Chapter 1

fs: ① data redundancy
② Data isolation; inconsistency
③ accessing data 困难 ④ Integrity
⑤ Atomicity ⑥ Concurrent Access
⑦ Security

DB的特点：数据持久、访问便利、完整，多用户并发控制，故障恢复，安全控制

DB的那些问题解决 fs？

Chapter 2 Relational Model

relation r: $D_1 \times D_2 \times \cdots \times D_n$ 子集
└ tuple 的 set. unorder

attribute 取值：domain. key nonkey

superkey、candidate key. 外键约束/引用完整性

$\sigma . \Pi . \cup . - . \times . \rho_x$
 ρ: └ rename?

$\bowtie$ ⟗ ⋉ 柄合 $\div$ $\leftarrow$

分组属性（保留） $_{a_i}\mathcal{G}_{F(E)}$

and 入 可以 null (PK除外)

Chapter 3 SQL

DDL: char(n), varchar(n). int. smallint.
numeric(p,d), real float(n),
共p位，.后d位 └ ≥ n digits

builtin: time, date, timestamp, interval

create table: primary key (A₁, A₂ ...),
 foreign key (A₁, A₂ ...) references r (A)
 default. not null, unique primary key.
 on delete/update cascade/set null/default/restrict

(默认 null) 引用

alter table r add A D
 drop A

大小写不敏感 (table, attr)

★★★ distinct/all

where: between and.
 like '%串 _符'

natural join, join ... using(A) 属性 on 条件
inner join
left/r/full outer join

limit 限制返回数量

order by desc/asc.

except. union. intersect 去重
 加 all 保留

null: unknown (is)

select 中属性必出现在 group by 中. 或聚合函数 也可以
having (先 where 组 - have)

set membership: in/not in
 comparison >some/all 标量 要求返回一个
 可用于子查询
 exists, unique → ⚠ empty 也是 true ?

delete from r where
insert into r values _
 select

update r set ... where
 a = case
 when _ then _
 else _
 end

Chapter 4 Intermediate SQL

默认 (inner) join grant by current role.

create type __ as __
 domain 目的类型
 └ 可以有约束 not null constraint + check

integrity constraint:
check(_ in (_)) e.g. dept references ...
create assertion _ check ...

create view as ...
create/drop index _ on _

grant <privilege> on _ to _ 给的是 (e.g. public) (with grant option)
revoke .... from ... cascade/restrict
role

Chapter 5 Advanced SQL

JDBC: getConnection, createStatement
executeUpdate ... rset. getString
 next
 wasnull
SQLJ: (嵌入 java)
ODBC

create trigger before/after delete/insert/update of A on r
 orow - nrow
 referencing new/old row as
 for each row 每个改变的 row
 begin ... end

create function __ (name type ...) returns __
 procedure 没有返回值 而是 in/out 参数
begin ... end 中间可以 declare, return
调用：call procedure/func 与 C 相同
 while ... do set ... end while
 repeat ... until ... end repeat
可以返回一个 table for _ as do set ... end for

Chapter 6 Entity-Relational Model

□ entity degree: n元
◇ relationship attribute: composite
 multivalued 多值: {phone}
⊟ attribute

(一) 一对一 A ← B: 1A 多B — 多对多
二 全参与 l..h (上下限)
 如果没有属性 多元关系只有一个 → 多对多, 多值属性 必须转为单独 schema
id 分辨序号
◈ 标识性实体集 □—◈—□
归约为关系模式：一对多可以在多 加上一的属性(可能有 null)

特化 ↗ overlapping
 disjoint

Chap 7 Relational DB Design

pitfall: 信息重复，插入异常，更新困难

α → β dependency trivial: β ⊆ α

① $\beta \subseteq \alpha \Rightarrow \alpha \to \beta$ 自反
② $\alpha \to \beta \Rightarrow \gamma\alpha \to \gamma\beta$ 增补 └ augmentation
③ $\alpha \to \beta, \beta \to \gamma \Rightarrow \alpha \to \gamma$ 传递
④ $\alpha \to \beta, \alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$
⑤ $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta, \alpha \to \gamma$
⑥ $\alpha \to \beta, \gamma\beta \to \delta \Rightarrow \alpha\gamma \to \delta$

canonical cover F_c.
closure F^+

依赖保持 $(F_1^+ \cup \ldots)^+ = F^+$

BCNF: 非平凡函数依赖左边是 key (无损：∩ 是 key) superkey
 └ not 依赖保持 不是包含在 key 中

3NF α → β α 是 key / β − α 的属性包含在 R 的候选码中
找 F^+, 全 split, 再去冗余

4NF: 不存在非平凡的多值依赖 也是 bcnf
$\alpha \twoheadrightarrow \beta$
$\alpha \to \beta$

Chap 12 Physical Storage Systems

volatile: cache, main memory
non- ...: flash, disk, tape
 magnetic ─ optical
 SSD

magnetic disk: track, sector
 cylinder

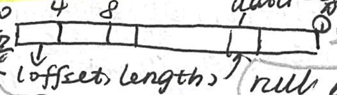
Access time = seek + rotational latency + data transfer

