Relational Algebra

Rename：$\rho_x(E)$ or $\rho_{x(A1,A2,\ldots An)}(E)$　$\$\ i$　theta join

Aggregate：$\mathcal{G}_{\mathbf{average}(salary)}(instructor)$　count_distinct

Groupby: $_{G_1,G_2,\ldots,G_n}\mathcal{G}_{F_1(A_1),\ F_2(A_2),\ldots,\ F_m(A_m)}(E)$

$r \div s$ is the largest relation t(R-S) $t \times s \subseteq r$

SQL

create table T(A1 D1, A2 D2 , <C1> , <C2>);

<C>: primary key (A*k*)　| unique(A1, A2..) | check(P)

<C>: foreign key (Ak) references T(Aj)

insert into T values (V1, ...);

delete from T;　drop table T;

alter table T add A D;　alter table T drop A;

select ***distinct / all***　A from T where C;

select * from R natural join S natural join T;

select * from (R natural join S) join T using(A1,A2...);

select R.A1, S.B1 from RR as R, SS as S;

字符串函数："||"，upper(s)，lower(s)，trim(s)

"%"匹配任意子串 "_"匹配任意一个字符　\转义

like　order by A desc, B asc　between C and D

集合运算通过 union / intersect / except　all 保留重复

avg min max sum count count(distinct A)

除了 count(*)外的聚集函数都忽略输入集合中的 null 值

Nested Subqueries:

select ... from ... where A in (SFW);

select ... from ... where A > some / all (SFW);

select ... from ... where exists / not exists (SFW);

select ... from (SFW) where ...;

with R(A1,A2...) as (SFW) SFW;

Scalar Subquery: 只要子查询只返回单属性单元组

select A, (select C(*) from S where R.A=S.A) from R;

update S set A = (SFW);

insert into T SFW; 将一张表信息插入到另一张表

update T set A = case

when P1 then R1 when P2 then R2 else R0 end

select * from R join S on R.A = S.A;

natural left / right / full outer join

create view V as <Query Expression>;

create assertion AS check (not exists (~P));

grant <权限>(属性) on <关系/视图> to <用户/角色>

revoke<权限>(属性) on <> from <> [restrict]

ER & DB Design

强实体与弱实体的联系只能是 1：1 或 1：N。

无损分解条件：$R_1 \cap R_2 \to R_1$or $R_1 \cap R_2 \to R_2$

平凡 trivial：$\alpha \to \beta \Leftarrow \beta \subseteq \alpha$

检查函数依赖： $\alpha \to \beta \subseteq F^+ \Leftrightarrow \beta \subseteq \alpha^+$

判断超码： $\alpha \to R \subseteq F^+ \Leftrightarrow R \subseteq \alpha^+$

计算闭包： $\gamma \subseteq R$，　$S \subseteq \gamma^+ => \gamma \to S$

Canonical Cover: $\alpha \to \beta$ in F =>表示逻辑蕴含

if $A \in \alpha\ F \Rightarrow (F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$

if $A \in \beta, F \Rightarrow (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$

测试属性 $A \in \alpha$是否是多余的(extraneous)：

compute $T = (\{\alpha\} - A)^+$　check $T \supseteq \beta$ *?*

测试属性 $A \in \beta$是否是多余的：

compute α^+ using $(F - \{\alpha \to \beta\}) \cup (\alpha - \{\beta - A\})$

check $\alpha^+ \supseteq \beta$ *?*

BCNF 范式：对于 F+中任意 FD $\alpha \to \beta$，要么该 FD 是平凡的，要么该α是 R 的 superkey。

BCNF 分解：①$\alpha \cup \beta$②$R - (\beta - \alpha)$

3NF：β-α中的每个属性 A 都包含 R 的一个候选码中。

保持依赖：验证每一个关系 Ri 对应的函数依赖的并集的闭包与原闭包相同。

XML

default namespace：<tag xmls:zju='www.zju.edu.cn'>

namespace：<yale:course>

存储包含标签的值：<![CDATA [<tag>...</tag>]] >

<!DOCTYPE university [

<!ELEMENT university ((department|course)+)>

<!ELEMENT course(cid, title, dept, credits)>

<!ELEMENT cid(#PCDATA)>

] >

属性：<!ATTILIST course cid CDATA #REQUIRED ...>

CDATA ID IDREF IDREFS　#IMPLIED

XPath：/university-3/instructor/name 返回带标签的

/university-3/instructor/name/text()不带外围标签

访问属性：/unicersity-3/course@course_id

/university-3/course[credits>=4]/@course_id

/university-3/course[@dept_nam='cs']

