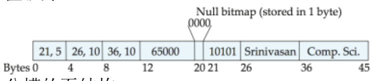
Relational Model
笛卡尔积要求属性均不同相同,否则需要重命名;∏balance(account) – ∏account.balance(σaccount.balance<d.balance (account x ρd(account)))来寻找 account(account-number, branch-name, balance)里最大的 account balance;$r \bowtie_\theta s = \sigma_\theta(r x s)$;÷常用于查询“所有”,$\prod_{Sno, Cno}(enrolled) \div \prod_{Cno}(course)$来查询选修了所有课程的学生;

SQL
表属性修改:alter table r add A/alter table r drop A;union 交,intersect 并,except 差;select 子句中不在聚集函数中的属性要用 group by;From→where→group (aggregate)→having→select→distinct→order by;和 null 的算术运算、比较运算结果都为 null;要用 where A is null 而不是 A=null;所有的聚集函数除了 count(*)以外均忽略空值,注意 count(A)也忽略;查最大值:SELECT account_number, balance FROM account A WHERE balance >= (SELECT max(balance) FROM account B WHERE A.branch_name = B.branch_name);找非最小值:SELECT distinct T.branch_name FROM branch as T, branch as S WHERE T.assets > S.assets and S.branch_city = ‘Brooklyn’;=some 就是 in,≠不是 not in,=all 不是 in,≠all 是 not in;exists r 等于 r 非空,not exists 等于 r 为空集;对于“所有”查询,常用 not exists…except…结构,原理是 X 包含于 Y 等价于 X-Y=∅,也就是 not exits X except Y;
Ex1 找出选修了 Biology 系的所有课的学生
Select S.ID, S.name From student as S Where not exists((select course_id from course where dept_name=’Biology’)except(select T.course_id from tabkes as T where S.ID=T.ID))
unique 和 not unique 来检查唯一性;CREATE VIEW <v_name> AS SELECT c1, c2, … From …;WITH max_balance(value) as SELECT max(balance) FROM account 来建立局部视图,在本次查询中使用;用 with 查询大于平均工资总额的系:WITH dept_tatal (dept_name, value) as (select dept_name, sum(salary) from instructor group by dept_name),   WITH dept_total_avg(value) as (select avg(value) from dept_total), select dept_name from dept_total, dept_total_avg where dept_total.value>=dept_totla_avg.value;from 中通过嵌套得到的表必须重命名;update…set…where…;inner join 不消除重名属性;CREATE ASSERTION <assertion-name> CHECK <predicate>;Create trigger overdraft-trigger after update of balance on account;GRANT <privilege list> ON <table | view> TO <user list>;REVOKE <privilege list> ON <table | view> FROM <user list> [restrict | cascade] ;

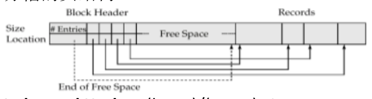
E-R model
右箭头的是一,无箭头的是多;单线是部分参与,在两边都是部分参与时,多的一方作主码;一边全参与,一边是一时,可以把联系集合并到多中;一对一也可以合并到任意一个一中;所有的弱实体集关系都是多余的;

Normal Form
无损连接分解: $r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$,判定方法:如果 R1∩R2 是 R1 或者 R2 至少一个之一的 SK,那么久无损;函数依赖,α 确定后 β 也就随之确定;检验 α → β is in F+等价 β ⊆ α +,检验 a 是 SK,等价于 R⊆ a+;无关属性检验:对于 F 中的 FD 中的属性 A,如果 A 是左边的(A α ’→ β),检验 α ’→ β 能否由删掉该 FD 中 A 后的 F 推出,如果 A 是右边的(α →A β ’即 α →A 和 α → β ’),检验 α →A 能否由删除掉该 FD 中的 F 推出;计算 FC 时,先用结合律;依赖保持:分解前的每个 FD 在分解后的某个 R 中仍然存在;BCNF:对 F+中的每个 FD(对于未进行过分解的原始 R,可以改为对 F 中),满足下面条件至少一个:①FD 是平凡的②FD 非平凡,但是 α 是 SK;BCNF 分解;$(result–R_i)\cup(\alpha,\beta)\cup(R_i–\beta)$,结果必是无损分解的;3NF:比 BCNF 多一个条件:β–α中的每个属性都在某个 CK 中,三个满足一个就行;3NF 分解:把 FC 中的每个 FD 都分解为 Ri=(α,β),最后保证某个 Ri 中存在 R 的 CK,同时保证依赖保持和无损分解;

