一. Intro

OLTP OLAP
(write) (read)

view / logical / physical

DDL: 定义、约束
DML: 数据检索
DML: (过程性) 关系代数
DCL: 访问控制

数据库设计步骤:
需求 → 概念 → 逻辑 → 改善 → 物理 → create & init / 安全

component: storage, query, txn

两级架构 vs. 三级架构 √

二. Relational Model

superkey → candidate key → primary key → foreign key (references) constraints
(识别唯一元组) (最小超键) (指定候选键)

☆ Relational Algebra

优先级

投影 → 选择 → 笛卡尔积 → 连接 / 除法 → 交 → 并、差

$\Pi_{A_1 \ldots A_n}(r)$ $\sigma_P(r)$ r×s

自然连接 r⋈s 相同属性 拼接

Theta连接 $r \bowtie_\theta s$ θ条件

r∩s r∪s r−s

外连接: r、s 属性都要对上 / 保左 / 保右 / null填充

左: ⟕ 右: ⟖ 全: ⟗ 左∪右

$\Pi_{F_1 \ldots F_n}(E)$ (广义) ∧ ∨ ¬ 连接多个谓词 / 算术运算

赋值 "←" 重命名 $\rho_x(E)$ (E的结果叫作X)

聚合: $_{G_1 \ldots G_n} \mathcal{G}_{F_1(A_1) \ldots F_n(A_n)}(E)$
group by 的属性 / 聚合函数: avg, min, max, sum, count

del: r←r−E insert: r←r∪E update: r←$\Pi_{F_1 \ldots F_n}(r)$

三. SQL

关键字大小写不敏感

type: char(n) 不可变, varchar(n), int, float(n), date, time, timestamp -- 注释

表级操作: create table r(A₁ D₁, ..., Aₙ Dₙ, primary key(A₁...Aₖ), foreign key(A₁...Aₘ) references s, constraints → not null);

drop table r; (删除关系+模式)
alter table r add A D;
drop A;

查询: select <[distinct] c1, c2, ...> from <r, ...>
[where condition] 不同关系同名属性 / 连接词 and, or, not / between...and... (闭区间)
[group by <c1, c2, ...> [having condition2]]
[order by <cl [desc|asc]>, ...>] 默认

重命名: old as new

* 所有属性 / 去重 / 可作算术计算 / 笛卡尔积

执行顺序: from → where → group → having → select → order by

字符串: 用单引号, 大小写敏感, 拼接 ||, 模式匹配 like [%: 任意字符串 / _: 单个]
转义 escape '\'

limit n: 只输出前n个元组(有用!) offset m (从第m个开始)

集合运算: union, intersect, except 加all表示不去重
并 交 差

null 值 (未知/不存在) 逻辑表达式结果: true, false, unknown
其他可 unknown 或 true = true
unknown and false = false
unknown 结果均为 unknown

聚合: 在 select 子句, 但不在聚合函数内的属性必须出现在 group by

e.g. select avg(s.gpa) as avg_gpa, e.cid
from enrolled as e, student as s
where e.sid = s.sid
group by e.cid
having avg(s.gpa) > 3.9

嵌套子查询: in / not in 集合成员 / 是否在子查询 / 结果集合内
C <comp> some r (满足其一)
all r (满足全部)
exists 子查询返回结果非空为 true
unique 元组 (无重复)

with XXX as (
...
) CTE (临时表)
select ...;

delete from r where P;
insert into r values(...);
update r set A = ① [where ...]

case
when pl then d1 (或者)
...
else d0
end;

r1 natural join r2 n j ...
join...using/on <condition>
left/right/full outer join

semijoin: $\ltimes_\theta$ (r⋉s中r的那部分)
antijoin: $\overline{\ltimes}_\theta$: r−(r⋉s)
(最后简单的select语句) updatable view

