两个必须条件
    - having的使用以及规则,后面跟随的条件

    - 数据库插入(普通插入,批量插入,利用子查询插入),删除(全部删除,部分删除),更新(全部更新,条件更新),缺省
- 9 约束constraint: 约束的定义,种类/外键的定义,外键表达方式,维护外键约束的3种方式(default,cascade,Null(缺省,级联,置空))以及选择策略/三种检测(attribute-based, Tuple-based, assertion断言)的方式,时机/触发器trigger描述,使用时机,创造规则:event-condition-action
    - 约束的定义,种类
    - foreign key外键的定义,外键表达方式,维护外键约束的3种方式(default,cascade,null(缺省,级联,置空))以及选择策略
    - 三种检测(attribute-based,tuple-based,assertion)的方式,时机
    - 触发器trigger描述,创建,使用时机,创造规则,event-condition-action规则
- 10 transaction-view-index事物-视图-索引: transaction定义,ACID,导致事物结束的两种情况:commit,rollback/回滚rollback:导致回滚的情况,解决方法/sql4个隔离层级isolated level,选择/事物序列化的定义,操作/view的定义,种类,声明,结合触发器的视图,基表更改引起实例化视图更改问题以及解决方法/index索引定义,声明,使用索引优化数据库查询tuning的优点,缺点
    - transaction定义,ACID,commit
    - roll back:导致回滚的情况,解决方法
    - sql4个隔离层级isolated level,选择
    - 视图的定义,两种类别,声明,结合触发器的视图,视图实例化,基表更改引起实例化视图更改问题的解决方法
    - index索引定义,声明,使用索引优化数据库查询tuning的优点,缺点
- 11 psm持久型存储模块(存储过程),pl与sql: psm定义,参量的三种类型,声明,invoke调用,语法:判断,循环,指针,return/动态SQL声明,调用
    - psm定义,参量,声明,invoke,3个基本种类及作用,语法:判断,循环,指针
    - 动态SQL的声明,调用
- 12 grant授权: 语法:授权(操作权限,授权权限),撤销授权revoke,撤销授权的两种选项/授权图的点,边,AP,P*,P**,授权规则,撤销授权规则
    - 语法:授权(操作权限,授权权限),撤销授权(操作权撤销,授权权撤销),撤销授权的两种选项
    - 授权图的点,边,AP,P*,P**,授权规则,撤销授权规则
- 13 concurrency 并发控制: 基本概念:transaction事物,conflicting action冲突行为的种类,schedule调度,调度的4种方式/并发事务运行存在的3个异常/冲突等价conflict equivalent的定义/precedence graph符号表示,前驱图的节点,边,两个定理/(一级)加锁解锁协议:符号,legal schedule合法调度和well-formed调度/2PL两阶段锁协议描述,避免数据出错的其他4种方法(共享锁,多粒度,插入删除,其他机制),共享锁和排他锁描述,3个法则(well-formed,legal,upgrade变化问题)/increment lock增量锁和update lock的描述,符号/锁的兼容性/系统一个解决并发多用户加解锁问题的方法/多粒度封锁granularity:多粒度封锁描述,锁的类型((主要是意向锁:)IS,IX,SIX),6个规则,兼容性,加锁对象上允许再加的其他锁/pehantom数据重影定义,解决方法
    - 基本概念:transaction,conflicting action的种类,schedule,调度的四种方式
    - 并发事务运行存在的3个异常
    - 冲突等价conflict equivalent定义
    - precedence graph前驱图的符号表示,节点,边,两个定理
    - 加解锁协议:符号,legal schedule和well-formed
    - 2PL:描述,避免数据出错的其他4种方法(共享锁,多粒度,插入删除,其他机制),共享锁和排他锁描述,3个法则(well-formed,legal,upgrade变化问题)
    - increment lock和update lock描述,符号
    - 锁的兼容性
    - 系统一个解决并发多用户加解锁问题的方法

1 intro: 数据库的作用/DBMS的定义/两种data model/数据独立性data independence/concurrency control并发控制(使用加锁协议)/crash recovery(存储备份)/数据库的分层架构以及其应用

数据库的作用

1. **store** huge amount of data over a long time
2. allow apps to **query and update** data
3. protect from **unauthorized** access
4. protect form **system crash**
5. protect from **incorrect input**
6. support **concurrent access**
7. allow administrator to change data
8. different database operation

DBMS定义

a software to store and manage data, so applications don't have to worry about them

- DDL
- DML
- storage management
- transaction management(交易/事物管理)
- security, efficiency, scalability(可扩展性)

两种data model

- E/R
- relational model(表, 元祖...)

数据独立性data indenpendence

DBMS的分层架构包括一些views, 一个概念层conceptual schema, 一个物理层physical schema. 各层之间改变某一层的数据仅对其上层造成影响

并发控制

- 解决: 加锁locking protocol

crash recorvery

- 解决: 使用日志log, 所有对数据的写操作记录在日志中,存在稳定的存储系统里(stable storage)

数据库的分层架构

- 二层: ODBC, JDBC
- 三层: 基于Web的应用程序

2 E/R: 实体,实体集,属性/ER图的实体,属性,关系(多对多,多对一,一对一),关系集,关系的属性,弱实体集,键,子类/关系的角色roles/子类,ER子类与OO子类的区别/键keys/弱实体集定义,两个约定/数据库设计的

三个法则,数据冗余,使用实体而非属性的两种情况/关系可合并的两种情况,RE图转关系模型:实体的变换,属性的变换,关系的变换,弱实体集的变换,带有子类的ER图的三种转换法

实体,实体集,属性

- 实体entities: "thing" or object
- 实体集 entity set: 实体的集合
- 属性attribute: property(属性) of an entity set

E/R图的实体,属性,关系(多对多,多对一,一对一),关系集 reletionship set

- 实体: 矩形
- 属性: 椭圆
- 关系: 菱形, 直线连接形成关系的实体
  - 多对多: 不带箭头(一个酒吧卖很多种酒, 一种酒杯很多酒吧卖)
  - 多对一: 指向一(一个人只有一款最喜欢的酒, 一款酒可以被很多人最喜欢)
  - 一对一: 双向箭头(一个人只有一个老婆,反之也是)
- 关系集: 一张表, 里面有关系中各属性的值, 一行为一个元组, 可以有多行
- 键: 在相应属性下方画横线
- 关系的属性: 由形成关系的双方共同确定
- 弱实体集: 箭头指向弱实体集所依赖的实体
  - 实体: 双框矩形
  - 关系: 双框菱形
  - 键: 在代表键的属性下画横线, 包括自己的键和依赖实体集的键
- 子类: 用带isa三角形的箭头从子类指向父类
  - 实体: 矩形
  - 属性: 椭圆,只标明子类特有属性
  - 键: 只有基类实体有键, 并且该键作用于整个继承机制的所有类

关系的角色role

在ER图的边旁标识, 对关系中多次出现的实体进行定义

子类,ER子类与OO子类的区别

- 子类 = 特殊情况 = 更少的实体, 更多的属性
- ER子类与OO子类的区别
  - OO: 子类继承自父类,其对象实体只属于一个类(objects are in one class only)
  - ER: 子类的对象在子类和父类中都存在(E/R entity has representatives in all subclasses which they belong)

键

- 键: 键是关系集中的一组属性, 若两个元组的键值相同, 则其他非键属性必定相同

> A key is a set of attributes for one entity set such that no two entities in this set agree on all the attributes of the key.