Storage and File Structure
Access time=Seek time 寻道时间: time to reposition the arm to the correct track+Rotational latency 旋转延迟: time it takes for the sector to be accessed under the head;定长记录:使用空闲链表完成插删(在第一条记录前加一个 header);变长记录:按照属性出现顺序依次写,对于边长属性用 4 字节的(offset,length)来标记,属性记录结束后用 1 字节(取决于有几个属性,这里假设有 4 个,每个对应一位)表示属性是否为空值,为空值十将对应位置为 1,然后写前面的各个边长属性的值就好.



分槽的页结构:



Index and Hash n=(bs-ps)/(vs+ps)+1
Index 包含查询码和指针两部分;辅助索引必须使用稀疏索引(sparse);根节点指针数在⌈n/2⌉到 n 之间;叶节点查询码在⌈(n-1)/2⌉到 n-1 之间;Special cases:
1.If the root is not a leaf, it has at least 2 children.2.If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values;插入时的叶节点分裂: 将 m 个键值对排序,分裂为 2 组,前半放原节点,后半放新节点;新节点的最小键值对插入父节点,并排序.若已满,再分裂,向上;非叶节点分裂:会把新节点的最小值插入父节点中,且不在新节点中出现;

Query Processing
Selection Operation
basic algorithms : File scan –do not use index
A1 (linear search).
BT=br, AT=1; If selection is on a key attribute, we can stop on finding record. BT=br/2, AT=1
A2 (binary search).
if selection is an equality comparison on the attribute on which file is ordered.
BT =⌈log2(br)⌉— cost of locating the first tuple by a binary search on the blocks. Time cost=⌈log2(br)⌉*(tT+ts)
非码属性要加上所有满足条件的块,Sc(A,r)是满足条件的记录数,fr 是每个块的记录数
BT=⌈log2(br)⌉+⌈Sc(A, r)/fr⌉-1
Selections Using Indices and equality
A3 主索引,码属性的等值比较
Cost=(Hi+1)*(tT+ts), Hi 为索引树高
A4 主索引,非码属性的等值比较
Records will be on consecutive blocks
Let b be number of blocks containing retrieved records. b=⌈Sc(A, r)/fr⌉
Cost=hi*(tT+ts)+b*tT+ts
A5 利用辅助索引的等值比较
(1)Retrieve a single record if the search-key is a candidate key:
Cost=(Hi+1)*(tT+ts)
(2)Retrieve multiple records if search-key is not a candidate key
Cost=(Hi+b)*(tT+ts)不好用,可能比 linear 差
Selections Involving Comparisons
A7 基于辅助索引的比较
For $\sigma_{A \geq V}(r)$ use index to find first index entry   A ≥ v and scan index sequentially from there to the end, to find pointers to records.
Cost= (index + scan sequentially on leaf nodes + number of records)同样可能不如 linear

External Sort-Merge
# of merge passes: $\left\lceil \log_{M-1}\left\lceil \frac{b_r}{M} \right\rceil \right\rceil$
# of BT for sorting:$b_r(2\left\lceil \log_{M-1}\left\lceil \frac{b_r}{M} \right\rceil \right\rceil + 1)$
# seeks $2\left\lceil \frac{b_r}{M} \right\rceil + \left\lceil \frac{b_r}{b_b} \right\rceil * \left(2\left\lceil \log_{M-1}\left\lceil \frac{b_r}{M} \right\rceil \right\rceil - 1\right)$

