1 形式关系查询语言

关系代数：选择 σ，投影 Π，并 $\cup$，差 $-$，笛卡尔积 $\times$，更名 ρ。**推广的运算**：交 $r \cap s = r - (r - s)$，自然连接 $r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge \cdots \wedge r.A_n = s.A_n}(r \times s))$，除法 $r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$，赋值 $\leftarrow$。**扩展的关系代数**：泛化投影（投影的参数是函数或运算）、聚集函数（有 avg、min、max、sum 和 count，$_{G_1,\cdots,G_n} g_{F_1(A_1),\cdots,F_m(A_m)}$）、外连接。**谓词元组关系演算**：$\{t | P(t)\}$，由以下组成，1) 属性和常量，2) 比较运算符，3) 逻辑运算符，4) 量词。**域关系演算**：$\{\langle x_1, \cdots, x_n \rangle | P(x_1, \cdots, x_n)\}$。

2 数据库设计和 E-R 图

E-R 组成：实体集（实体是对象，矩形）、关系集（可以是多个实体之间，个数称为 degree，菱形）、属性（可以是关系的，这时候是虚线）。**属性分类**：简单和组合（缩进表示），单值和多值（花括号括住），派生（末尾一对括号）。**映射基数限制**：一对一，一对多，多对一，多对多（箭头指向实体集代表一，无箭头代表多，可以用 $a..b$ 来代表更复杂基数限制，其中 b 也可以是 $*$ 代表没有限制）。**超键**：可以决定其他属性的一组属性。**候选键**：最小的超键。**主键**：挑选出一个候选键（下划线属性）。**全参与与部分参与**：全参与（双线）是所有实体有至少一个关系，部分参与（单线）是存在实体没有关系。**冗余属性**：对于一些出现在两个实体集像外键的属性，在 ER 图需要移除。**弱实体集**：（双矩形，区分属性下划虚线，关系双菱形）所有属性不足以形成主键的实体，依赖于（被 own）强实体（identifying entity），强实体与弱实体的关系（identifying relationship）只能是一对一或一对多且弱实体是全参与。**角色**：对于实体集多次参与同一关系集，每次参与都有个角色（线上文字）。**E-R 图转数据库 schema**：强实体集转化为所有属性组成 schema；弱实体集还带上强实体集的主键；多对多关系转化包含两者主键；一对多或多对一如果多的一边是全参与则在多的一边添加一的一边的主键，如果不是全参与则使用 null 值；一对一两边都可以当做多的一边处理；弱实体集关系是冗余的；组合属性扁平化；多值属性则用单独表示包含主键和多值属性。**多元关系**：为了避免困惑，只有一个出箭头是允许的，可以转化为二元关系。**特化和泛化**：重合（箭头直接指向基础实体）、分离（箭头汇合后指向基础实体）；两种转换方法：1) 派生实体包含基础实体的主键，2) 派生实体直接包含基础实体；完整性约束：全/部分（基础实体是否必须是派生实体）。