视图: create view xxx as <query expression>
视图结果一直存在, 定义式
物化视图: 存储实际内容(查询), 维护

创建时不写 / 失了约束
完整性约束:
unique(A₁...Aₙ) → superkey
check(P) → 每个元组必须满足P (不能子查询) (type sys)
on delete cascade: 当S中被参照的记录删除/更新
update 级联 时, r中参照该记录的记录也要删除/更新 (chain)
constraint xxx <constr> 为约束命名
create index <name> on <relation> (<attr-list>);
drop index <name>

授权
grant/revoke <privilege> (select, insert, update, delete, all privileges)
on <relation name or view name>
to <user/role list> (public 全体用户)
create role xxx; grant <role> to <user>;
[with grant option] (可传递授权)
revoke (默认对关, 可用 cascade 显式指出)
[restrict]; (阻止级联撤回)

JDBC ODBC

函数/过程
create function dc(dn varchar(20))
returns int (也可以返回整个查询结果)
begin
declare cnt int;
select count(*) into cnt
...
return cnt;
end
-- invocation;
select dn ...
where dc(dn) > 12

create procedure dep(
in dn varchar(20)
out dc int)
begin
...
end
-- invocation
declare dc int
call dep('CS', dc);

while ... do
end while
repeat
until ...
end repeat

for ... do
end for

if ... then ...
else if ... then ...
else ...
end if

buffer manager
pin/unpin, pin count (=0 → can be evicted) block
replacement: LRU, MRU

触发器:
create trigger tcl after insert on s → delete, update
table / statement
referencing new row as nr (过渡变量)
for each row (逐行每条记录)
(触发条件) when (nr.ts_id not in(
...
))
begin
rollback
end

after update of A on s
referencing old row as orow
old row / new row 可以都要
disable trigger XXX
drop trigger XXX

Ⅲ. E-R Models

实体集 → 关系
实体 → 元组
联系集 (大多为二元)
描述属性
{(e₁, e₂) | e₁∈E₁, e₂∈E₂}
主键
逆归联系集
复杂属性
复合 (composite): 可被划分的
多值 (multivalue): 可有0个或多个值
派生 (derived): 来自其他地方

映射基数
一对一 一对多 多对一 多对多
(1个A最多和B, 1个B可有多个A)

完全/部分参与
total partial

基数约束: 0..* / 1..1
l..h (最小基数和最大基数)
*无限制

联系集主键 = 所有实体集主键 + 描述属性 (多对多)

E-R reduce to relation schema
1:1, 1:N → 0, M:N → 1

(弱) 识别实体集 / 识别联系集 / 弱实体集 / 识别唯一的实体
判别属性
(其存在性取决于识别联系和另一个实体集) → 实体
关系模式: 弱~属性 + 强~属性 (外键)

disjointness 约束 / constraints / 重叠 xx / 互斥 xx
属性继承、参与继承
泛化 / 特化
completeness constraints total / partial (default)
聚合

五. Relational Database Design

无损分解: $R = R_1 \cup R_2 \cup \cdots \cup R_n$
$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \cdots \bowtie \Pi_{R_n}(r) = R(r)$

函数依赖: α→β (t₁[α] = t₂[α] ⇒ t₁[β] = t₂[β])
(一般版) K→R ⇒ K is superkey of R
平凡依赖: α→β and β⊆α
② $R_1 \cap R_2$ is superkey of R_1 or R_2

范式 (normal form):
☆ BCNF: ∀α→β∈F⁺ (α,β⊆R), 满足以下条件之一:
① α→β是平凡的 ② α是R的superkey
依赖保留: $R = R_1 \cup R_2 \cup \cdots \cup R_n \Rightarrow F^+ = (F_1 \cup F_2 \cup \cdots \cup F_n)^+$
3NF: BCNF的条件 + β−α的属性是R的候选键成员 (可以各不同)
INF: 属性的原子性
4NF: 多值依赖
检验: NPhard

logically implied
闭包 F⁺ (从F推出来的所有函数依赖)
Armstrong's axioms
① 自反: β⊆α → α→β
② 增补: α→β → γα→γβ
③ 传递: α→β, β→γ → α→γ
additional rules: 合并: α→β, α→γ → α→βγ
分解: α→βγ → α→β, α→γ
伪传递: α→β, γβ→δ → γα→δ
R有n个属性 ⇒ $|F^+| = 2^{2n}$