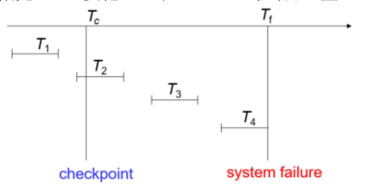
Join Operation
Nested-Loop Join
Worst: if there is enough memory only to hold one block of each relation.也就是 M=3,3 个分别分给内外关系及最终结果
$BT = n_r * b_s + b_r \quad AT = n_r + b_r$
Best: $M \geq b_s + b_r + 1$
$BT = b_s + b_r$ AT=2
Block Nested-Loop Join
worst case:$BT = b_r * b_s + b_r \quad AT = 2 * b_r$
Best case:$BT = b_r + b_s \quad AT = 2$
Improvement:M>3,M<bs,M<br
$BT = \left\lceil \frac{b_r}{M-2} \right\rceil * b_s + b_r \quad AT = 2\left\lceil \frac{b_r}{M-2} \right\rceil$(-2 是给内层关系留一个,最终结果留一个,其他全部给外层M = memory size in blocks)
Indexed Nested-Loop Join
Worst:$time = b_r * (t_T + t_S) + n_r * c$
$BT = AT = b_r + n_r * c$
C is the cost of traversing index and fetching all matching s tuples for one tuple or r.
In B+ index, c=height+1
C can be estimated as cost of a single selection on s using the join condition
Merge Join
Sort first, then join the two relation. Can be used only for equal-joins and natural joins. BT= $b_r + b_s \quad AT = \left\lceil \frac{b_r}{b_b} \right\rceil + \left\lceil \frac{b_s}{b_b} \right\rceil$(not including sort)
$b_b \geq \sqrt{b_s + b_r}$
Hash Join
$n_h = f * \left\lceil \frac{b_r}{M} \right\rceil$,f 一般为 1.2;在$M > n_h + 1$ 或者简化为 $M > \sqrt{b_s}$时不使用递归划分
no recursive BT=3(b_r + b_s) +4 ∗ n_h AT=2(⌈b_r / b_b⌉+⌈b_s / b_b⌉)
recursive:
# of passes required for partitioning build relation s is⌈$\log_{M–1}(b_s)$ – 1⌉,choose the smaller relation as the build relation(here is r). Total cost estimate is: BT=2(b_r + b_s)⌈$\log_{M–1}(b_s)$ – 1⌉+ b_r + b_s AT=2(⌈b_r / b_b⌉+⌈b_s / b_b⌉)⌈$\log_{M–1}(b_s)$ – 1⌉
If the entire build input can be kept in main memory no partitioning is required
BT=b_r + b_s. – best case

Statistical Information for Cost Estimation
(1) Catalog Information about relation
nr : number of tuples in a relation r.
br : number of blocks containing tuples of r.
fr : blocking factor of r — i.e., the number of tuples of r that fit into one block.
lr : number of bytes for a tuple in r
ls: number of bytes for a tuple in s
V(A, r): same as the size of ∏A(r).
$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$
$\sigma_{A=v}(r)$ n_r/V(A, r)如果有直方图,用区间频数代替 nr,区间中属性 A 不同值的个数代替 V
$\sigma_{A \leq v}$: $n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
不知道 v 时,c is assumed to be nr / 2.
Complex Selections
If Si is the number of tuples in r that satisfy θi , the selectivity 概率 of θi is Si / nr
Conjunction: σθ1∧ θ2∧. . . ∧ θn (r).
$$n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$$
σ¬θ(r). nr(1 - Si / nr)
Disjunction:σθ1∨ θ2 ∨. . . ∨ θn (r).
$$n_r * \left(1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r})\right)$$
Estimation of the Size of Joins
1) The Cartesian product r x s produces $n_r \bullet n_s$ tuples; each tuple occupies lr + ls bytes.
If R ∩ S = ∅, then r |x|s is the same as r x s.
2)若连接属性是 r 的候选码,则连接结果得到的行数≤s 的行数
3)若连接属性是 r 的外码并参照 s,则结果行数=r 的行数, 结果行数最多为参照关系的行数
4) If R ∩ S = {A} is not a key for R nor S.
$\frac{n_r * n_s}{V(A,s)}$ $\frac{n_r * n_s}{V(A,r)}$ If V(A, r)≠ V(A,s), 选结果小的
Equivalence Rules: σθ1(E1|x|θ2 E2) = E1|x|θ1∧ θ2 E2; (E1|x|E2) |x|E3 = E1|x|(E2|x| E3); (E1|x|θ1 E2)|x|θ2 ∧ θ3E3 = E1|x|θ1∧ θ3(E2|x|θ2 E3) where θ2 involves attributes from only E2 and E3.选择分配律:先连后选=>先选后连
all attributes in θ0 involve only attributes of one of the expressions being joined.
   σθ0(E1 |x|θ E2) = (σθ0(E1)) |x|θ E2
(b) When θ1 involves only the attributes of E1 and θ2 involves only the attributes of E2.
  σθ1∧θ (E1 |x|θ E2) = (σθ1(E1))|x| θ (σθ (E2))