3 关系数据库设计

第一范式：属性的域都是原子的（不可分割）。**函数依赖**：$\alpha \to \beta, \alpha \subseteq R \wedge \beta \subseteq R$，如果 $\forall t_1, t_2 \in r, t_1[\alpha] = t_2[\alpha] \to t_1[\beta] = t_2[\beta]$。**超键**：$K$ 是超键 $\Leftrightarrow K \to R$。**候选键**：K 是候选键 $\Leftrightarrow K \to R \wedge \neg\exists\alpha(\alpha \subseteq K \wedge \alpha \to R)$。**平凡的函数依赖**：$\beta \subseteq \alpha \Leftrightarrow \alpha \to \beta$ 是平凡的。**函数依赖集合的闭包**：令 F 是函数依赖的集合，所有 F 隐含的函数依赖集合是 F 的闭包，记作 F^+，一定有 $F \subseteq F^+$。**BCNF 范式**：$\forall \alpha \to \beta \in F^+, \alpha \to \beta$ 是平凡的 ($\beta \subseteq \alpha$) 或 α 是超键 ($\alpha \to R$)。**分解为 BCNF 范式**：对于违反 BCNF 范式的函数依赖 $\alpha \to \beta$，分解为 $\alpha \cup \beta$ 和 $R - (\beta - \alpha)$，分解有时候不保留依赖关系。**第三范式**：$\forall \alpha \to \beta \in F^+, \alpha \to \beta$ 是平凡的或 α 是超键或 $\forall A \in \beta - \alpha, A$ 被某个候选键包含。符合 BCNF 范式一定符合第三范式，第三范式保留依赖关系。**求解函数依赖闭包**：反复应用下面 3 条法则，即可求出闭包：1)（自反性）$\beta \subseteq \alpha \Rightarrow \alpha \to \beta$；2)（提升性）$\alpha \to \beta \Rightarrow \gamma\alpha \to \gamma\beta$；3)（传递性）$\alpha \to \beta \wedge \beta \to \gamma \Rightarrow \alpha \to \gamma$。**闭包额外的性质**：1)（联合）$\alpha \to \beta \wedge \alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$；2)（分解）$\alpha \to \beta\gamma \Rightarrow \alpha \to \beta \wedge \alpha \to \gamma$；3)（伪传递）$\alpha \to \beta \wedge \gamma\beta \to \delta \Rightarrow \alpha\gamma \to \delta$。**属性的闭包**：在函数依赖 F 下，能够被属性集 α 决定的属性集 α^+。**属性闭包的应用**：1) 测试超键 α，$\alpha^+ = R$；2) 测试函数依赖 $\alpha \to \beta$，$\beta \subseteq \alpha^+$；3) 计算函数依赖 F 的闭包，对于每个 $\gamma \subseteq R$，计算属性闭包 γ^+，然后对于每个 $S \subseteq \gamma^+$，输出依赖 $\gamma \to S$。**Canonical 覆盖**：最小的函数依赖集合。**无关属性**：对于函数依赖集 F 中的 $\alpha \to \beta$，1) A 是 α 中无关的属性，如果 F 逻辑上蕴含 $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$；2) A 是 β 中无关的属性，如果 $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ 逻辑上蕴含 F。**测试属性是否是无关的**：对于函数依赖集 F 中的 $\alpha \to \beta$，1) 测试 $A \in \alpha$，用 F 计算 $(\{\alpha\} - A)^+$，如果它包含 β，则 A 是多余的；2) 测试 $A \in \beta$，用 $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ 计算 α^+，如果包含 A，则 A 是多余的。**计算 Canonical 覆盖**：先使用联合规则合并函数依赖，在测试属性是否多余，循环往复。**无损分解**：将 R 分解为 R_1 和 R_2，如果 $R_1 \cap R_2 \to R_1$ 或 $R_1 \cap R_2 \to R_2$ 则为无损分解。**依赖保留**：F_i 为各个分解的函数依赖集，如果 $(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$，则分解为依赖保留的。**测试 BCNF 分解算法**：需要用到分解前的 F^+ 中相关的部分。**第三范式分解**：首先计算 Canonical 覆盖 F_c，对于 F_C 中每个 $\alpha \to \beta$ 的函数依赖，将 $\alpha\beta$ 添加进分解中，如果没有一个分解包含候选键，则随便添加一个候选键到分解里，最后将那些被包含在其他分解里的分解移除。

4 存储和文件结构

架构层数：两层（直接操纵数据）、三层（通过服务端操纵数据）。**存储分类**：volatile（停电丢失数据）、non-volatile。**存储层级**：Cache 缓存、内存、闪存（读快写慢擦除更慢，如 SSD）、磁盘、光盘、磁带，分为一级（缓存、内存）、二级（闪存、磁盘），三级（光盘、磁带）。**磁盘**：柱面（多盘面的某一磁道组成）、盘面、磁道、扇区；网络连接方式有 SAN（Storage Area Networks）和 NAS（Network Attached Storage）。**磁盘的性能测量**：访问时间（寻道时间 + 旋转延时），数据传输速率，MTTF（Mean time to failure）。**优化性能**：块（多个连续的扇区），电梯算法，文件组织（去碎片），non-volatile 缓存，Log disk，日志文件系统。**存储访问**：数据库文件被组织成固定长度的块，Buffer 缓存用于存储磁盘块的副本。**缓存置换策略**：LRU 策略（Least recently used）。**文件组织**：固定长度记录的删除，1) 将之后的记录往前移；2) 将最后的移到最前；3) 维护空闲列表，将它加入空闲列表。变长记录使用槽页结构，其头部包含了记录数目，空闲的末尾，每条记录的地址和大小。**文件中记录的组织**：堆，序列，哈希。**序列文件组织**：删除使用指针链；插入时如果有空闲则插入到空闲，否则插入到溢出块，最后更新指针链。