BCNF分解: result = {R}
检验F中每个非平凡的α→β
若α不是superkey
分解R, 得到 (result − Rᵢ) ∪ (Rᵢ − β) ∪ (α, β)
这个Rᵢ已含以外的属性
直到 result 内所有Rᵢ都是BCNF为止

属性闭包 α⁺: 从F中可以推导出α→β的β的并集
α⁺ include all attr → α is superkey
∀γ⊆R, ∀S⊆γ⁺, γ→S的集合 ⇒ F⁺

正则覆盖: 没有多余属性 + 箭头左侧是唯一的 消除
3NF分解: 找正则覆盖 Fc (多项式时间)
for α→β in Fc, Rᵢ = (α, β)
if Rⱼ ⊆ Rₖ, 去掉 Rⱼ
结果 (R₁, R₂, ..., Rₘ)

六. Storage

cache → memory → flash → magnetic disk → CD → tape
(erase before rewriting)

RAID 0 (nonredundant), 1 (mirror), 5 (block-interleaved), 6 (P+Q)
data striping ← distribute
记录
删除: 前移, 寄新的补, free table (文件头)
(根据空余空间, 用链表维护) ECC
变长记录
定长偏移: 0 4 8 12 20 24 36 ...
变长属性位置信息 / null bitmap / 变长属性
槽式页结构: Size / Block Header / 记录属性是否为空(1为空) / Free Space / Location / End of Free Space / Records / insert new record
column-oriented
for OLAP √ I/O↓ cache perf↑ compress vector
for OLTP × reconstruct delete, update decompress costly

文件组织
① 堆 ~: 任意顺序, 插入, 删除, free-space map (多级 elem→block, level)
② 顺序 ~: order by search key (不必是 PK or SK) val: 空余空间
插入、删除 → 溢出块 (物理连续性 X → reorganize high cost)
③ multitable clustering → fast for specific joins

数据字典/系统目录: 元数据, 非规范化的

七. Indexing

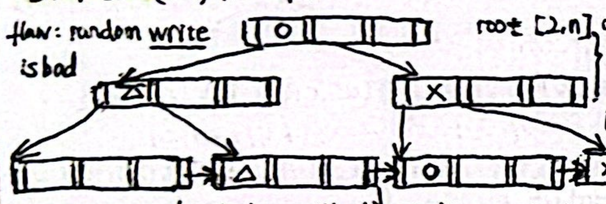
search key

clustering / primary index：搜索键顺序 = 文件内顺序
nonclustering / secondary 只访问 ≠

dense v.s. sparse：是否每个 search key 都有 index

multilevel index → binary search　　nonunique search key

B+ Tree　(n-1)sk + n ptr



flaw：random write is bad

range search　findRange(lb, ub)　≥ ⌈(n-1)/2⌉ values leaf nodes (in order)

update = delete old + insert new

find → delete (underflow) → find proper pos in leaf → insert
overflow → split leaf node (redistribute entries evenly) (0...ceil(n/2); ceil(n/2)+1...n)
merge with sibling (overflow) → redistribute (borrow one entry)

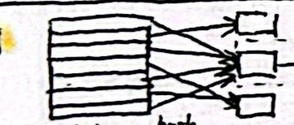
Complexity (height of tree, N entries)
best (all-full)：$H_{min} = 1 + \lceil \log_{n} N \rceil$
worst (half-full)：$H_{max} = 2 + \lfloor \log_{\lceil n/2 \rceil} \lfloor \frac{N}{(n-1)/2} \rfloor \rfloor$

B+ 树文件组织：存真实记录

bulk loading：一次插入多个项 (有序)

cost	
no recur	3(br+bs)+4nh
recursive	2(br+bs) + (overflow 的话 +br+bs)