Transaction
ACID 性质:
1. Atomicity. Either all operations of the transaction are properly reflected in the database or none are.
2. Consistency. Execution of a transaction in isolation preserves the consistency of the database.
3. Isolation. Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
That is, for every pair of transactions Ti and Tj, it appears to Ti that either Tj, finished execution before Ti started, or Tj started execution after Ti finished.
4. Durability. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
事务状态
Active - the initial state.
Partially committed - after the final statement has been executed.
Failed - after finding the normal execution can no longer proceed.
Aborted - after the transaction has been rolled back
Committed - after successful completion
Serializability
冲突操作无法交换,非冲突可以交换只有写写是非冲突的.优先图中被指向的一方是冲突操作中晚开始的一方.可恢复调度:如果 Tj 读取了之前由 Ti 所写的数据,则 Ti 要比 Tj 先提交;无级联调度:如果 Tj 读取了之前由 Ti 所写的数据,则 Ti 要比 Tj 的这一读操作先提交;所有的无级联调度都是可恢复的;
Lock-Based ProtocolsX 读写 S 只读
2PL Protocol
Growing Phase Shrinking Phase
Lock points 最后加锁的地方 2PL 保证可串行化,按照锁点排序进行串行化
Strict 2PL:事务在提交/回滚之前一直拿着 X 锁,能够防止级联回滚
rigorous 2PL:事务在结束之前拿着所有锁不放,能够根据事务提交点进行串行化,也能防止级联回滚.
Graph-Based Protocols
if $d_i \rightarrow d_j$ then any transaction accessing both di and d_j must access d_i before access d_j.
Only X locks are allowed.
Ti 首次加锁可以对任意数据;此后 Ti 对 A 加锁时必须有 A 爸的锁;随时可以解锁;一个数据只能被 Ti 加解锁各一次.
死锁检测图:如果 Ti 等着 Tj 解锁,那么就有 Ti →Tj.
Intention Lock
Intention locks are put on all the ancestors of a node before that node is locked explicitly.

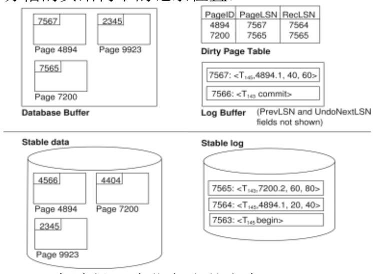
	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

Multiple Granularity Locking Scheme
Transaction Ti can lock a node Q, using the following rules:
1. The lock compatibility matrix must be observed.
2. The root of the tree must be locked first, and may be locked in any mode.
3. A node Q can be locked by Ti in S or IS mode only if the parent of Q is currently locked by Ti in either IX or IS mode.
4. A node Q can be locked by Ti in X, SIX, or IX mode only if the parent of Q is currently locked by Ti in either IX or SIX mode.
5. Ti can lock a node only if it has not previously unlocked any node (that is, Ti is two-phase ---2PL).
6. Ti can unlock a node Q only if none of the children of Q are currently locked by Ti(解锁自下而上)
Observe that locks are acquired in root-to-leaf order, whereas they are released in leaf-to-root order(加锁自顶向下,解锁自下而上,且遵守 2PL 协议)
Data item can be unlocked at any time
A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_j.
Deadlock Prevention
Wait-die 当 Ti 比 Tj 老(时间戳小),Ti 才会等待 Tj 解锁,否则就 Ti 回滚
Wound-wait 当 Ti 比 Tj 年轻(时间戳大),Ti 才会等待 Tj 解锁,否则就 Ti 回滚
Immediate Database Modification 基于 log
Undo:no<Ti commit>or<Ti abort>.Redo:have <Ti commit>or<Ti abort>. Undo first, redo next 做完 undo 要把 CLR 和 Ti abort 写到日志里