/university-2/instructor[count(./teaches/course)>2]

id("foo")返回属性类型为 ID 且值为 foo 的结点

/university-3/course/id(@dept_name) 返回被 course 元素的 dept_name 属性引用的所有系的元素。

/university-3/course/id(@instructor) 返回被 course 元素的 instructors 属性所引用的教师元素。

合并使用"|"，按照在文档中出现的顺序返回。

跳过多层："**//**"　**doc(name)**返回文档的根 往上找：**$x../@cno**

XQuery：for $x in /university-3/course

let $cid:=$x/@course_id where $x/credits>3

return <course_id>{$cid}</course_id>

return <course course_id="{$x/@course_id}" />

return element course {

attribute course_id {$x/@course_id},

element title {$x/credits} }

for $c in /university/course, $i in /university/instructor,

$t in /university/teaches where $c/cid=$t/cid

where some/every $e in path satisfies P

for $i in /university/instructor order by $i/name

return <instructor>{$i/*}</instructor>

B+ Tree

Leaf node: $\left[\left\lceil\frac{n-1}{2}\right\rceil, n-1\right]$ values

Non-leaf node: $\left[\left\lceil\frac{n}{2}\right\rceil, n\right]$ childrent

Height: $\left\lceil log_{M-1}\frac{N}{M}\right\rceil \approx \lceil log_M N\rceil \le h \le \left\lfloor \log_{\lfloor M/2\rfloor} N/2\right\rfloor + 1$

Size estimate:

$$\left\lceil\frac{K}{n-1}\right\rceil + \left[\left\lceil\frac{K}{n-1}\right\rceil * \left\lceil\frac{1}{n}\right\rceil\right] + \left[\left\lceil\frac{K}{n-1}\right\rceil * \left\lceil\frac{1}{n^2}\right\rceil\right] + \ldots + 1 \le Size$$

$$Size \le \left\lceil\frac{K}{\lceil\frac{n-1}{2}\rceil}\right\rceil + \left[\left\lceil\frac{K}{\lceil\frac{n-1}{2}\rceil}\right\rceil * \left\lceil\frac{1}{\lceil\frac{n}{2}\rceil}\right\rceil\right] + \left[\left\lceil\frac{K}{\lceil\frac{n-1}{2}\rceil}\right\rceil * \left\lceil\frac{1}{\lceil\frac{n}{2}\rceil^2}\right\rceil\right] + \ldots + 1$$

Query Processing

A1：线性搜索 Cost = b_r block transfer + 1 seek

A1：线性搜索，码属性等值比较

average Cost = $\left(\frac{b_r}{2}\right)$ block transfer + 1 seek

A2：B+树主索引，码属性等值比较　cost = $(h_i + 1) * (t_T + t_S)$

A3：B+树主索引，非码属性等值比较。其中 b 是包含指定搜索码记录的块数。cost = $h_i * (h_T + h_S) + t_S + t_T * b$

A4：B+树辅助索引，码属性等值比较 cost = $(h_i + 1) * (t_T + t_S)$

A4：B+树辅助索引，非码属性等值比较，n 为记录数

(worst case) cost = $(h_i + n) * (t_T + t_S)$

A5：B+树主索引，比较。同 A3 非码属性等值比较。

A6：B+树辅助索引，比较。同 A4 非码属性等值比较。

Select for A5 & A6:

for $\sigma_A \ge v(r)$ use index to find first tuple >=v and scan relation sequentially

for $\sigma_A \le (r)$ just scan relation sequentially till first tuple > v; do not use index.