弱实体集weak entity定义, 两个约定

- 弱实体集: 为了唯一标识一个实体集中的实体, 需要依赖其本身的键以及一个或多个多对一关系中的另一个实体集的键.
- 约定:
  1. 弱实体集与依赖的实体集之间存在一个或多个多对一关系
  2. 弱实体集的键包括其本身的键和所依赖的实体集的键

数据库设计的三个法则, 数据冗余,使用实体而非属性的两种情况

- 数据库设计的三个法则
  1. 避免冗余(avoid redundancy)
  2. 尽量避免弱实体集的使用(limit the use of weak entity sets)
  3. 能用属性就不用实体(don't use an entity set when an attribute will do)
- 数据冗余: 使用不同的方法描述同一件事物(saying the same thing in two different ways)
- 使用实体的两种情况
  1. 包含的属性不仅仅是名字,或含有至少一个非主属性
  2. 在多对多或多对一关系中属于"多"

关系可合并的两种情况,RE图转关系模型:实体的变换,属性的变换,关系的变换,弱实体集的变换,带有子类的ER图的三种转换法

- 关系可合并两种情况
  - 同一实体集中的关系可合并
  - 多对一关系中的"多"可合并
    - ```
      Drinkers(name, addr)
      Favorite(drinker, beer)
      合并为:
      Drinker1(name, addr, favbeer)
      ```
      ```

- ER图转关系模型
  - 实体集 -> 关系
  - 关系 -> 关系(仅包含构成关系的各实体集的键)
  - 弱实体集 -> 关系:
    - 用实体表示关系而非关系表示关系
    - 形成弱实体集的关系, 其依赖的实体集的关系
    - 弱实体集形成的关系属性包含弱实体集的全部属性和其依赖的实体集的键
  - 带有子类的ER图转换的3种方法:
    1. OO: 每个子类都是一个关系, 包含父类以及子类自己的全部属性
    2. use Null: 只有一个关系, 包含子类和父类的所有属性, 其中父类对象不具有的子类属性用null代替
    3. ER: 一个子类一个对象, 包含父类的键值和子类自己的全部属性

3 fd: 函数依赖functional dependence的定义,右部分解/键与超键/依赖的推导,闭包closure的推导,利用closure判断依赖是否存在,隐含依赖的推导,规范化normalization, nontrival fd非平凡函数依赖/依赖的投影projected FD,求依赖投影的简单指数算法exponential algorithm,两个技巧/3种异常, 判断无损分解, 分解的评判标准与chase检测/BCNF的定义,BC的分解/minimal basis最小函数依赖集的求法,prime主属性3NF的定义,3NF的合成(3NF synthesis),BC和3NF的比较/完全函数依赖(full function dependence)/2NF,1NF的定义与判断

函数依赖functional dependence的定义,右部分解

- 函数依赖: 在关系R中, 如果当属性集X中所有属性值相同时, 属性集Y中的所有属性值也相同, 称关系R中存在函数依赖$X->Y$,简写作FD

> X ->Y is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X, then they must also agree on all attributes in set Y.
>
> - Say “X ->Y holds in R.”
> - Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
> - Convention: no set formers in sets of attributes, just ABC, rather than {A,B,C }.

- 函数依赖的右部分解: 右部可分左部不可分, 通常右部表示为单一属性集

键与超键

- $superkey$: 由$superkey$中属性可推出关系的全部属性
- key: 最小$superkey$集

依赖的推导,闭包closure的推导,利用closure判断依赖是否存在,隐含依赖的推导, nontrival fd非平凡函数依赖

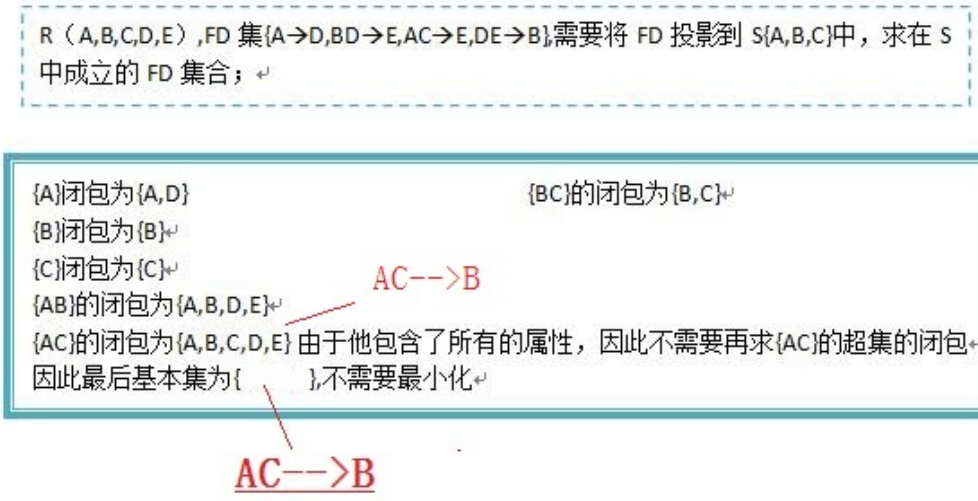
- 依赖的推导: (1)根据传递性($X->Y, Y->Z; => X->Z$) (2)或找X值相同时值相同的属性,看是否可以用已知的FD推导出来, 是则存在依赖, 不是则不存在
- closure的计算: 可以使用闭包判断属性集之间是否存在依赖关系
  - 求属性集$\{A_1, A_2, ..., A_n\}$的闭包, 已知FD集S:
    1. 假设U终将成为$\{A_1, A_2, ..., A_n\}$的闭包,初始化$U = \{A_1, A_2, ..., A_n\}$,分解S中每个FD, 使其右部属性单一化
    2. 重复搜索这样的函数依赖$B_1B_2B_3...B_i->C$, 使左部$\{B_1B_2B_3...B_i\}$都在U中但是右部不在, 则将右部加到U中
    3. 重复上述过程知道没有属性可以加到U中, 则此时的U就是$\{A_1, A_2, ..., A_n\}$的闭包
- 利用closure判断依赖是否存在: 判断某依赖A->B是否存在关系X中, 只要求出X中A的闭包, 看B是否出现在闭包中
- 隐含依赖的推导: 看属性集中是否存在可由已知FD推导出的其他FD
- Nontrival FD非平凡函数依赖: 左部不包含右部
- normalization函数依赖规范化:

依赖的投影projected FD,求依赖投影的简单指数算法exponential algorithm,两个技巧

- $projected\_FD$: 给定一个关系和满足的依赖集, 求对部分属性投影后依然成立的依赖集
- 求依赖投影$Exponential_Algorithm$

> - 输入: 关系R, 关系R的投影$R_i = \pi_L(R)$, 在R中成立的FD集合S
> - 输出: 在R_i中成立的FD集合S_i
>
> 1. 设T为最终输出的FD集, 初始化为空
> 2. 对于R_i属性集和的每一个子集X_i, 根据R中的FD计算${X_i}^+$
>   (此时, 计算结果可能包含不属于R_i的属性)
> 3. 找出其中右部A在${X_i}^+$中且属于R_i的所有非平凡函数依赖$X->A$, 添加到T中
> 4. 最小化T : 不断重复下列两个步骤知道T不变
>   - 如果T中某个依赖F能从T中其他FD推导出来, 则删除F
>   - 如果T中有$Y->B$, Y中至少有两个属性, 删除Y中一个属性Z,
>   并发现由T中其他FD可以推出$Z->B$, 则用$Z->B$代替$Y->B$