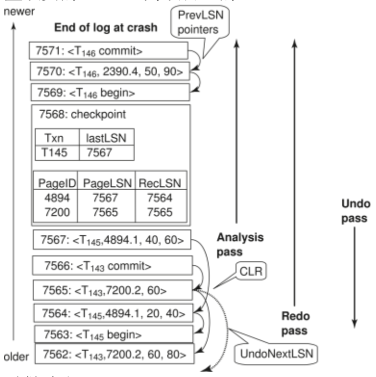


T_1 can be ignored, T_2 and T_3 redone, T_4 undone.
ARIES
数据结构

LSN to identify log records 日志顺序号 Stores LSNs in pages to identify what updates have already been applied to a database page
Physiological redo,减少日志记录的开销
Dirty page table to avoid unnecessary redos during recovery,脏页表指内存中已更新磁盘上未更新的页;
Fuzzy checkpointing 检查点只记录脏页表和活动事务列表(active transaction list),记录每个事务的 LastLSN,就是该事务所写的最后一个日志记录的 LSN.
ARIES 将日志分为多个文件,同一日志的多个文件的文件号递增,LSN 由文件号以及在该日志中的 offset 组成.
每页都有 PageLSN,当有更新操作发生在页上,该操作将其日志记录的 LSN 存在页的 PageLSN 里,在 undo 过程中,LSN≤PageLSN 的日志记录将不在该页上执行.
每个日志记录也包含同一事务的前一个日志记录的 LSN,存在 PrevLSN 里.
CLR 补偿日志记录就是前面说的 undo 后加入日志中的恢复值语句,类似于<T1,A,10,20>被 undo 了,就要加一句 CLR<T1,A,10>,CLR 还记录 UndoNextLSN,记录当书屋被回滚时,下一个需要 undo 的日志的 LSN.帮助跳过已经回滚过的日志.
脏页表为每一页保存 PageLSN 和 RecLSN,RecLSN 来标识已经实施于该页的磁盘上的日志记录,当一页被插入到脏页表中,RecLSN,被置为日志的当前末尾,若页被写回磁盘,就从脏页表移除该页.
日志中的数据项分为两部分,如 4894.1 第一部分为页码,第二部分为页中的某个记录（比如分槽的页结构中的记录位置）



ARIES 全过程,正向指朝宕的方向
分析阶段
首先找到最后的检查点记录,读入脏页表,将 RedoLSN 设为脏页表中页的 RecLSN 的最小值,若没有脏页就将其设为检查点日志记录的 LSN,重做阶段将从 RedoLSN 开始扫描日志记录,该点之前的日志记录已经反映在了磁盘上.分析阶段将 undolist 设置为检查点日志记录中的事务列表,并记录每一个事务最后一个日志记录的 LSN（也就是每个 undo 开始的位置）,继续正向扫描,如果有不在 undo 中的事务,就加到 undo 中,如果有事务 end 掉,就从 undo 中删掉.如果有某个在页上进行更新操作的日志,如果该页不在脏页表中就加进去,并设置该页的 RecLSN 为该日志的 LSN.