External Merge Sort:

Total number of merge passes required: $\left\lceil \log_{M-1}\left(\frac{b_r}{M}\right)\right\rceil$

Total number of block transfer: $b_r\left(2\left\lceil \log_{M-1}\left(\frac{b_r}{M}\right)\right\rceil + 1\right)$

Cost of seeks $2\left\lceil\frac{b_r}{M}\right\rceil + \left\lceil\frac{b_r}{b_b}\right\rceil * \left(2\left\lceil \log_{M-1}\left(\frac{b_r}{M}\right)\right\rceil - 1\right)$

bb 是每次从一个 run 中读取的缓冲块数，M=3 时为 1

Some times M in log is $\lfloor M/bb\rfloor$

Join Operation

r is called outer relation, s is inner relation of the join

Nested-Loop Join:

$n_r * b_s + b_r$block transfers + $(n_r + b_r)$ seeks

Block Nested-Loop Join:

worst case: $b_r * b_s + b_r$ block transfer + $2 * b_r$ seeks

Best case: $b_r + b_s$ block transfer + 2 seeks

Improvement:

Cost = $\left\lceil\frac{b_r}{M-2}\right\rceil * b_s + b_r$ block transfers + $2\left\lceil\frac{b_r}{M-2}\right\rceil$ seeks

Indexed Nested-Loop Join: Cost = $b_r * (t_T + t_S) + n_r * c$

Merge Join：记得先排序

Cost = $b_r + b_s$ block transfers + $\left\lceil\frac{b_r}{b_b}\right\rceil + \left\lceil\frac{b_s}{b_b}\right\rceil$ seeks

Hash Join:

无需递归划分 Cost = $3(b_r+b_s)+4n_k$　transfer

$+2\lceil br/bb\rceil + \lceil bs/bb\rceil + 2n_k$seeks

需要递归划分：$2(br + bs)\lceil log_{M-1}(bs - 1)\rceil + br + bs$ transfer

$+\ 2(\lceil br/bb\rceil + \lceil bs/bb\rceil) + \lceil log_{M-1}(bs) - 1\rceil$ seeks

Estimate Size of Expr: V(A, r): size of Project(A(r)).

$b(r) = \lceil n(r)/f(r)\rceil$　　$\sigma_{A=v}(r) = n(r)/V(A,r)$

$$\sigma_{A \ge v}(r) = n(r) \times \frac{v - Min(A,r)}{Max(A,r) - Min(A,r)}$$

若 $R \cap S$ 是 R 的码，则连接后最多不会超过 n(s)

若 $R \cap S = \{A\}$，有：$n(r)n(s)/V(A,s)$　注意顺序

Concurrency Control:

可恢复调度：如果 Tj 读取了 Ti 所写的数据项，那么 Ti 应该先于 Tj 提交。

无级联调度：如果 Tj 读取了 Ti 所写的数据项，那么 Ti 必须在 Tj 这一读操作前提交。

Lock-Based Protocols

exclusive mode：　locked(X)

shared mode：　locked(S)

Two-Phase Locking Protocol

Growing Phase：obtain locks，not release locks

Shrinking Phase：release locks，not obtain locks

Lock points：from obtaining locks to releasing locks.

strict two-phase locking: 严格两阶段封锁协议必须要求事务所有的排他锁必须在事物提交后方可释放。可以避免级联回滚。

rigorous two-phase locking: 强两阶段封锁协议要求事务提交之前不得释放任何锁。

upgrade：锁从 S 转换到 X，只能发生在增长阶段。

downgrade：锁从 X 转换到 S，只能发生在缩减阶段。

2PL 是保证事务调度冲突可串行性的充分条件，而非必要条件。

DeadLock

wait-die：当 Ti<Tj 时允许 Ti 等待 wound-wait 相反

等待图：Ti→Tj 表示 Ti 在等待 Tj 释放所需数据项

Multiple Granularity

intention-shared(IS)　intention-exclusive(IX)

shared and intention-exclusive(SIX)：以该节点为根的子树增加排他锁

Mode	NL	IS	IX	S	SIX	X
NL	Yes	Yes	Yes	Yes	Yes	Yes
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	Yes	No	No	No
S	Yes	Yes	No	Yes	No	No
SIX	Yes	Yes	No	No	No	No
X	Yes	No	No	No	No	No

The look compatibility matrix must be observed.