- - 计算闭包: 先计算单属性, 再计算双属性, 三属性, 四属性...
  - 最小化:
    - 如果投影后某函数依赖可以由其他依赖推导出,则删除
    - 如果某个依赖删除右部某个属性后左部依然成立, 则删除右部那个属性
- 两个技巧:
  1. 空集和key的闭包不需要计算
  2. 如果某属性X的闭包包含全部属性, 则不需要再计算X与其他属性的组合的闭包



3种异常, 判断无损分解, 分解的评判标准与chase检测

- 异常:
  - 冗余redundance
  - 更新异常: 对象的一个值被更改, 但并非所有该对象的值都被更改
  - 删除异常: 删除元组时所有该对象的记录都丢失,导致对象明明存在但是在数据库中无影无踪
- 无损分解的判断: 如果$R_1 \bigcap R_2$是R_1或R_2的$superkey$, 则R上的分解(R_1, R_2)是无损分解
- 分解的评判标准
  1. 消除异常
  2. 无损连接: 分解后再连接是否和原来的关系一致
     - chase检测
  3. 依赖保持: FD的投影在分解的关系中成立,但是自然连接后不能满足原来的FD

BCNF的定义,BC的分解

- BCNF: 凡是关系R中的非平凡函数依赖,其右部都是R的surperkey
- BC的分解
  1. 找到当前关系中的键
  2. 在关系所给FD中随便选一个违反BC(右部不是键的非平凡依赖)的FD, 如果找不到, 该关系不需要分解,返回原FD,算法结束;否则到3
  3. 对于违反BC的每一个FD, 根据上层关系的FD求右部属性R_i的闭包${R_i}^+$, 将关系的属性分解为$S_1 = {R_i}^+; S_2 = R - {R_i}^+ + R_i$, 将上层FD分别投影到$S_i$中, 对于每一个$S_i$以及投影到他上面的函数依赖, 执

行1

minimal basis的求法,prime,3NF的定义,3NF的合成(3NF synthesis)

- minimal basis最小函数依赖集求法
  1. 右部属性单一化: 分拆右部
  2. 左部属性最小化: 逐一去掉FD,看剩下的FD是否能够得到刚去掉的FD, 能则刚去掉的FD是多余的,复杂度$O(n^2)$
  3. 无多余函数依赖: 若$X->A$,且$X=B_1B_2...B_n$,逐一检查B_i, 若$A\in(X-B_i)_F{}^+$, 则以$X-B_i$取代X
- prime: key的成员
- 3NF: 左部是superkey或者右部是主属性
- 3NF的合成
  1. 求关系函数依赖FD的最小基本集G
  2. 求关系的键
  3. 对于G中每一个FD: $X->A$, 将左右部放在一起形成一个关系模式XA
  4. 查看所有生成的关系模式中是否包含键, 如果包含,则不需要操作, 直接返回XA集作为结果;否则,添加一个键作为一个关系模式,与XA们一起作为结果返回
- BC和3NF的比较
  - BC: 无损连接lossless join
  - 3NF: 无损连接lossless join + 依赖保持dependency preservation

完全函数依赖(full function dependence)

FD: $X->Y$的左部最小

2NF,1NF的定义与判断

- 2NF: 每一个非主属性只由主属性决定
- 1NF: 每个属性都保持原子性(每一列都是不可分割的数据项)

3_2 fd的公理系统: 基本符号:关系模式,函数依赖的闭包,属性集的闭包,最小函数依赖集/Armstrong的3条规则和3个推理/闭包的计算,求闭包算法的循环次数/FD等价与FD覆盖

基本符号: 关系模式,函数依赖的闭包,属性集的闭包

- 关系模式: $R(U,F)\begin{cases}U: & \text{关系}R\text{上的属性总体}\\F: & \text{关系}R\text{上的函数依赖}\end{cases}$
- 函数依赖的闭包: ${F}^{+}$: $R(U,F)$中所蕴含的函数依赖全体
- 属性集的闭包: ${X_F}^{+}$: 所有能由$R(U,F)$中的函数依赖导出的右部的集合
- 最小函数依赖集: F_m : 通过右部单一化, 左部最小化, 消除多余函数依赖得到的函数依赖集
  - 最小函数依赖集不一定唯一

Armstrong的3条规则和3个推理

- Armstrong的3条规则: 自反, 增广, 传递
- Armstrong的3个推理: 合并,伪传递, 分解

闭包的计算,求闭包算法的循环次数

- 闭包的计算: 求属性集$X(X\in U)$关于U上的函数依赖集F的闭包${X_F}^{+}$

> 输入: X, F
> 输出:${X_F}^{+}$
>
> 1. 令$X^{(i)}=X$, 此时$i=0$
> 2. 求B : B是以X^i中任意一属性为左部的F中某函数依赖的右部元素的集合
> 3. 令$X^{(i+1)}=X^{(i)}\bigcup B$
> 4. 判断$X^{(i)}==X^{(i+1)}$或$X^{(i+1)}==U$? $\begin{cases}yes & -----\text{算法结束}, {X_F}^{+}=X^(i)\\no & -----i=i+1, \text{回到}2\end{cases}$

- 闭包算法最大循环次数: $|U|-|X|$

FD等价与覆盖

- FD等价: ${F}^{+}={G}^{+}$(F覆盖G,反之也可以)
  - 充要条件: $F\in {G}^{+}$且$G\in {F}^{+}$
  - 判定: 逐一对F中的$FD:X->Y$, 考察Y是否在${X_{G^+}}^{+}$中

4 MVD多值依赖: MVD的定义,平凡多值依赖与非平凡多值依赖,MVD的6个性质,MVD与FD的关系/4NF的定义,4NF与BCNF的对比,4NF的分解/MVD或FD是否存在的判断(使用表格,或利用传递性)

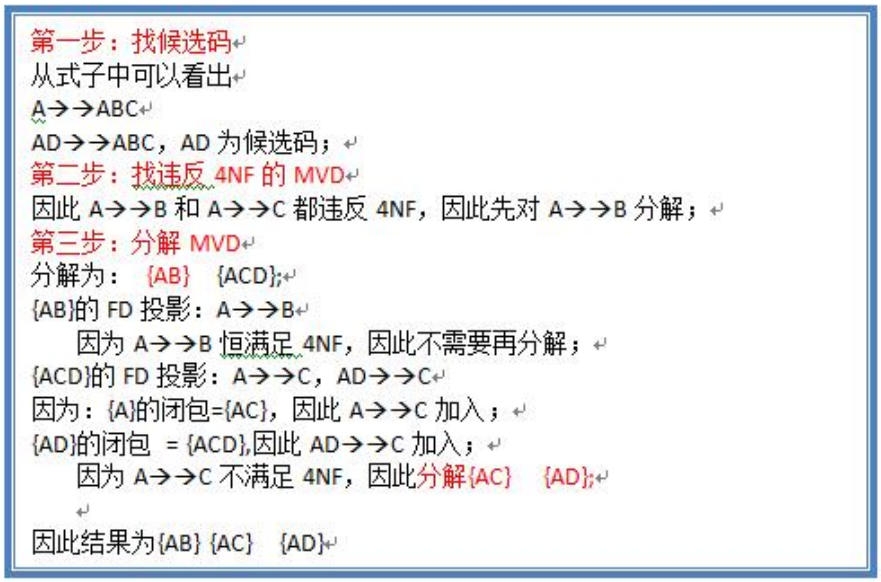
MVD的定义,平凡多值依赖与非平凡多值依赖,MVD的6个性质,MVD与FD的关系