5 索引和哈希

搜索键：用于查找的键。**索引文件**：存放搜索键、指针二元组的文件，有顺序和哈希两种。**有序索引**：主索引搜索键的顺序决定了索引的顺序，又称聚集索引；反之称之为二级索引或非聚集索引；密集索引每一个搜索键都有索引；反之称之为稀疏索引。二级索引：索引指向桶，桶再指向记录。多级索引：如果主键索引不能放到内存，就将索引当做记录，对它创建稀疏索引。**删除记录的索引更新**：密集索引直接删除对应索引；稀疏索引将下一个搜索键作为索引，如果该搜索键已经有索引，则删除索引。**插入记录的索引更新**：密集索引直接插入索引；稀疏索引寻找对应位置插入。B^+ **树性质**：根节点到叶子节点的路径等长；非根非叶的节点有 $\lceil n/2 \rceil$ 到 n 个子节点；叶节点有 $\lceil (n-1)/2 \rceil$ 到 $n-1$ 个节点；如果根不是叶，则至少有 2 个子节点；如果根是叶，它有 0 到 $n-1$ 个值；K 个搜索键的高度不超过 $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$。$B^+$ **树节点结构**：每个节点有 $n-1$ 个有序的搜索键 K_i 和 n 个指针 P_i；对于非叶节点指针指向孩子；对于叶节点指针指向记录或桶，最后一个指针指向下一个叶节点，$P_i < K_i, P_{i+1} \geq K_i$。**重复搜索键**：$K_1 \leq K_2 \leq \cdots \leq K_{n-1}, P_i \leq K_i$。$B^+$ **树的插入**：如果查找键出现在了叶节点里，则添加入桶，否则将查找键和指针插入叶节点中，如果此时空间不够，则分裂节点；分裂叶节点时左边留 $\lceil n/2 \rceil$ 个节点，剩余的留在右边，再对父节点插入元组，如果父节点满再将分裂传递下去；分裂非叶节点时，左边留 $P_1, K_1 \cdots K_{\lceil n/2 \rceil - 1}, P_{\lceil n/2 \rceil}$，右边留 $P_{\lceil n/2 \rceil + 1}, K_{\lceil n/2 \rceil + 1} \cdots K_n, P_{n+1}$，再将 $(K_{\lceil n/2 \rceil}$, 新节点) 插入到父节点。B^+ **树的删除**：合并兄弟节点，并删除父节点到删除节点的搜索键和指针。**静态哈希**：哈希函数映射到桶，桶包含多条记录；如果溢出，存放在溢出桶里。**哈希索引**：采用哈希的索引，一定是二级索引。**动态哈希**：桶地址表规模为 2^i，初始 $i = 0$，每个桶对应于一个 i；插入时如果桶满则分裂，对于桶 j，如果 $i > i_j$ 则插入一个新桶，如果 $i = i_j$ 则重新计算地址表；删除时，如果桶空，则合并。**位图索引**：应用于取值很少的属性，是位的数组。