重做阶段
从 RedoLSN 开始正向扫,如果遇到更新操作的日志,就判断:1.如果该页不在脏页表中,或者是该操作 LSN 小于脏页表中该页的 RecLSN,跳过该日志;2.否则从磁盘中调出该页,如果磁盘页中的 PageLSN 小于该日志记录的 LSN,重做.如果以上两点有一个是不成立的,那么就无需重做.
撤销阶段
对日志进行反向扫描,对 undolist 中的所有事务进行撤销,利用分析阶段记录的最后一个 LSN 来确定 undo 起始位置,每次选这些 LSN 中最大的进行 undo.每完成一个操作的 undo 就写一个 CLR 到日志中(A 的 CLR 记为 A'),并将 CLR 的 UndoNextLSN 设为该操作记录的 PreLSN.如果遇到一个 CLR,则他的 UndoNextLSN 指明了该事务需要 undo 的下一个操作的 LSN,该 LSN 到 CLR 之间的操作已经回滚过了.除了 CLR 之外的日志记录,PreLSN 指明事务需要 undo 的下一个操作的 LSN.
Ex:后一个完整检查点指向 LSN7568,分析阶段完成时 RedoLSN 为 7564,undolist 中只有 T145,脏页包括 4894 和 7200 以及检查点后被修改的 2390.
重做阶段将从 LSN7564 开始,对出现在脏页表中的日志记录进行重做.撤销阶段对 T145 进行撤销,其 LastLSN 为 7567.从 7567 开始反向扫描到 LSN7563 遇到 T145 start.
Aries 其他特性:
嵌套的顶层动作:允许某些操作即使回滚也不被撤销（通过虚拟 CLR）
回复的独立性:使某页独立恢复
保存点:允许部分回滚到某个保存点,对于死锁处理特别有用
细粒度的封锁:允许索引元素级的封锁而不是页级的封锁,大大提高并发性
恢复最优化:重做的顺序和时间非常科学.
XML
无子元素的元素可以不用头尾 tag,直接一个 tag 并且可以使用属性<account number="A-101" balance="200" />
<![CDATA[<account> … </account>]]>允许这样直接打印<account>字符串
```
<bank Xmlns:FB="http://www.FB.com">
  <FB:branch>
```
XML 数据 ex:
```
<bookstore>
  <book ISBN = "7-04-011049-0">
    <title> Dadatbase System Concepts </title>
    <author> Abraham Silberschatz </author>
    <author> Henry F. Korth </author>
    <author> S.Sudarshan </author>
    <price> 59.5 </price>
  </book>
</bookstore>
```
DTD 语法
<!ELEMENT element (subele-specification)>
<!ATTLIST element (attributes)>
元素语法例子
<!ELEMENT bank ((account | depositor)+)>
<!ELEMENT message (to+ from, cc*, sub, text)>
<!ELEMENT to (#PCDATA)>
names of elements , #PCDATA (parsed character data), EMPTY (no subelements) or ANY (anything can be a subelement), "|" - alternatives, "+" - 1 or more occurrences "*" - 0 or more occurrences, "?" - 0 or 1 occurrences
属性语法例子:
<!ATTLIST name1 type1 default1|#REQUIRED|#IMPLIED… >
类型可以是 CDATA, ID, IDREF, IDREFS
<!ATTLIST account acct-type CDATA "checking">
<!ATTLIST customer customer-id ID #REQUIRED accounts IDREFS #REQUIRED>
DTDex:
```
<!DOCTYPE research-proj [
  <!ELEMENT research (project +, developer + )>
  <!ELEMENT project (pname, budget, from,
to )>
    <!ATTLIST project
      pid ID # REQUIRED
      members IDREFS # REQUIRED >
  <! ELEMENT pname (#PCDATA)>
  <! ELEMENT budget (#PCDATA)>
  <! ELEMENT from (#PCDATA)>
  <! ELEMENT to (#PCDATA)>
  <!ELEMENT developer(dname,age)>
    <!ATTLIST developer
      did ID # REQUIRED >
  <! ELEMENT dname (#PCDATA)>
  <! ELEMENT age (#PCDATA)>
]>
```
XML Schema
```
<xs:schema xmlns:xs="www.">
<xs:element name ='dept'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='name' type='xs:string'>
      <xs:element name='budget'
type='xs:decimal'>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```
XPath
/bank-2/customer/customer_name evaluated on the bank-2 data we saw earlier returns:
<customer_name> Joe </customer_name>
<customer_name> Mar</customer_name>
/bank-2/customer/customer_name/text( ) Returns Joe Mar
/bank-2/account[balance > 400] Returns account elements with a balance value greater than 400
/bank-2/account[balance] Returns account elements containing a balance sub-element.
/bank-2/account[balance >400] /@account_number Returns the account numbers of accounts with balance > 400
/bank-2/account[count(./customer) > 2] Returns accounts with >2 customers
/bank-2/account/id(@owner) Returns all customers referred to from the owners attribute of account elements
Boolean connectives and and or and function not() can be used in predicates.
"|" used to implement union. "//" can be used to skip multiple levels of nodes. ".." specifies the parent. doc(name) returns the root of a named document.
doc("university.xml")/university/department
XQuery
for … let … where … order by …result …分别对应 from, rename, where, order by, select.{}里才是查询出的东西,其他 return 都是字符串
ex let 句子
```
for $x in /bank-2/account
let $acctno := $x/@account_number