- MVD: 在关系R中X中的属性完全相同时,交换Y中的属性值,其结果依然属于关系R
- 非平凡多值依赖$nontrivalMVD$: $X->->Y$, 其中, Y不是X的子集而且$X+Y \neq U$
- 平凡多值依赖trival MVD: 不是非平凡MVD
- MVD的6个性质:
  1. $X->->Y => X->->(U-Y-X)$
  2. $X->->Y, Y->->Z => X->->Z$
  3. $X->Y => X->->Y$
  4. $X->->Y, X->->Z => X->->(Y \cup Z)$
  5. $X->->Y, X->->Z => X->->(Y \cap Z)$
  6. $X->->Y, X->->Z => X->->(Y-Z)$或$(Z-Y)$
- MVD与FD的关系
  - FD左部不可拆,右部可拆
  - MVD左部不可拆,右部不可拆
  - FD是特殊的MVD

4NF的定义,4NF与BCNF的对比,4NF的分解

- 4NF: 对于R中每一个非平凡MVD,其左部都是$superkey$
- 4NF与BC: 满足4NF一定满足BC,反之不成立
- 4NF的分解
  1. 在MVD中找一个4NF违例$X->->Y$,找不到则分解结束,返回当前属性集合
  2. 分解为两个子集$S_1 = XY, S_2 = X + (U-Y)$
  3. 分别投影到S_1, S_2中, 执行1

给定 R(A,B,C,D),存在 MVD：
(1)A→→B
(2)A→→C
分解为满足 4NF 的关系



MVD或FD是否存在的判断(使用表格,或利用传递性),规律小结

- MVD或FD是否存在的判断:

> - 问题描述: 给定一个关系R,给出MVDs和/或FDs,求某个MVD或FD是否在关系中
> - 解决方法: 将关系用表格的形式表示出来, 查看表格中的元组是否满足函数依赖或多值依赖

1. 生成表元素:
   - 生成FD数据元素: 两行元组左部属性相同,右部也相同
   - 生成MVD数据元素: 两行元组左部属性相同,右部相同,其他属性相同,还有一行左部与其相同,右部交换,其他属性相同
2. 已知MDV.FD:
   - 证FD(需要证明相同左部,右部相同): 先使用MVD,在已知FD左部相同时交换求证FD右部属性;再使用已知FD,证右部相同

The Tableau for A->C

Goal: prove that $c_1 = c_2$.

A	B	C	D
a	b_1	~~c_1~~ c_2	d_1
a	b_2	c_2	d_2
a	b_2	c_2	d_1

Use A->->BC (first row's D with second row's BC).

Use D->C (first and third row agree on D, therefore agree on C).

- 证MVD(需要派生出相同左部,右部交换的元组): 连用两次不同MVD,交换使两行元组只有目标MVD右部不同

The Tableau for A->->C

Goal: derive tuple (a, b_1, c_2, d_1).

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2
a	b_1	c_2	d_2
a	b_1	c_2	d_1

Use A->->B to swap B from the first row into the second.

Use B->->C to swap C from the third row into the first.

5 Relational Algebra (ra 关系代数) : relational algebra的定义,操作数operand与算子operation/核心关系代数:selection,projection,product and join,rename,执行结果/表达式的构建,表达式的三种写法,算符优先级/包bag定义,与集合的区别,包与集合上操作的不同之处

relational algebra关系代数的定义,操作数operand与算子operation

- ra: 操作数+算子(由操作数和算子组成的数学系统)
- operand: 操作对象,(属性啥的)
- operation: 算子

核心关系代数:selection,projection, product and join, rename, 执行结果

- $selection$: 在属性集R_2中选择满足条件C的属性组成集合R_1
  - $R_1 := \sigma_C(R_2)$
  - 结果模式与操作数模式相同(属性个数不变)
- $projection$: 在属性集R_2中按顺序选择L中的属性组成新的集合R_1
  - $R_1 := \pi_L(R_2)$
  - 结果由新的属性列构成
    - 扩展投影: L可能是一个关于属性的表达式, 可能导致结果中属性重复出现

R =

A	B
1	2
3	4

$\pi_{A+B->C,A,A}$ (R) =

C	A1	A2
3	1	1
7	3	3

- $product$笛卡尔积: 按$R_1(m行), R_2(n行)$的顺序将两个关系的每一行连接起来$(m \times n行)$, 同名列前用点标注来源
  - $R_3 := R_1 \times R_2$
  - 结果属性集是两个关系的属性合集
- $join$:
  1. $Theta - join$: 先求笛卡尔积, 再找满足条件的元组
     - 有下标C,C是条件
     - 结果与笛卡尔积一样
  2. $Nature - join$: 根据同名同值属性合并两表
     - 没有下标C
     - 结果是两个关系的级联
- $rename$: 重命名$R_1 := \rho_{R_1(A_1,A_2,...A_n)}(R_2)$

表达式的构建,表达式的三种写法,算符优先级

- 表达式的构建:结合算符优先级
  - 三种写法:
    1. 赋值语句序列: 新的对象 := 对原有对象的操作
    2. 具有多个算符的表达式:

       Example: the theta-join R3 := R1 $\bowtie_C$ R2

       can be written: R3 := σ_C(R1 X R2)
    3. 表达树:叶:操作数; 内部节点: 应用于一个或多个子节点的运算符
- 算符优先级: 选择投影重命名 > 笛卡尔$join$ > 交 > 并, 减

  选择投影重命名,笛卡尔join跟后面,接着是求集合交,最低求并与求减

包bag定义,与集合set的区别,包与集合操作的不同之处

- bag: 元素可以重复出现
- 与set的区别:
  1. 元素是否重复出现
  2. bag上操作不去重
  3. sql基于bag
  4. 一些操作(譬如投影)在bag上更高效
- 与set的同: 求并集的交换律成立

6 数据库-连接库: SQL注入/基于库libraries的SQL接口/三层架构/php游标cursors

基于libraries的SQL接口

- $C + CLI$
- $JAVA + JDBC$
- $PHP + PEAR或DB$

数据库环境的三层结构:

1. Web server: 直接和用户交互
2. application server: 执行逻辑指令
3. database server: 按要求完成交互

基于libraries的SQL接口

- CLI: C的SQL连接库
- JDBC: Java database connectivity,类似于C的CLI,但是主语言是java

php游标cursors

相当于指针

7 SQL intro: select-from-where/使用as重命名/模糊查询的两种模式/空值null的两种含义/三值逻辑/

多重关系查询:定义,方法/self-join查询同一关系中的两个不同对象/子查询描述,返回值/in(与普通query的区别),exists,any,all,union,intersect,except用法,语句的值,用在哪里/基于包执行的操作,基于集合执行的操作/消除重复项duplicate elimination--distinct/join操作: 笛卡尔积,自然连接,theta连接

模糊查询的两种模式

- 使用$like$或者$notlike$
- $\%$: 字符串
- _ : 字符

空值null的两种含义

1. missing value: 存在但是不知道
2. inapplicable: 不存在

三值逻辑

- True:1
- False: 0
- unkown: $\frac{1}{2}$
- 操作:
  - and : 取最小
  - or : 取最大
  - $not(x) : 1-x$