Hashing



#of B ≥ $\frac{n_r}{f_r}(1+d)$　d：空的容量 ≈0.2

skew

多维/ADC 索引结构

LSM Trees (write 快)
Memory ← insert
Disk：L_0 is full → merge with L_1
L_1 (all entries) L_2 (level i: $\propto \log_k(\frac{r}{M})$) L_3
#of $L_{i+1} = k \cdot$ #of L_i

Buffer Trees (read 慢)
delete/update 的时候要插入 deletion/update entries 表明 entry 被 delete/update 了
internal node 插入：不到 leaf，先放 buffer，buffer 满了 → yes 再往到下一级 buffer，直到 leaf / no

Bitmap indices
0 1 1 0 1 0 1 → one val of attr
# of bits = # of records

八. Query Processing & Optimization

cost = block transfer + random I/O access (seek time)

search algos:
A1 linear search：$t_S + b_r \cdot t_T$ (equality：$t_S + \frac{b_r}{2} \cdot t_T$)
A2 clustering B+ key：$(h_i+1)(t_T+t_S)$
A2 nonkey：$h_i(t_T+t_S) + t_S + b \cdot t_T$
A3 secondary B+ key：$(h_i+1)(t_T+t_S)$
A4 nonkey：$\Rightarrow (h_i+n)(t_T+t_S)$
A5 comp → A7-A10 ∧ (1-4)
A6 comp
complex selection

height of tree　块的已决数

others
projection → ~~duplicate~~ → sort

set operation：bt：b_r+b_s，sk(worst)：b_r+b_s，can use hash
aggregation：bt：b_r　sk：1 (best，use immediate ~)　3br　2br (worst)

Evaluation 优化 → double buffering (parallel 提高 utilization)
materialization：保存中间结果到 disk → OLTP
pipelining：中间结果马上用于下次运算 → OLAP

Transformation　$\sigma_{\theta_1}(E_1 \bowtie_{\theta_0} E_2) \equiv (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_0} (\sigma_{\theta_2}(E_2))$
equivalence rules (partial)　$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$
连续有交换、结合律

cost estimate：n_r、b_r、l_r、f_r：$V(A,r)$ (# of distinct val of A in r)

histogram (distribution for attr) → assume evenly distribute
- equi-width
- equi-depth ☆ space↓

selection：$\frac{n_r}{V(A,r)}$

$\sigma_{A=a}(r)$：$\frac{n_r}{V(A,r)}$

$\sigma_{A \le V}(r) = \begin{cases} 0, & V < \min(A,r) \\ n_r, & V \ge \max(A,r) \\ n_r \cdot \frac{V - \min(A,r)}{\max(A,r) - \min(A,r)}, & \end{cases}$　# of $\Pi_A(r)$ = # of $G_{A}(r)$ = $V(G,A)$

$\sigma_{\theta_1 \wedge \dots \wedge \theta_n}(r)$：$n_r \cdot \frac{s_1 \cdots s_n}{n_r^n}$

$\sigma_{\theta_1 \vee \dots \vee \theta_n}(r)$：$n_r \left[1 - (1 - \frac{s_1}{n_r}) \cdots (1 - \frac{s_n}{n_r})\right]$　$\sigma_{\neg\theta}(r)$：$n_r - (\sigma_\theta(r)$ 的结果)

join：① $R \cap S = \emptyset \Rightarrow n_r \cdot n_s$ tuples，(l_r+l_s) bytes　② $R \cap S = R \Rightarrow \le n_s$　③ $\min(\frac{n_r \cdot n_s}{V(A,r)}, \frac{n_r \cdot n_s}{V(A,s)})$　# of $r \bowtie s \le$ # of $r \times s$ + # of r