The root of the tree must be locked first, and may be locked in any mode.

A node Q can be locked by T_i in S or IS mode only is the parent of Q is currently locked by T_i in either IX or IS mode.

A node Q can be locked by T_i in X, SIX, or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.

T_i can lock a node only if it has not previously unlocked and node(two-phase)

T_i can unlock a node Q only if none of the children of Q are currently locked by T_i.

Locks are acquired in root-to-leaf order, whereas they are released in leaf-to-root order.

Recovery Method

Log-Based Recovery

1. Register itself by writing a $< T_i$ start $>$ log record
2. Before executes write(X), a log record $< T_i, X, V_1, V_2 >$ is written, where V_1 is the value of X before the write, and V_2 is the value to be written to X.
3. When T_i finishes it last statement, the log record $< T_i$ commit $>$ is written.

checkpoint

✧ Output all log records currently residing in main memory onto stable storage.

✧ Output all modified buffer blocks to the disk.

✧ Write a log record < checkpoint L> onto stable storage.

✧ L 是执行检查点时正在活跃的事务列表，记 T 为 L 和 L 之后的事务，对任意事务 Tk，若没有<Tk commit>或者<Tk abort>则 Undo(Tk)，否则 Redo(Tk)

Transaction rollback (during normal operation)

Scan log backwards from the end , for each <Ti, Xj,V1,V2>

1. perform the undo by writing V1 to Xj,
2. write a log record <Ti , Xj,V1>
3. Once the record <Ti start> is found stop the scan and write the log record <Ti abort>

Recovery from failure: Two phases

Redo phase: Find last <checkpoint L> record, and set undo-list to L. Scan forward from above <checkpoint L> record

1. when found <Ti, Xj, V1, V2> , redo it by writing V2 to Xj
2. when found <Ti start> , add Ti to undo-list
3. when found <Ti commit> or <Ti abort> , remove Ti from undo-list

Undo phase: Scan log backwards from end . When found <Ti, Xj, V1, V2> and Ti is in undo-list perform rollback:

1. perform undo by writing V1 to Xj.
2. write a log record <Ti , Xj, V1>
3. When found <Ti start> , Write <Ti abort> and Remove Ti from undo list.
4. Stop when undo-list is empty.

ARIES

- 使用一个日志顺序号 LSN 来标识日志记录，并将 LSN 存储在数据库页中，来标识哪些操作已经在一个数据库页上实施过了。一个 LSN 由一个文件号以及在该文件中的偏移量组成。
- 每一页维护一个页日志顺序号(PageLSN)的标识，每当一个更新操作发生在某页上时，该操作将其日志记录 LSN 存储在该页的 PageLSN 域中。在恢复撤销阶段，LSN≤PageLSN（较老的记录）将不在该页上执行。
- 每个日志记录包含同一事务的前一日志记录的 LSN 值，并存放在 PreLSN 中。
- 事务回滚中会产生一些特殊的 redo-only 记录，称为补偿日志记录(CLR)。CLR 有一个额外的字段，叫做 UndoNextLSN，记录当事务被回滚时日志中下一个需要 undo 的日志的 LSN。
- 脏页表为每一页保存其 PageLSN 和 RecLSN，当一页插入到脏页表时，RecLSN 的值被设置成日志的当前末尾。只要页被写入磁盘，就从脏页表中移除该页。
- Checkpoint 中含有 Dirty Page Table 和 Active Transaction List
- Why checkpoint operation of Aries puts less side effects on normal transaction processing of DBMS? Doesn't output dirty pages in buffer to disk during checkpointing.