多重关系查询Multirelation query定义,方法

- 定义: 从至少一个关系中查询
- 方法: $from$子句后跟多个关系名称, $where$子句后的查询属性名前标识关系名

self-join查询同一关系中不同的两个对象

在from子句后,差多少对象就写多少遍关系名, 每一个都需要重命名

子查询subquery描述,返回值

- 描述: 将括号括起来的完整select-from-where子句作为条件插入from或者where之后
- 返回值: 子查询必须返回一个有确定值的单属性元组
  - 如果返回元组值为空或者有多个值,则会run-time error

in,exists,any,all,union,intersect,except用法,语句的值,用在哪里

- in:
  - $< 属性 > in < 子查询 > \begin{cases} 属性在子查询返回值中： & true \\ 属性不在子查询返回值中： & false \end{cases}$
  - 用在: $where$子句之后
  - 相应的: $not \quad in$
  - 与普通query的区别: in 可能只输出一次, 普通Query可能有重复输出
- exists:
  - $exist < 子查询 > \begin{cases} 子查询结果非空： & true \\ 否则： & false \end{cases}$
  - 子查询用括号括起来
- any:
  - $x < 操作符 > any < 子查询 > \begin{cases} 子查询结果中有一个元组值与x满足操作符关系： & true \\ 否则：false \end{cases}$
  - 子查询结果中必须包含至少一个元组,操作符可以是任何比较操作符
- all:
  - $x < 操作符 > all < 子查询 > \begin{cases} 子查询结果中所有元组值与x满足操作符关系： & true \\ 否则：false \end{cases}$
- union,intersect,except:
  - 交,并,差
  - 是子查询与之查询之间的操作

基于包执行的操作,基于集合执行的操作

- 基于包执行的操作: select-from-where
- 基于set执行的操作: 交并差

消除重复项duplicate elimination--distinct

$select \quad distinct$属性名...可以消除重复元组

join操作: 笛卡尔积,自然连接,theta连接

- product笛卡尔积:$R \quad cross \quad join \quad S$
- nature join自然连接:$R \quad nature \quad join \quad S$
- theta join: $R \quad join \quad S \quad on < 条件 >$

8 ra-sql: 拓展关系代数:包去重,排序/悬挂元组dangling,内连接,外连接:左外连接,右外连接,全连接/聚集aggregation:sum,max,min,avg,count,以及与distinct的结合,聚集中的空值/分组,使用分组对select进行限制,分组中的聚集函数,使用分组的两个必须条件/having的使用以及规则,后面跟随的条件/数据库插入(普通插入,批量插入,利用子查询插入),删除(全部删除,部分删除),更新(全部更新,条件更新),缺省

拓展RA:包去重,排序,分组聚集

- 包去重: $R_1 := \delta(R_2)$
- 排序: $R_1 := \tau_L(R_2)$ ----------唯一一个结果既不是set也不是bag的关系代数
  - L是属性的排列,按L中属性的顺序依次比较R_2元组的值来排序

悬挂元组dangling tuple,内连接,外连接中悬挂元组的表示,外连接:left outer join, right outer join, full outer join

- 内连接: 根据两表中相同的列中的同值元组合并两表, 包括自然连接
- 悬挂元组: theta连接中属于左边的关系,但是右边关系中没有元组和它连接
- 外连接中悬挂元组的表示: 用Null
- 外连接: 属于自然连接, 根据同列名同值元组合并两表
  - left outer join:根据左表匹配右表,右表中没有匹配项,则结果表的右表相应位置null
  - right outer join:根据右表匹配左表,左表中没有匹配项则结果表的左表相应位置null
  - full: 外连接的默认选项, 结合左右两种情况

aggregation:sum,max,min,avg,count,在聚集函数中使用distinct,聚集函数中元组null的情况

- 例:

```
select student, sum(score) as score_sum
from Student
where score>=60
```

- 在聚集函数中使用distinct:

```
select count(distinct price)
from sells
where beer = 'Bud';
```

- 聚集函数中元组null的情况: 忽略,不计数,若只有null,则认为是0

分组,使用分组对select进行限制,分组中的聚集函数,使用分组的两个必须条件

- 分组group by: group by作用于select from where语句的执行结果,并在select语句之后的聚集函数中体现

```
select beer, avg(price)
from sell
group by beer
```

结果: 返回已经售出的每款酒(按名字分组)的平均售价

- 使用分组的两个必须条件(满足其中之一):当sql语句中使用分组时,其select之后的属性列表必须:
  1. 有聚集函数修饰 或
  2. 属于gruop by修饰的属性

having的使用以及规则,后面跟随的条件

- having与group by结合使用,后面跟随 **对每一组分组的约束条件**
- 例:在已售表中找特定厂家生产的酒或者至少三家酒吧都售卖的酒

```
select beer, avg(price)
from Sell
group by(beer)
having count(bar) >= 3 or
    beer in (select name
             from Beer
             where manf = "Bud")
```

- having后面跟随的条件: 子查询,否则必须是聚集函数或必须属于group by所依据的属性

数据库插入(普通插入,批量插入,利用子查询插入),删除(全部删除,部分删除),更新(全部更新,条件更新),缺省

- 插入: insert into <表名> values (值),多行值用逗号分开括号
  - 利用子查询插入: 注意子查询的返回值与插入对象匹配,在insert into XXX后直接跟子查询语句

```
insert into Drinkmates
(select d2.name
 from Drinker d1, Drinker d2
 where d1.name="Saly" and d2.name <> "Sally" and d1.bar = d2.bar)
```

- 删除: delete from <表名> where <条件>
  - 全部删除: 不写where,表置空,where选择有特定值的列:特定列删除
  - 部分删除: 譬如如果有一款酒的厂家制造了其他酒,就把这家厂子的所有酒删掉:where后面跟exists,其对象是一个子查询,查找同厂酒并返回酒的名字,**注意:在执行完where之后exists后面的子句时,已经有一个静态结果表了,在删除时是静态的从结果表中找名字,直接在目标表删除那些名字的元组**

```
delete Beer b
where exists(
    select name
    from Beer
    where manf = b.manf and name <> b.name
);
```

- 更新: update <表名> set <列的值> where <条件,用来判断哪个元组需要更新>

9 约束constraint: 约束的定义,种类/外键的定义,外键表达方式,维护外键约束的3种方式(default,cascade,Null(缺省,级联,置空))以及选择策略/三种检测(attribute-based, Tuple-based, assertion断言)的方式,时机/触发器trigger描述,使用时机,创造规则:event-condition-action

约束的定义,种类

- constraint: DBMS强制执行的数据之间的关系

- 种类: $\begin{cases} key \\ foreign \quad key \\ value-based \quad constraint \\ tuple-based \quad constraint \end{cases}$

foreign key外键的定义,外键表达方式,维护外键约束的3种方式(default,cascade,null(缺省,级联,置空))以及选择策略

- foreign key表达方式: 表R的a属性后面加primary key,表S的b属性后面加references R(a); 或表S最后加一行foreignk key(b) references R(a).注意a必须是pk或者unique
- 维护外键约束的3种方式
  - default: 拒绝改动
  - cascade: 一方修改,另一方也跟着修改
  - null: 删除操作:将元组的值置null,再修改另一方相应元组为null
- 选择策略:
  - 定义外键时, 对于删除和插入用置空或级联
  - 定义外键之后,对于更新和删除用null或级联cascade

```
foreign key beer references Beer(name)
on DELETE set null
on update cascade
```

三种检测(attribute-based,tuple-based,assertion)的方式,时机

- attribute-based check: 定义时写在相应属性后面, check <条件(可以是子查询,使用其他表必须用子查询)>
  - 时机: 更新或插入

```
beer char(20) check (
    beer in (select name from Beer)
)
price real check (price <= 5.00)
```

