数据库系统

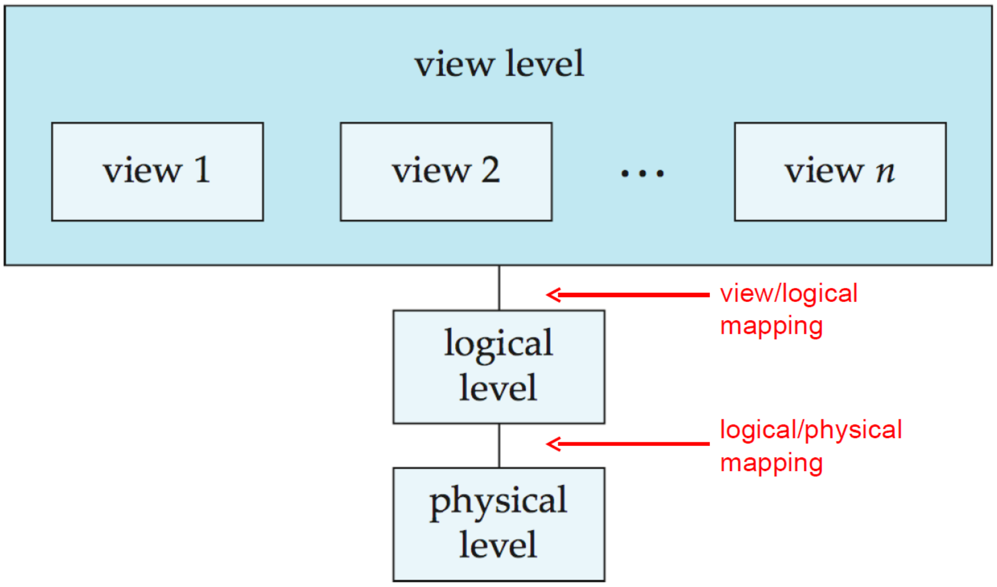
chapter1 Introduction

英文	中文	英文	中文
Data redundancy	数据冗余	inconsistency	不一致
data isolation	数据孤岛	integrity	完整性
atomicity	原子性	concurrent access anomalies	并发访问异常
authentication	认证	privilege	权限
audit	审计	schema	模式
instance	实例	explicitly	显式
Attribute	属性	tuple	元组
metadata	元数据	transaction	事务

characteristics of database:

1. 数据持久性data persistence
2. 数据访问便利性convenience in accessing data
3. 数据完整性data integrity
4. 多用户并发控制concurrency control for multiple user
5. 故障恢复failure recovery
6. 安全控制security control

view of data:三级抽象层次，具有适应变化的能力



数据独立性：

- 物理数据独立性physical data independence：适应物理存储的变化，数据不依赖物理存储
- 逻辑数据独立性logical data independence