Analysis pass:

Starts from last complete checkpoint log record

Sets RedoLSN = min of RecLSNs of all pages in DirtyPageTable

(If no pages are dirty, RedoLSN = checkpoint record's LSN)

Sets undo-list = list of transactions in checkpoint log record

Scans forward from checkpoint and complete undo-list

Whenever an update log record is found

If page is not in DirtyPageTable, it is added with RecLSN set to LSN of the update log record. else update PageLSN.

Redo pass:

Scans forward from RedoLSN.

If the page is not in DirtyPageTable or the LSN of the log record is less than the RecLSN of the page in DirtyPageTable, then skip the log record.

Otherwise fetch the page from disk. If the PageLSN of the page fetched from disk is less than the LSN of the log record, redo the log record.

Undo pass:

Backward scan optimized by skipping unneeded log records.

对 undo-list 的每个事务进行撤销。用分析阶段的最后一个 LSN 来找到每个日志的最后一个记录、

撤销阶段产生一个包含 undo 执行动作的 CLR，并将该 CLR 的 UndoNextLSN 设置为该更新日志的 LSN 值。

撤销产生 CLR 记录<T, X, V>，撤销结束产生<T, abort>

触发器

create trigger timeslot_check1 after **insert*[delete]*** on section

referencing new row as nrow

for each row

when (.....)

begin

　　rollback

end;

create trigger setnull_trigger before update of takes

referencing new ***[old]*** row as nrow

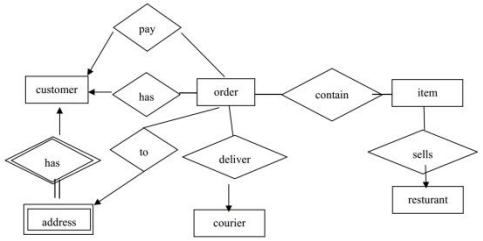
for each row

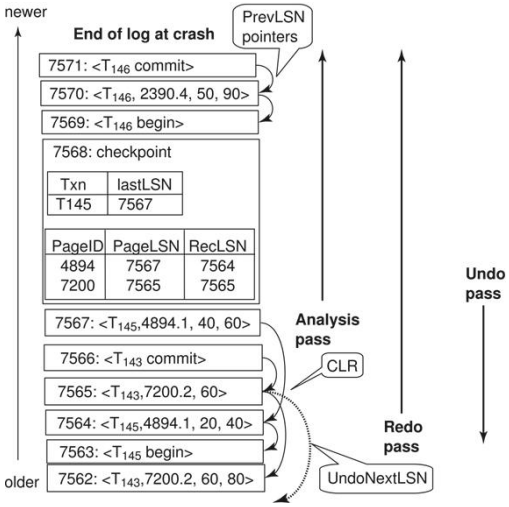
　when (nrow.grade ='')

　　begin atomic

　　　set nrow.grade = null;

　　end;





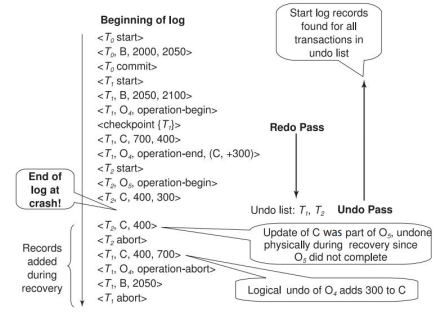


Figure 16.7 Failure recovery actions with logical undo operations

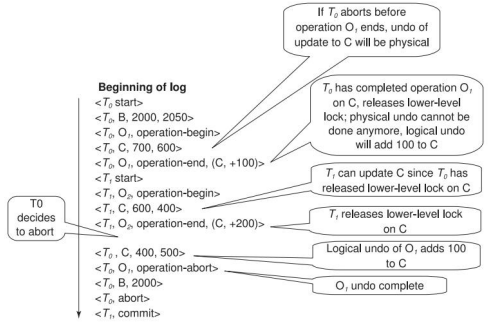


Figure 16.6 Transaction rollback with logical undo operations.