- tuple-based check: 在定义表的属性时单独做一行, check <条件(可以使用子查询,使用其他表必须用子查询)>
  - 时机: 更新或插入

```
create table Sell(
    bar char(20),
    beer char(20),
    price real,
    check  (bar = 'Bud' or price <= 5.00>)
);
```

- Assertion: create assertion 名字 condition (<条件>);
  - 时机: 任何关系的每一次修改

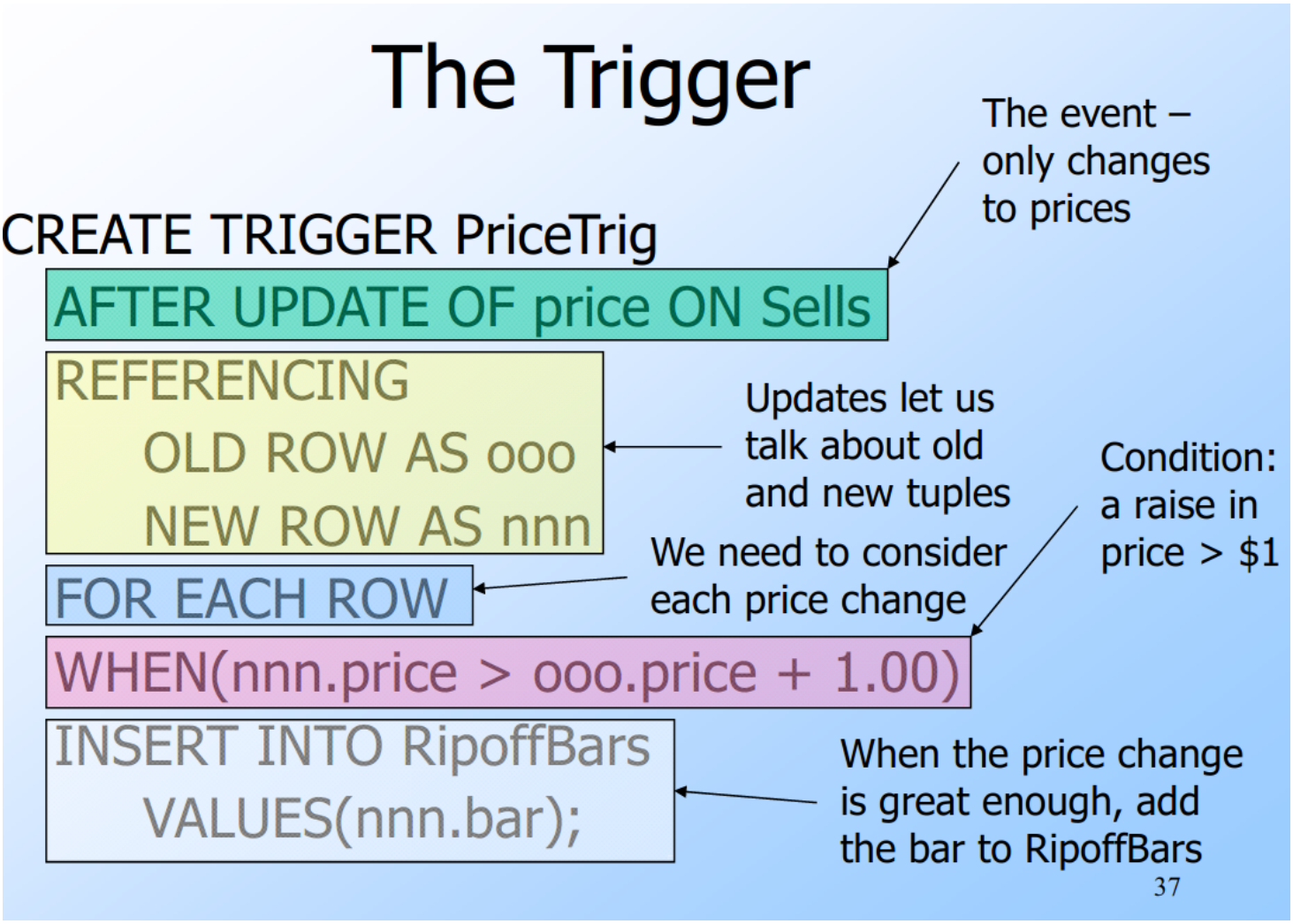
触发器trigger描述,创建,使用时机,创造规则,event-condition-action规则

- trigger描述: attribute-或者tuple-based check的使用时机是固定的, 触发器可以让用户自己决定何时check. 语法: `create trigger 触发器名字`
- 时机: 在用户需要的时候
- event-condition-action: 当event出现时,对具有某条件condition的表或属性,执行某action

> 例子: 有两张表,Beer记录有的酒,Sell记录卖出去的酒,Sell中的Beer存在与Beer之间的外键约束,现在卖出了一款新酒,要向Sell中直接插入记录
> 代码:

```
create trigger Beertrig
    after insert on Seel -- event
    referencing new row as newtuple
    for each row
    when (newtuple.beer not in (select name from Beer)) -- condition
    insert into Beer(name) values (newtuple.beer); -- action
```

另一个例子



10 transaction-view-index事物-视图-索引:transaction定义,ACID,导致事物结束的两种情况:commit,rollback/回滚rollback:导致回滚的情况,解决方法/sql4个隔离层级isolated level,选择/事物序列化的定义,操作/view的定义,种类,声明,结合触发器的视图,基表更改引起实例化视图更改问题以及解决方法/index索引定义,声明,使用索引优化数据库查询tuning的优点,缺点

transaction定义,ACID,commit

- transaction事物: 在数据库上进行的查询或修改操作
- ACID: 事物的4个基本特点:原子,一致,隔离,持久
  1. Atomic: Whole transaction or none is done.要么发生要么不发生
  2. Consistant: constraints preserved. 约束保持良好,前后不变化
  3. Isolated:it appears to the user as if only one process execute at a time.
  4. Durable: 事物一旦提交,其对数据的改变不可更改
- 导致事物结束的两种情况:
  - commit: sql通过commit提交事务对数据的操作,导致事务完成
  - roll back

roll back:导致回滚的情况,解决方法

- 导致回滚的情况:被0除,违反约束,程序员弄的
- 解决回滚带来的数据混乱: 用commit代替rollback,在commit之前,其他人无法看到其效果

sql4个隔离层级isolated level,选择

- 四个级别:
  - read uncommited: 未提交读
  - read commited: 提交读
  - repeatable read: 重复读
  - serializable: 序列化
- 选择: 自己选

视图的定义,两种类别,声明,结合触发器的视图,视图实例化,基表更改引起实例化视图更改问题的解决方法

- view的2种类别
  - virtual,不可修改,因为它不存在
  - Materialized
- 声明:

```
create [种类] view <名字> as <操作(select from where之类)> -- 种类默认是虚的
```

- 结合触发器的视图
  可以通过instead of触发器来对虚拟视图进行修改,instead of insert on 视图,XXXXX(操作)
- 基表更改引起实例化视图更改问题以及解决方法
  解决: 定时更新视图

index索引定义,声明,使用索引优化数据库查询tuning的优点,缺点

- index: 加速查询,哈希表或B-树
- 声明: create 索引名 on 表名(列名);多个列名括号内用逗号隔开
- 使用索引优化数据库查询tuning的优点,缺点
  - 优点: 加快查询sppeed up query
  - 缺点: slow down all modifications on relation(因为索引也需要随之更新)