nested-loop → block nested-loop

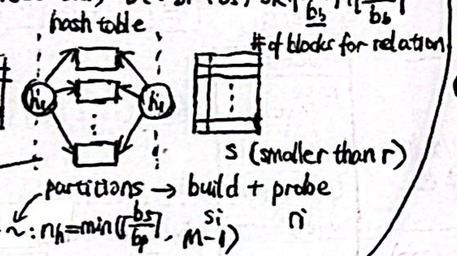
Join
r：outer relation
s：inner ~
n：# of tuples
b：# of blocks

bt：$n_r b_s + b_r$　sk：$n_r + b_r$　$b_r b_s + b_r \to |b_r + b_s$ (if s in mem)　$2 \cdot b_r \to 2$ (conjunctors)

indexed nested-loop
join cost (worst)：$b_r(t_T + t_S) + n_r \cdot c$

merge join (先 sort 再 join)　bt：$b_r + b_s$，sk：$\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil$

hash join



partitions → build + probe
of ~：$n_h = \min(\lceil \frac{b_s}{b_b} \rceil, M-1)$　s_i　r_i
$n_h \ge M$ → recursive partitioning
$M \ge \sqrt{b_s}$ not →
$2(\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil) + 2n_h$
$2(\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil)\lceil \log_{\lceil \frac{M}{b_b}-1 \rceil}(\frac{b_s}{M}) \rceil$

Sort (physically) → external merge sort
M = # of blocks in memory
high case: N-way merge
创建 run 时用 1 块，合并 run 时用 M-1 个块
cost
# of merge pass：$\lceil \log_{\lceil \frac{M}{b_b}-1 \rceil}(\frac{b_r}{M}) \rceil$
block transfer：$b_r(2 \cdot \#\text{ of mp} + 1)$
seek：$2\lceil \frac{b_r}{M} \rceil + \lceil \frac{b_r}{b_b} \rceil (2 \cdot \#\text{ of mp} - 1)$
pass 1 (out of order) runs (in order)　pass 2
b_b：# of blocks for I/O buffer
b_r：# of blocks for records

九. Evaluation Plan

cost-based join-order selection → dp → find best join order　$\frac{(2(n-1))!}{(n-1)!}$ orders)　$O(3^n)$

best plan：plan → cost，return best plan[S] (array)
S → S1 / S-S1 (recursively, disjoint)
cost of S = min(cost of S1 + cost of S-S1 + A)

heuristics：尽早 查询 & 投影
left-deep join order → $O(n2^n)$
cost budget → plan caching
equivalence rules：for 聚合、外连接、嵌套查询

物化视图 → 维护 → incremental / immediate / deferred
- join：$V^n = V^o \cup (i_R \bowtie S)$
- select：$V^n = V^o - (d_R \bowtie S)$；$V^n = V^o \cup (\sigma_\theta(i_R))$；$V^n = V^o - \sigma_\theta(d_R)$

十. Transactions

ACID
- atomicity：all or none
- consistency：commit 后的结果指受存在 (即使有 failure)
- isolation：多为间不会相互干扰
- durability

5 states
active：执行完最后一句 → partially committed → committed
active：正在执行 → failed → abort → restart / kill
failed：已回滚

concurrency ⇒ throughput↑, response time↓

schedule
- serial
- parallel → serializable (same result as serial one)

(same data, diff txn) conflicts
- no →
- RAW：unrepeatable read
- WAR：dirty read
- WAW：lost update

precedence graph：Ti → Tj (conflict)
cycles → topological order → serializability order
check

conflict serializable

recoverability
- recoverable：Tj read data written by Ti and Tj's commit is later than Ti's
- cascadeless：no multiple txn rollback owning to single txn

isolation levels	dirty read	unrepeatable read	lost updates	phantom
seriolizable	×	×	×	×
repeatable read	×	×	×	
read committed	×			
read uncommitted				

high ↓ low

十一. Concurrency Control

Lock
- S：read √
- X：read + write
- IS：为后代加 S 锁
- IX：为后代加 S 或 X 锁
- SIX：子树根节点加 S 锁，其他加 X 锁