where $x/balance > 400
return <account_number> { $acctno }
</account_number>
```
ex 嵌套查询
```
<bank-1> { for $c in /bank/customer return
<customer> {$c/*} {for $d in
/bank/depositor[customer_name =
$c/customer_name], $a in
/bank/account[account_number=$d/account_n
umber] return $a} </customer> } </bank-1>
```
Ex group by 的实现
```
For $d in /university/department return
<department-total-salary>
<dept_name>{$d/[dept_name}</dept_name>
<total_salary>{fn:sum( for $i in
/university/instructor[dept_name=$d/dept_na
me] return $i/salary)}</total_salary>
</department-total-salary>
```
找出每个系中所有教师的薪酬和
Order by 可以用 descending 降序
XSLT
```
<xsl:template match = "/bank-2/customer">
   <customer>
   <xsl:value-of select = "customer_name"/>
   </customer>
</xsl;template>
   <xsl:template match = "*"/>( matches all
elements that do not match any other
template)
```
结果为
```
<customer> Joe   </customer>
<customer> Mary </customer>
```
递归
```
<xsl:template match = "/bank">
   <customers>
   <xsl:template apply-templates/>
   </customers >
</xsl:template>
   <xsl:template match = "/customer">
   <customer>
      <xsl:value-of select = "customer_name"/>
   </customer>
</xsl:template>
   <xsl:template match = "*"/>
```
连接
```
<xsl:key name = "acctno" match = "account"
use = "account_number"/>
<xsl:key name = "custno" match = "customer"
use = "customer_name"/>
<xsl:template match = "depositor">
  <cust_acct>
  <xsl:value-of select = key("custno",
"customer_name")/>
  <xsl:value-of select = key("acctno",
"account_number")/>
  </cust_acct>
</xsl:template>
<xsl:template match = "*"/>
```
排序
```
<xsl:template match = "/bank">
  <xsl:apply-templates select = "customer">
  <xsl:sort select = "customer_name"/>
  </xsl:apply-templates>
</xsl:template>
<xsl:template match = "customer">
  <customer>
    <xsl:value-of select = "customer_name"/>
    <xsl:value-of select = "customer_street"/>
    <xsl:value-of select = "customer_city"/>
  </customer>
<xsl:template>
<xsl:template match = "*"/>
```
DTD 和 XML schema 的 ex
```
<!DOCTYPE parts [
   <!ELEMENT part (name, subpartinfo*)>
   <!ELEMENT subpartinfo (part, quantity)>
   <!ELEMENT name ( #PCDATA )>
   <!ELEMENT quantity ( #PCDATA )>
]
```
corresponding XML schema
```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchem
a">
  <xs:element name="parts" type="partsType"/>
  <xs:complexType name="partType">
    <xs:sequence>
      <xs:element name="name"
type="xs:string"/>
      <xs:element name="subpartinfo"
type="subpartinfoType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="subpartinfoType">
    <xs:sequence>
      <xs:element name="part"
type="partType"/>
      <xs:element name="quantity"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
An XML data corresponding to this DTD
```
<parts>
  <part>
    <name> watermelon </name>
    <subpartinfo>
      <part>
        <name> pulp </name>
      </part>
      <quantity> 1 </quantity>
    </subpartinfo>
  </part>
  <part>
</parts>
```
路径表达式
bookstore 选取 bookstore 元素的所有子节点
/bookstore 选取根元素 bookstore. 假如路径起始于正斜杠(/),则此路径始终代表到某元素的绝对路径.
bookstore/book 选取所有属于 bookstore 的子元素的 book 元素.
//book 选取所有 book 子元素,而不管它们在文档中的位置.
bookstore//book 选择所有属于 bookstore 元素的后代的 book 元素,而不管它们位于 bookstore 之下的什么位置.
//@lang 选取所有名为 lang 的属性.
/bookstore/book[1] 选取属于 bookstore 子元素的第一个 book 元素.
/bookstore/book[last()-1]选取 bookstore 子元素的倒数第二个 book 元素
/bookstore/book[position()<3] 选取最前面的两个属于 bookstore 元素的子元素的 book 元素.
//title[@lang] 选取所有拥有名为 lang 的属性的 title 元素.
//title[@lang='eng'] 选取所有 title 元素,且这些元素拥有值为 eng 的 lang 属性.
/bookstore/book[price>35.00]/title 选取所有 bookstore 元素中的 book 元素的 title 元素,且其中的 price 元素的值须大于 35.00.
bookstore/* 选取 bookstore 元素的所有子节点
//title[@*]　选取所有带有属性的 title 元素.
//book/title | //book/price 选取所有 book 元素的 tilte 和 price 元素.