11 psm持久型存储模块(存储过程),pl与sql: psm定义,参量的三种类型,声明,invoke调用,语法:判断,循环,指针,return/动态SQL声明,调用

psm定义,参量,声明,invoke,3个基本种类及作用,语法:判断,循环,指针

- PSM持久型存储模块(存储过程)参量3种类型:
  1. in : 使用, 但不更改
  2. out : 相当于C引用
  3. $inout$: 都有
- 声明:

```
create procedure 名字 (<参量类型> <参量名>) --多个参量用逗号隔开
    <变量声明>
    <操作>; -- 常规sql语句select-from-where之类
```

- invoke: `call 存储过程名(实参)`
- 语法:判断,循环,指针,return
  - 判断: $if..then..elseif..then..else..end\quad if$;,总之end if之前一定是else
  - 循环:

```
循环名字:Loop
    循环内操作;
    leave 循环名; -- 跳出条件
end Loop;

while<条件>
do <操作>
end while;

repeat <操作>
until<条件>
end repeat;
```

  - 指针: create 指针名 cursor for <查询>
    - 使用: open 指针名
    - 关闭: close指针名
  - return:并不代表结束,只是return

动态SQL的声明,调用

- 声明:

```
exec sql prepare 名字 from <sql语句>;
```

- 调用:

```
exec sql execute 名字;
```

12 grant授权: 语法:授权(操作权限,授权权限),撤销授权revoke,撤销授权的两种选项/授权图的点,边,AP,P*,P**,授权规则,撤销授权规则

语法:授权(操作权限,授权权限),撤销授权(操作权撤销,授权权撤销),撤销授权的两种选项

- 授权
  - 操作权: grant <操作列表> on <需要操作的对象> to <被授权者>

    - 授权权: 既可以操作,又可以授权,在操作权授权后加 with grant option

```
--举个例子
grant select,update
on Sell
to Sally;
```

- 撤销授权
    - 语法: revoke <操作列表> on <需要操作的对象> from <被撤销授权的人>
- 撤销授权的两种形式
    - cascade级联: 权限被撤销,则由该用户授权的所有其他用户的权限也撤销,不管传得有多远
    - restrict限制: 如果被撤销授权的用户还对其他用户授权了,则不能撤销对他的授权并弹出警告

授权图的点,边,AP,P*,P**,授权规则,撤销授权规则

- 点: 圆,(被)授权对象\权限
    - 只要不同权限,哪怕在同一数据上操作,都是两个不同节点
- 边: 授权者指向被授权者
- AP: 用户A有P权
- P*: P的授权权
- P** :P权本权(包含授权权)
- 授权规则: A把P权(或P*权,代表有授权权)授予B,就用一个实线箭头指向B
- 撤销授权: 注意是否是级联,是则无法撤销.每个节点都必须有到相应P**的路径,否则把它删掉

13 concurrency 并发控制: 基本概念:transaction事物,conflicting action冲突行为的种类,schedule调度,调度的4种方式/并发事务运行存在的3个异常/冲突等价conflict equivalent的定义/precedence graph符号表示,前驱图的节点,边,两个定理/(一级)加锁解锁协议:符号,legal schedule合法调度和well-formed调度/2PL两阶段锁协议描述,避免数据出错的其他4种方法(共享锁,多粒度,插入删除,其他机制),共享锁和排他锁描述,3个法则(well-formed,legal,upgrade变化问题)/increment lock增量锁和update lock的描述,符号/锁的兼容性/系统

一个解决并发多用户加解锁问题的方法/多粒度封锁 granularity:多粒度封锁描述,锁的类型((主要是意向锁:)IS,IX,SIX),6个规则,兼容性,加锁对象上允许再加的其他锁/pehantom数据重影定义,解决方法

基本概念:transaction,conflicting action的种类,schedule,调度的四种方式

- transaction:事物,(狭义)读写行为
- conflicting action:
  1. 同一事物的读r和写w
  2. 不同事物对同一元素的w
  3. 不同事物对同一元素一个r一个w
- schedule:调度
  - 四种方式:

$$\begin{cases} serial \quad schedule: \text{串行化调度, 事物之间没有交错执行的部分, 按顺序一个一个来} \\ serializable \quad schedule: \text{可串行化调度, 调度结果与某个串行调度执行结果等价} \\ ACR: avoid \quad cascating \quad rollback: \text{避免级联回滚：事物仅读已经提交事务修改的数据} \\ strict: \text{事物写入的值在其} commit \text{之前没有其他事物读或写} \end{cases}$$

并发事务运行存在的3个异常

1. 丢失修改
2. 不可重复读
3. 读脏数据

冲突等价conflict equivalent定义

- conflict equivalent: S_1可以通过非冲突事物交换转换为S_2

precedence graph前驱图的符号表示,节点,边,两个定理

- 符号:$P(S)$ $-------S$是调度序列
- 节点: S中的事物
- 边: 先执行指向后执行
- 两个定理:
  1. 冲突等价的事物前驱图相同,反之不成立
  2. 有环<=>冲突可序列化

加解锁协议:符号,legal schedule和well-formed

- 符号:加锁：$L_i(A)$;　解锁：$U_i(A)$
- well-formed: 同一事物对同一数据的所有操作在锁内执行
- legal schedual: 数据被某事物锁了的时间内不允许其他事物来上锁

2PL:描述,避免数据出错的其他4种方法(共享锁,多粒度,插入删除,其他机制),共享锁和排他锁描述,3个法则(well-formed,legal,upgrade变化问题)

- 描述:事物运行完所有工作之前不允许解锁
- 避免数据出错的其他4种方法: 共享锁shared lock,multiple granularity,insert delete phantom,other types of C
- 共享锁和排他锁:
  - 共享锁: $S(A)$, 加上之后其他事物不可加排他锁
  - 排他锁: $X(A)$, 仅允许本事物操作数据, 其他食物不能加锁不能操作
- 3个法则
  1. well-formed
  2. legal: 共享锁上锁期间不允许其他事物加排他锁,排他锁上锁期间不允许其他事物上锁
  3. 对于更新操作:如果操作后锁的数量比原来多,则不允许更新;否则可以更新

increment lock和update lock描述,符号

- increment lock:$IN_i(A)$
- update lock:意向锁,以I开头,意向读,意向写

锁的兼容性

X均不兼容,共享锁与意向锁兼容本身

	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

系统一个解决并发多用户加解锁问题的方法

don't trust transactions' requests of locks, system hold all locks until transaction commits

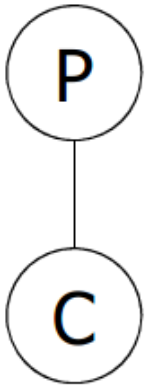
多粒度封锁granularity:多粒度封锁描述,锁的类型((主要是意向锁:)IS,IX,SIX),6个规则,兼容性,加锁对象上允许再加的其他锁

- granularity: 封锁对象的大小称封锁粒度,譬如一张表,一个属性啥的
- 锁的类型: 锁的强度 : $X > SIX > (IX = S) > IS$
  - IS: 意向共享
  - IX: 意向排他
  - SIX: 共享意向排他
- 6个规则:
  1. 遵循粒度强度
  2. 先锁根节点
  3. 上共享或意向共享(S\IS)锁之前必须有某个父节点被上了意向锁(共享排他都可以)
  4. 上X,IX,SIX之前必须有某个父节点被上了IX\SIX
  5. 事物满足2PL
  6. 某事物给节点解锁的条件是其子节点被该事物锁了
- 兼容性