MTTF, Mean Time To Failure

Latency	DRAM	NVM	SSD	HDD
Read	1x	2-4x	500x	10^5x
Write	1x	2-8x	5000x	10^5x
Persistence	✗	✓	✓	✓
Byte-address	✓	✓	✗	✗
endurance	✓	✗	✗	✓

Chap 13 Data Storage Structures

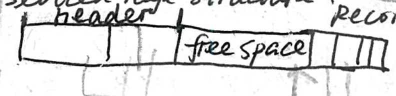
Variable-length Records length = −1 if empty



data 一条记录：① (offset, length), 定长数 ② 据 ③ 不定长数据

null bitmap: 表示属性是否为空 (1), 可存任何 pos.

Slotted Page Structure



header: ① 记录 num ② free 尾 ③ 每个记录的位置、大小

组织方式：heap, sequential, 多表聚合, B+, hash
 free-space map └ 多个表放同一个文件里

data dictionary (catalog): 关系 (table), user & accounting, 统计数据, 物理文件组织, index

database (disk) ⟷ buffer ⟷ memory
 └ LRU. Clock

面向列的存储：↓ I/O, ↑ CPU cache, ↑ 压缩效率, 向量处理;
 ↑ 元组重构, 删改 cost ↑, 解压

Chapter 14. Index

search key

Primary index: 与文件内存储次序相同 (clustering) (即 search key, 但不一定是 PK)
 secondary 辅助索引

dense index 每个 key 一个索引项
sparse: 部分 key 才有索引项 — space, 维护代价 ↓

B+ Tree

① 内点 ⌈n/2⌉ ~ n 儿子
② 叶子 ⌈(n−1)/2⌉ ~ n−1 values (keys)
③ root (1) nonleaf ≥ 2 儿子
 (2) Leaf 0 ~ (n−1) values 最多 n−1 个 key

P₁	K₁	P₂	---	Pₙ₋₁	Kₙ₋₁	Pₙ

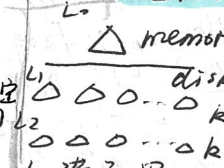
如果 K 个 key height: $\log_n(k) \sim \lceil \log_{\lceil n/2 \rceil}(k) \rceil$

internal 问题 本质是指针分配
middle key 作父节点

size estimation
4K. 10^6 records of size 68
records per block: $10^6/68$
B+ n: (4096 − 4)/(18 + 4) + 1
 key pointer
min: $2 \times \lceil \frac{n}{2} \rceil \times \cdots \lceil \frac{n}{2} \rceil \times \lceil \frac{n-1}{2} \rceil$ 多个 P
max: $n \times n \times \cdots \times n \times (n-1)$

把 pointer 换成 record ⇒ B+ 文件组织
可以3 要 half full, 个空间利用

LSM Tree



Bulk load 移了之后原树删除
只要 sequential I/O;
叶子 full, ↓ I/O 数量
但 query ↑,
每层 copy 多次

自底向上

L₀ 满了，写入 L₁ ① L₁ 为空, build
 ② L₁ 不为空, build + replace

k>1: stepped-merge index
L₀ 写入时直接建一棵新的 L₁
Lᵢ k 棵满了合并到 Lᵢ₊₁

Buffer Tree

B+: 内点有 buffer 存插入 在内存
满了才放下去
拆分也要分 buffer

查找还需要遍历内点缓冲区

随机 I/O (seek ↑)
查询开销 ↓, 可用于任意树形结构中.