6 查询处理

查询时间开销：A1 线性搜索 b_r(块数目)t_T(传输时间) + t_S(寻道时间)；A2 B^+ 树主索引，判相等，搜索键 $(h_i$(索引高度) + 1) $\times$ $(t_T + t_S)$；A3 B^+ 树主索引，判相等，非搜索键 $h_i \times (t_T + t_S) + t_S + b$(搜索键对应的块数) $\times$ t_T；A4 B^+ 树二级索引，判相等，搜索键，同 A1；A4 B^+ 树二级索引，判相等，非搜索键 $(h_i + n$(记录数)) $\times$ $(t_T + t_S)$；A5 B^+ 树主索引，比较 B^+，同 A3；A6 B^+ 树二级索引，比较，同 A4 非搜索键；A7 利用 *1* 个索引合取选择；A8 使用组合索引合

取选择；A9 通过标识符的交实现合取选择；A10 通过标识符的并实现合取选择。**排序操作**：内存中可使用快排；否则使用外部排序（归并排序），令 M 是内存块的个数，b_b 是每次归并读取的块数，其磁盘块传输总数 $b_r(2\lceil\log_{M-1}(b_r/M)\rceil+1)$，寻道总数为 $2\lceil b_r/M\rceil+\lceil b_r/b_b\rceil(2\lceil\log_{\lfloor M/b_b\rfloor-1}(b_r/M)\rceil-1)$。**连接操作**：嵌套循环，块传输总数 $n_r\times b_s+b_r$，寻道总数 n_r+b_r；嵌套块循环，块传输总数 $b_r*b_s+b_r$，寻道总数 $2b_r$；索引嵌套循环，开销为 $b_r(t_T+t_S)+n_r\times c$(找到 r 中对应元素的开销)；合并连接，先对两个关系排序，再连接，块传输总数 b_r+b_s，寻道总数 $\lceil b_r/b_b\rceil+\lceil b_s/b_b\rceil$；哈希连接，先哈希，再对每一个哈希的块连接，如果不能全部加载入内存会有递归划分，不考虑递归划分，块传输总数 $3(b_r+b_s)+4n_h$(划分总数)，寻道总数 $2(\lceil b_r/b_b\rceil+\lceil b_s/b_b\rceil)$，考虑递归划分，块传输总数 $2(b_r+b_s)\lceil\log_{\lfloor M/b_b\rfloor-1}(b_s/M)\rceil+b_r+b_s$，寻道总数 $2(\lceil b_r/b_b\rceil+\lceil b_s/b_b\rceil)\lceil\log_{\lfloor M/b_b\rfloor-1}(b_s/M)\rceil$。**表达式求值**：物化和流水线。