Comp		Requestor				
		IS	IX	S	SIX	X
Holder	IS	T	T	T	T	F
	IX	T	T	F	F	F
	S	T	F	T	F	F
	SIX	T	F	F	F	F
	X	F	F	F	F	F

- 加锁对象上允许再加的其他锁

Parent locked in	Child can be locked by same transaction in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none



pehantom数据重影定义,解决方法

- pehantom: 多用户并发执行操作之后同一数据有了不同的值
- 解决: use multiple pehantom tree, before insert of node Q,lock Q's parent

14 tp: 事务结束的两种情况/可恢复调度,避免级联回滚的概念/死锁的检测,4个避免方法

事务结束的两种情况

- commit
- abort:会回滚

可恢复调度,避免级联回滚的概念

见13

死锁的检测,4个避免方法

- 死锁的检测: wait-for graph,指向请求
- 死锁避免的4个方法
  1. resource ordering
  2. timeout
  3. wait-die:请求资源的进程只能等待固定时间,不然就die

4. Wound-wait

15 view serializability视图可串行化: 视图等价,视图可串行化定义,定理,判断方法

view equivalent,view serialiability定义,定理,判断方法

- view equivalent
  - 定义：调度视图等价于一个串行调度
  - 定理：冲突可串行化 $=>$ 视图可串行化, 反之不对
  - 判断方法：使用代表及的优先图, 只要有一个优先图无环, 调度是视图可串行化
- view equivalent定义：对一个事物集的两个调度S_1, S_2：如果对于每个数据项：
  - 事物在S_1中读取其初值, 则在S_2中也读取其初值
  - 事物在S_1中对由T_x产生的值执行$read$, 则在S_2中也对T_x产生的值$read$
  - S_1中有事物执行了最后的$write(Q)$, S_2中该事物也必须执行最后的$write(Q)$

21 xml可扩展标记语言: xml概念,两种类型,结构/DTD概念,结构,元素,用法,属性/ID和IDREF/XML例子

xml概念,两种类型

- xml:可扩展标记语言,标记电子文件使其具有某种结构
- xml两种类型
  - well-formed:可以自定义标签(invent own tag)
  - valid:必须符合一个标准模式(conform a certain DTD)
- 结构: 开头结尾被"<...xml...></...xml...>"修饰,中间用其他标签修饰

DTD概念,结构,元素,用法,属性

- DTD: 文档类型定义,声明于xml中
- 结构:

```
<!DOCTYPE <root tag>[
    <!ELEMENT <NAME>(<components>)>
    ...其他elements....
]>
```

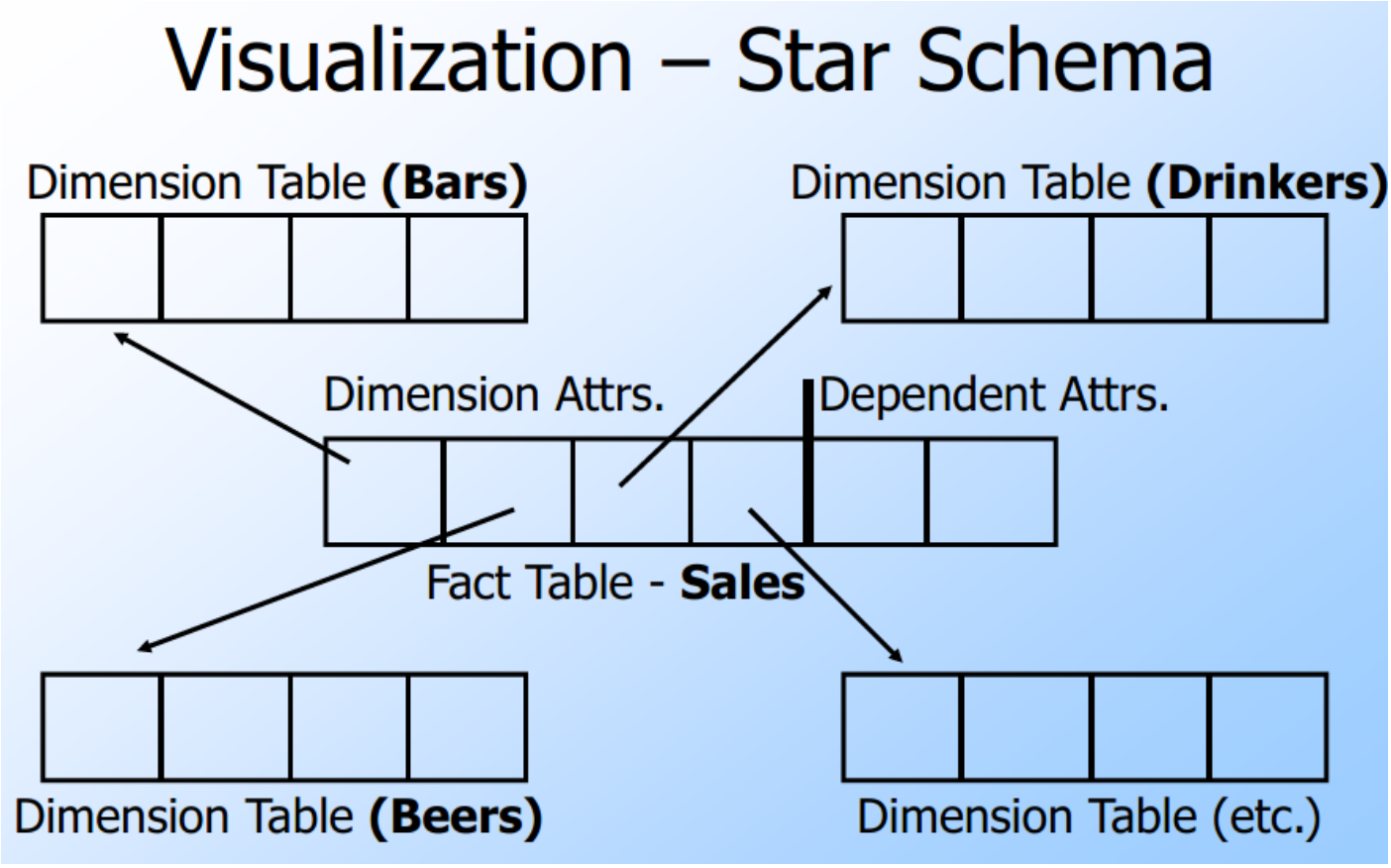
22 olap: data warehouse,olap,oltp,data mining/star schema的两个组成部分(事实表,维表)/cube的drill-down和roll-up操作

data warehouse,olap,oltp,data minging

- data warehouse: 数据仓库
- OLAP: 联机分析处理,探索挖掘数据价值作为决策参考
- OLTP: 联机事务处理,一线业务操作
- data minging: 数据挖掘

star schema的两个组成部分

- 描述: 事实表被维表包围,维表主键关联事实表外键,一个粒度一个维度
- fact table: 一张
- demension table: 不同维度之间互不关联



cube的drill-down和roll-up操作

- drill-down: 向下钻取,"拆分group by"
- roll-up: 向上钻取,合并

24 distributed分布式数据库: 优点,问题

分布式数据库优点

模块化,容错,高性能,数据共享,低内耗
modularity,fault tolerance,high performance,data sharing,low cost components

分布式数据库解决的问题

数据分配,并行,并发与恢复,异质性
data distribution,exploiting parallelism,concurrency and recovery,heterogeneity