满了再放下.
开始下放 (all)
(buf 里按插入顺序)

Chapter 15. Query Processing

$$\begin{tikzcd}
\boxed{\text{query}} \arrow[r] & \boxed{\text{parser translator}} \arrow[r] & \text{关系代数 exp.} \arrow[d] & \\
 & & \text{优化} \arrow[d] & \text{统计 data} \arrow[l] \\
\text{query output} & \boxed{\text{evaluation engine}} \arrow[l] & \text{执行计划} \arrow[l] &
\end{tikzcd}$$

t_T (transfers) + t_S (seek)

Select: ① Linear worst: $b_r \cdot t_T + t_S$ → 文件中块数
avg: $b_r/2 \cdot t_T + t_S$ → key

② B+主索引码上等值 $(h_i+1)\cdot(t_T+t_S)$ ③ B+主索引非码上等值 $h_i\cdot(t_T+t_S) + t_S + b\cdot t_T$ ← 符合的块数

④ B+辅助索引,码上等值 同② ④ B+辅助,非码上等值 $(h_i+m+n)\cdot(t_T+t_S)$

⑤ B+主,比较 $>$ or $\le$ 同③ ⑥ B+辅助比较 同④

Sort: M (内存里能放的页)

① 创建N个归并段 ② $N<M$: N路归并,否则多趟pass

共有 $\lceil b_r/M\rceil$ runs,要 $\lceil \log_{M-1}(b_r/M)\rceil$ 趟 — 一次 $x \to \frac{x}{M-1}$

transfer: $b_r(2\lceil\log_{M-1}(b_r/M)\rceil + 1)$ ← 流水线 少一次!!!

seek: $\underbrace{2\lceil b_r/M\rceil}_{\text{create run}} + b_r(2\lceil\log_{M-1}(b_r/M)\rceil - 1)$

一次run读出 b_b 个块,这样要 $\lceil\log_{\lfloor M/b_b\rfloor-1}(b_r/M)\rceil$ pass

transfer: $b_r(2\lceil\log_{\lfloor M/b_b\rfloor-1}(b_r/M)\rceil + 1)$

seek: $2\lceil b_r/M\rceil + \lceil b_r/b_b\rceil(2\lceil\log_{\lfloor M/b_b\rfloor-1}(b_r/M)\rceil - 1)$

Join: 外s内 ① nested-loop T: $b_r + n_r\cdot b_s$; S: $n_r + b_r$

② block nested. T: worst $b_r + b_r\cdot b_s$; S: worst $2\times b_r$
best $b_r + b_s$ best 2

☆ M块内存,M−2给r,1给s,1用来ans, $\lceil b_r/(M-2)\rceil\cdot b_s + b_r$ T + $2\lceil b_r/(M-2)\rceil$ S

③ Index: $b_r(t_T+t_S) + n_r \times c$

④ Merge: b_r+b_s T; $\lceil b_r/b_b\rceil + \lceil b_s/b_b\rceil$ S (要先排序)

⑤ hash 非递归: T: $3(b_r+b_s) + 4\cdot n_h$ S: $2(\lceil b_r/b_b\rceil + \lceil b_s/b_b\rceil) + 2\cdot n_h$

$n_h \ge \lceil b_s/M\rceil$ 分区数量 ⇒ 递归: T: $2(b_r+b_s)\lceil\log_{\lfloor M/b_b\rfloor-1}(b_s/M)\rceil + b_r + b_s$ 其中 $b_b = \lfloor \frac{M}{n_h+1} \rfloor$

S: $2(\lceil b_r/b_b\rceil + \lceil b_s/b_b\rceil)\lceil\log_{\lfloor M/b_b\rfloor-1}(b_s/M)\rceil$

best: $b_r + b_s$

先分区,再其分区hash索引,最后probe build

物化执行:产生临时文件. v.s. pipeline

$M > \frac{b_s}{M} + 1$,不用递归

Chapter 16. 查询优化

$\Pi_{L_1\cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$

前提: join的属性在 $L_1\cup L_2$ 中

$\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$ 对∩也✓ 但对∪✗

outjoin ⟗ 没有结合性(包括全)

select, project 要早做

n_r: r中元组数 b_r: r中块数. l_r: 一个元组大小、
f_r: blocking factor, 一个块放多少元组
$V(A,r)$: r中属性A的不同值的数量

$\sigma_{A=v}(r)$: $\frac{n_r}{V(A,r)}$ size estimation

$\sigma_{A\le v}(r)$: $v<\min(A,r)$: 0; $n_r \frac{v-\min(A,r)}{\max(A,r)-\min(A,r)}$ ← $\frac{n_r}{2}$ (若不知 min/max)

$r \bowtie s$ ① $R\cap S=\phi$ $n_r \cdot n_s$
② $R\cap S$ key of R $\le n_s$
③ $R\cap S$ 是S中外码(引用R) $= n_s$
④ $R\cap S = \{A\}$ 非key. $\min(\frac{n_r\cdot n_s}{V(A,s)}, \frac{n_r\cdot n_s}{V(A,r)})$

$V(A, \sigma_\theta(r)) = \min(V(A,r), n_{\sigma_\theta(r)})$

Left Deep Join Trees: 右边可以中间的,右边必须是一个关系、

Time $O(n\cdot 2^n)$, space $O(2^n)$
DP: Time $O(3^n)$ space $O(2^n)$

Chap 17. Transactions

ACID: 原子性atomic、一致性consistent.
隔离性isolation. 持久性durability.