7 查询优化

等价规则：σ 的级联及交换律，$\sigma_{\theta_1\wedge\theta_2}(E)=\sigma_{\theta_1}(\sigma_{\theta_2}(E))=\sigma_{\theta_2}(\sigma_{\theta_1}(E))$；$\Pi$ 的级联，$\Pi_{L_1}(\cdots(\Pi_{L_n}(E))\cdots)=\Pi_{L_1}(E)$；选择、笛卡尔积及 θ 连接结合，$\sigma_\theta(E_1\times E_2)=E_1\bowtie_\theta E_2$，$\sigma_{\theta_1}(E_1\bowtie_{\theta_2}E_2)=E_1\bowtie_{\theta_1\wedge\theta_2}E_2$；$\theta$ 连接的交换性，$E_1\bowtie_\theta E_2=E_2\bowtie_\theta E_1$；自然连接的结合律，$(E_1\bowtie E_2)\bowtie E_3=E_1\bowtie(E_2\bowtie E_3)$；$\theta$ 连接的结合律，如果 θ_2 只涉及 E_2 与 E_3 的属性，则 $(E_1\bowtie_{\theta_1}E_2)\bowtie_{\theta_2\wedge\theta_3}E_3=E_1\bowtie_{\theta_1\wedge\theta_3}(E_2\bowtie_{\theta_2}E_3)$；选择连接对 θ 连接的分配律，如果 θ_0 只涉及 E_1，则 $\sigma_{\theta_0}(E_1\bowtie_\theta E_2)=(\sigma_{\theta_0}(E_1))\bowtie_\theta E_2$，如果 θ_1 只涉及 E_1 且 θ_2 只涉及 E_2，则 $\sigma_{\theta_1\wedge\theta_2}(E_1\bowtie_\theta E_2)=(\sigma_{\theta_1}(E_1))\bowtie_\theta(\sigma_{\theta_2}(E_2))$；投影运算对 θ 连接的分配律，如果 L_1、L_2 是 E_1、E_2 的属性，假设 θ 只涉及 $L_1\cup L_2$ 中的属性，则 $\Pi_{L_1\cup L_2}(E_1\bowtie_\theta E_2)=(\Pi_{L_1}(E_1))\bowtie_\theta(\Pi_{L_2}(E_2))$，假设 θ 还涉及了 L_3 中的属性，则 $\Pi_{L_1\cup L_2}(E_1\bowtie_\theta E_2)=\Pi_{L_1\cup L_2}((\Pi_{L_1\cup L_3}(E_1))\bowtie_\theta(\Pi_{L_2\cup L_3}(E_2)))$；集合的交和并有交换律；集合的交和并有结合律；选择对并、交、差的分配律；投影对并的分配律。**转换的例子**：先投影再连接，先连接小的。**开销估计的统计信息**：l_r 元组的大下，f_r 一个块中的元组个数，$V(A,r)=n_{\Pi_A,r}$，一定有 $b_r=\lceil n_r/f_r\rceil$。**选择大小估计**：$\sigma_{A=v}(r)$ 的大小约为 $n_r/V(A,r)$；$\sigma_{A\le V}(r)$ 的大小约为 $n_r\cdot\frac{v-\min(A,r)}{\max(A,r)-\min(A,r)}$；合取，$\sigma_{\theta_1\wedge\cdots\wedge\theta_n}(r)$ 的大小约为 $n_r\cdot\frac{S_1\cdot S_2\cdot\cdots\cdot S_n}{n_r^n}$，其中 S_i 是对 $\sigma_{\theta_i}(r)$ 大小的估计；析取，$\sigma_{\theta_1\vee\cdots\vee\theta_n}(r)$ 的大小约为 $n_r\cdot\left(1-\left(1-\frac{S_i}{n_r}\right)\cdots\left(1-\frac{S_n}{n_r}\right)\right)$；取反，$\sigma_{\neg\theta}(r)$ 的大小约为 $n_r-\text{size}(\sigma_\theta(r))$。**连接大小估计**：$R\cap S=\emptyset$，用笛卡尔积估计；$R\cap S=\{A\}$ 且 A 是 R 的键，不会超过 s 的个数；$R\cap S=\{A\}$ 且 A 不是 R 的键，$\min\left\{\frac{n_rn_s}{V(A,s)},\frac{n_rn_s}{V(A,r)}\right\}$。**其他操作大小估计**：$\Pi_A(r)$ 大小约为 $V(A,r)$；${}_Ag_F$ 大小约为 $V(A,r)$；$r\cup s$ 大小约为 n_r+n_s；$r\cap s$ 大小约为 $\min\{n_r,n_s\}$；$r-s$ 大小约为 n_r；$r\bowtie s$ 大小约为 $\text{size}(r\bowtie s)+n_r$；$r\rtimes s$ 大小约为 $\text{size}(r\bowtie s)+n_r+n_s$。$V(A,\sigma_\theta(r))$ **的估计**：若 θ 取特定值，估计为 1；若 θ 取给定值，估计为给定值个数；若 $\theta=A\ op\ v$，估计为 $V(A,r)\times s$，s 是选中概率；其他情况，估计为 $\min\{V(A,r),n_{\sigma_\theta(r)}\}$。$V(A,r\bowtie s)$ **的估计**：若 A 属性全来自 r，则估计为 $\min\{V(A,r),n_{r\bowtie s}\}$；若 A 包括了来自 r 的属性 $A1$ 和来自 s 的属性 $A2$，估计为 $\min\{V(A1,r)\times V(A2-A1,s),V(A1-A2,r)\times V(A_2,s),n_{r\bowtie s}\}$。