compatibility function

已 \ 想	IS	IX	S	SIX	X
IS	√	√	√	√	×
IX	√	√	×	×	×
S	√	×	√	×	×
SIX	√	×	×	×	×
X	×	×	×	×	×

starve：一直等不到锁

multiple granularity

two-phase locking protocol (2PL)：对保护可串行，但不保证死锁和级联回滚的出现

lock point：# growing phase　shrinking phase

two variations
- strict ~：保留 X 锁直到 commit / abort (cascadeless)　cascadeless
- rigorous ~：保留所有锁直至 commit (更 easy，但 concurrency↓)

lock conversion ⇒ concurrency↑
upgrade $S \to X$ (growing)　downgrade $X \to S$ (shrinking)

lock manager uses lock table (hash)

tree (rooted) protocol：
- A：数据被访问的顺序 = partial order
- B C D E：仅当 parent locked 才能 lock node (data)，但直接点加锁

可恢复性 / 无级联 / 不让其他可从未提交的 txn 读取

deadlock
- prevention
  - wait-die：Ti 的 Tj 未锁，Ti 等 Tj 释放锁 / Ti 的 Tj 未锁，Ti 等 Tj 释放锁
  - wound-wait：一个仅当 Tj 未新等，否则打断 / 抢夺
  - timeout：超时等 → 滚
- detection & recovery
  - wait-for graph：Ti → Tj：Tj 等 Ti 放锁 → has cycles → deadlock
  - victim rollback：total / partial
  - starve (counter)

mgp (con.)

concurrency↑ → deadlock possible
other operations：insert 和 delete 与其他操作都冲突
因此并行前上 X 锁

lock escalation (rarely used)：同一 txn 多次相同 query 结果不一致
phantom phenomenon
index lock / 前址

granularity：放粗 / 变细

predicate lock

十二. Recovery

failure：txn / sys / disk

log-based recovery
special types：<Ti start>　<Ti commit>　<Ti abort>
log record <Ti, Xj, V1, V2>
txnid　dataid　old val　new val

disk ⇄ (input / output) memory　modification < imm / df

group commit：多个 txn 一起交 log record (for force)
cost↓

redo：用新值 (txn 开始且已提交/中止)
undo：用旧值，完成后写 <Ti abort> 记录 (txn 开始但未提交/中止) (rollback)

checkpoint
- 执行时禁止任何数据更新 (bad)
- 将前面的修改结果写到 disk 上
- log record：<checkpoint L>

fuzzy checkpoint：blocking time↓、recovery time↑

ATT

buffering：是否在提交前把更新数据写到磁盘中
- 是否允许未提交 txn 覆写 disk 内容
- 是否允许未提交 txn 覆写 disk 内容

WAL：先写 log，再写 data (到 disk)

manage policy
- no-force
- force

data (到 disk)
- no-steal
- steal 写入快 需 redo+undo

trivial → 无需 undo 或 redo，但不保证

<Ti, Oj, operation-begin>
⋮
<Ti, Oj, operation-end, U>
undo 的 exe

logical record (only for undo)：已经 redo 的 txn 不用补偿
① 直接跳过 ② 插入 <Ti, Oj, operation-abort> ③ 补偿操作

ARIES

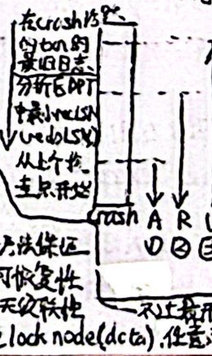
DPT：脏页表 (被改过的页)

list of LSN

flushedLSN	mem	last LSN in disk log
pageLSN	page	latest LSN in page
recLSN	DPT	oldest LSN in page from
lastLSN	ATT	latest LSN in txn
prevLSN	log record	LSN of previous log record

CLR

3 个阶段



① analysis：
1. redoLSN = min(recLSN in DPT, or LSN of ①)
2. create undo list
② redo：
③ undo

幂等 (idempotent)：多次执行某操作 ≡ 仅执行一次该操作