事务状态: active, partial committed (最后一条执行完 但未提交)
failed, aborted, committed

并发问题: ①丢失修改 ②读脏数据
③不可重复读 ④幽灵问题

冲突可串行化: 冲突等价(交换不冲突的指令) 于一个串行调度 ← 对同一个数据修改冲突

冲突可串行化 ⇄ 视图可串行化

Recoverable 可恢复调度: 要在脏数据提交后再提交 避免级联rollback

Cascadeless 无级联调度: 脏数据提交后才能read ← 一定recoverable

SQL 92 set transaction isolation level
① serializable ② repeated read 不管幽灵
③ read committed 不读脏数据 ④ read uncommitted 最低

Chap 18. Concurrency Control

2-PL: 先申锁 再放 lock point 获得最后锁

2PL保证冲突可串行化 → 非必要: write

T1	T2	T3
	wC	
wA		wA

严格2PL: commit时才能放X锁
强2PL: commit时才能放所有锁

LockTable: 记录id. hash. T2等T1(x) T1→T2 grant waited

死锁: T1 T2
RxWx wy
也可Ry Wy wx

wait-for图. Ti→Tj 等j

树协议: ①只有X锁 ②第一个锁任意,随后只能锁父节点被锁住的
③随时可以unlock
④unlock过不能再lock
deadlock-free

Multiple Granularity

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	✗
IX	✓	✓	✗	✗	✗
S	✓	✗	✓	✗	✗
SIX	✓	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

SIX: S+IX

子是 S/IS,父必为 IX/IS
X/SIX/IX IX/SIX

Chap 19 Recovery System

Idempotent 幂等性: 多次执行与执行一次一样

Log record: <Ti start>, <Ti commit>, <Ti abort>
<Ti, X, V1, V2> → physical

数据写到db前,要先把日志写到stable storage

repeat history: 未提交/abort — undo. 写旧值 倒序扫描 补偿日志 <Ti, X, V> 旧值
提交 — redo — 顺序执行

checkpoint: 输出所有log, 脏数据. <checkpoint L> — redo从这开始, L 活跃事务
这样repeat只用从checkpoint开始

log record buffer: 只有当<Ti commit>这条记录进入stable storage,事务才算结束.
数据进入db前,要日志先写入.

no-force policy: 更新不一定立即刷写. steal policy: 未提交就刷写dirty

Fuzzy checkpointing: 只标记,不实际刷写. 写完后再确定

非易失存储: fuzzy/online dump 到稳定介质

提前放锁 → 逻辑操作日志 <Ti, Oi, operation-begin>, <Ti, Oi, operation-end, U>
如果没有end,就物理undo.
逻辑undo用U撤销, 随后跳过内容。
Logical undo 结束后 用 logical undo <Ti, Oi, operation-abort> 跳过 <operation-abort> (直到begin)

ARIES

lsn. pagelsn: 反映在这页的已经最近的日志号 (最近更新,可能未到disk)

LSN | TransID | PrevLSN | RedoInfo | UndoInfo

UndoNextLSN: 下一个要被撤回的日志 (跳过了abort)

PrevLSN: 同一事务的前一条日志记录.

DirtyPageTable: 每一页有
① PageLSN ② RecLSN: 被写入disk的下一个lsn: 指的page

checkpoint log record 包括脏页表和LastLSN 事务对应最后一条lsn

Algorithm

① Analysis: 决定undo-list: 哪些页是脏; 从哪个lsn开始redo
读脏页表, RedoLSN = min{RecLSN in DPT}
若没有脏页, RedoLSN = checkpoint's lsn
从checkpoint开始正向扫描,更新undo-list和DPT → RecLSN = 新写的第一条修改日志
(可能要add) 要记录每个Ti lastLSN

② Redo
从RedoLSN开始,只redo在DPT里且
从磁盘中取出 → PageLSN < log LSN ≥ RecLSN

③ Undo
用Analysis找到的每个事务的LastLSN 进行撤销,直到<Ti, start> (沿PrevLSN / UndoNextLSN)