8 事务

事务的要求：原子性、隔离性、持久性、一致性（ACID）。**稳定性存储器**：永远不会丢失数据。**事务的状态**：活动的、部分提交的、失败的、终止的、提交的。**可串行化**：等价于串行调度的调度，有冲突可串行化和视图可串行化。**冲突**：如果两个指令访问了同一数据且有一个指令写了数据，则它们是冲突的。**冲突等价**：如果 S 通过交换非冲突指令得到 S'，则 S 和 S' 冲突等价。**冲突可串行化**：与串行调度冲突等价的调度。**优先图**：画一条 T_i 到 T_j 的边，如果两个事务冲突且 T_i 先访问数据，优先图无环则可串行序列化。**视图可串行化**：满足以下 3 条称为 S' 与 S 视图等价，1) S 中某事务读取初始值，S' 中也是如此；2) S 中某事务读取的值是另一事务的结果，S' 中也是如此；3) S 中某事务最后写，S' 中也是如此；冲突可序列化一定视图可序列化；每个非冲突可序列化的视图可序列化存在盲写。**可恢复调度**：T_j 读取了 T_i 写入的数据，T_i 必须出现在 T_j 的提交之前。**级联回滚**：一个事务的失败会造成一系列未提交事务的失败。**无级联调度**：T_j 读取了 T_i 写入的数据，T_i 的提交出现在 T_j 的读之前。**隔离性级别**：可串行化、可重复读、已提交读、未提交读。

9 并发控制

锁的类型：排他锁（可读可写，lock-X 获得）、共享锁（只读，lock-S 获得）；共享锁之间可以相容，别的情况都不可以；死锁可以通过回退事务解决。**两阶段加锁协议**：分为两阶段；先是增长阶段，事务只能获取锁，再是缩减阶段，事务只能释放锁；保证冲突串行化，但不保证不发生死锁，级联回滚可能发生；严格两阶段加锁，事务提交后方可释放排他锁，可避免级联回滚；强两阶段加锁，提交后方可释放资源。**锁转换**：第一阶段可将共享锁升级为排他锁，第二阶段可将排他锁降级为共享锁。**锁表**：哈希索引数据项的列表，元素为锁，保证了无饿死现象。**树形协议**：基于图的协议，数据项 $D=\{d_1,d_2,\cdots,d_h\}$，偏序关系 $d_i\to d_j$，访问 d_j 之前必须访问 d_i，树形协议是一种简单的图协议；只有排他锁，首次加锁可以是任何数据，接下来的加锁必须是已加锁的子节点，可以随时解锁，解锁完了不能再加锁；保证冲突可串行化和无死锁，不保证可恢复和无级联回滚。**死锁预防**：一次全部加锁；规定加锁次序，如树形协议；*wait-die* 机制（非抢占式），老的事务可以等待新的事务，当新的事务等待老的事务的时候回滚；*wound-wait* 机制（抢占式），新的事务可以等待老的事务，当老的事务等待新的事务的时候回滚；超时机制。**死锁检测**：使用等待图，顶点时事务，$T_i\to T_j$ 表示 T_i 在等待 T_j 释放所需要数据项，有环则有死锁。**多粒度**：细粒度、粗粒度；4 层，数据库、区域、文件、记录。**多粒度的意向锁**：共享意向（IS，底层只能加共享锁）、排他意向（IX，底层可加共享或排他锁），共享排他意向锁（SIX，底层加了共享锁，更底层加了排他锁）；IS-IS、IS-IX、IX-IX、IS-S、IS-SIX、S-S 相容。**基于时间戳的协议**：为每个事务记录了时间戳 $\text{TS}(T_i)$，为每个数据记录了两个时间戳，W-timestamp(Q) 最大执行 write(Q) 的时间、R-timestamp(Q) 最大执行 read(Q) 的时间；T_i 发出 read(Q)，若 $\text{TS}(T_i)<\text{W-timestamp}(Q)$ 则拒绝回滚，否则成功并更新时间戳；T_i 发出 write(Q)，若 $\text{TS}(T_i)<\text{R-timestamp}(Q)\vee\text{TS}(T_i)<\text{W-timestamp}(Q)$ 则拒绝并回滚，否则成功并更新时间戳；无死锁，可能出现及联合回滚，可能不可恢复。**Thomas 写规则**：当 $\text{TS}(T_i)<\text{W-timestamp}(Q)$，忽略写操作。**基于有效性检查的协议**：事务分为 3 阶段，读和执行、验证、写，又称为乐观并发控制，3 阶段对应 3 个时间戳 $\text{Start}(T_i)$、$\text{Validation}(T_i)$、$\text{Finish}(T_i)$，其中 $\text{TS}(T_i)=\text{Validation}(T_i)$。$T_i$ **的有效性测试**：对于所有的 $\text{TS}(T_k)<\text{TS}(T_i)$，满足下面 2 条条件之一，1) $\text{Finish}(T_k)<\text{Start}(T_i)$，2) T_k 写的数据与 T_i 读的数据不相交且 $\text{Start}(T_i)<\text{Finish}(T_k)<\text{Validation}(T_i)$，则 T_i 通过并提交，否则终止。**多版本机制**：多版本时间戳排序和多版本两阶段加锁。**多版本时间戳排序**：存储一系列版本，每个版本包含内容和读写时间戳；T_i 写时读写时间戳初始化为 $\text{TS}(T_i)$，T_i 读时如果 $\text{TS}(T_i)$ 大于读时间戳则更新；令 Q_k 是小于等于 $\text{TS}(T_i)$ 的最大写时间戳的版本；read(Q) 时返回 Q_k 的内容；write(Q) 时，若 $\text{TS}(T_i)<\text{R-timestamp}(\text{Q}_\text{k})$ 则回滚，若 $\text{TS}(T_i)=\text{W-timestamp}(\text{Q}_\text{k})$ 则覆盖 Q_k 的内容，否则创建新的版本；不保证可恢复性和无级联性。**多版本两阶段加锁**：区分只读事务和更新事务；数据项有时间戳 ts-counter；更新事务执行强两阶段加锁协议；只读事务开始时读取当前 ts-counter，读取小于 $\text{TS}(T_i)$ 的最大时间戳的版本的内容；更新数据项时，创建新版本，时间戳置为 ∞，提交时时间戳置为 ts-counter + 1，再对 ts-counter 加 1；可恢复的和无级联的。

10 恢复系统

错误分类：事务错误（包含逻辑错误、系统错误）、系统崩溃（Fail-stop 假设非易失存储的内容不会改变）、磁盘错误。**基于日志的恢复**：当事务 T_i 启动的时候，插入日志 $\langle T_i\ \text{start}\rangle$；当 T_i 执行 write(X)，插入日志 $\langle T_i,X,V_1,V_2\rangle$，$V_1$ 是旧值，V_2 是新值；当 T_i 执行完毕后，插入日志 $\langle T_i\ \text{commit}\rangle$。**数据库修改**：延迟修改，知道提交都没修改数据库；立即修改，修改在事务活跃时发生。**撤销和重做**：撤销 $\langle T_i,X,V_1,V_2\rangle$ 时将 V_1 写入 X，重做 $\langle T_i,X,V_1,V_2\rangle$ 时将 V_2 写入 X；undo(T_i) 时，回退所有操作，并写入特殊日志 $\langle T_i,X,V_1\rangle$，最后插入 $\langle T_i\ \text{abort}\rangle$；redo($T_i$) 时没有日志输出。**检查点**：隔段时间执行，写入所有日志到稳定存储，并插入日志 $\langle\text{checkpoint }L\rangle$，$L$ 是所有活跃的事务。**系统崩溃后的恢复**：重做阶段，1) 将 undo-list 设为 $\langle\text{checkpoint }L\rangle$ 中的 L，2) 遇到 $\langle T_i,X,V_1,V_2\rangle$ 或 $\langle T_i,X,V_2\rangle$，将 V_2 赋给 X，3) 发现 $\langle T_i\ \text{start}\rangle$，将 T_i 加入到 undo-list，4) 发现 $\langle T_i\ \text{abort}\rangle$ 或 $\langle T_i\ \text{commit}\rangle$，将 T_i 从 undo-list 中移除；撤销阶段，反向扫描日志，撤销 undo-list 中的操作。