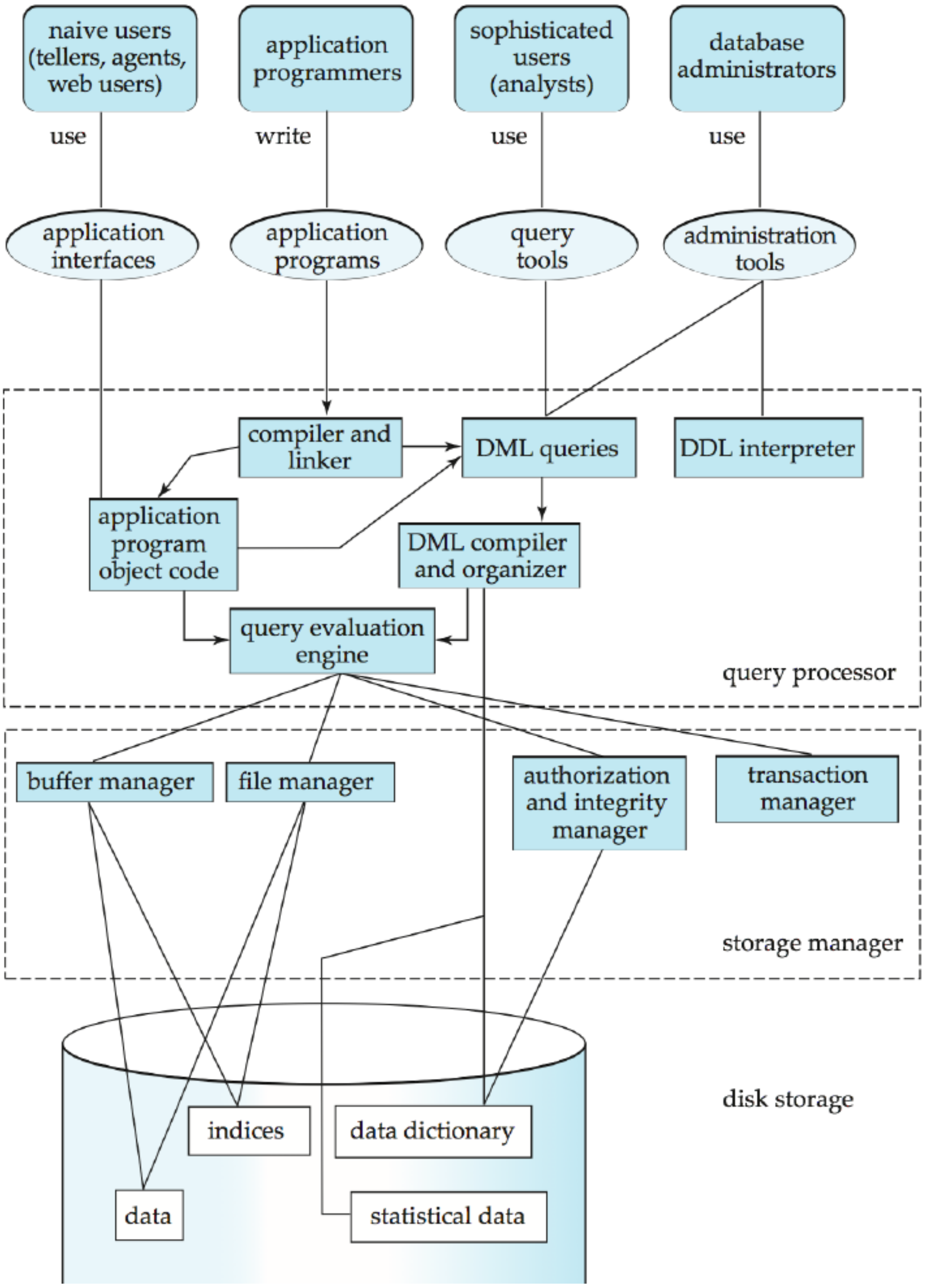
数据库语言：

1. Data Definition Language(DDL)：数据定义语言
    - 完整性约束
        - 主键
        - reference参照完整性
    - 权限
    - 产生metadata元数据储存在数据字典中（数据结构的数据）

```
create table instructor(
    ID       char(5),
    name     varchar(20),
    salary   numeric(8,2))
```

2. Data Manipulation Language(DML)：数据操作语言
    - procedural过程式：如C语言
    - declarative陈述式、非过程式：如SQL语句
3. API/embedded SQL：在应用程序中调用

数据库引擎



Database Administrator(DBA):

1. schema definition
2. storage structure and access method definition
3. schema and physical organization modification
4. granting user authority to access the database
5. specifying integrity constraints
6. acting as liaison with users
7. monitoring performance and responding to changes in requirements

---

chapter2 Introduction to Relational Model

chapter6.1 Relational Algebra

英文	中文	英文	中文
domain	域	atomic	原子的
compatible	相容的	multiset	多重集

Key

1. **superkey超键**：起到标识作用，独一无二的
2. **candidate key候选键**：最小的超键
3. **primary key主键**：其中一个candidate key
4. **foreign key外键**

Formal Algebra

- selectσ:

$$\sigma_{条件}(关系r)$$

- project$\prod$:

$$\prod_{属性A_1,A_2,\cdots,A_k}(关系r)$$

- union$\cup$:

$$r \cup s$$

  两个关系拥有相同的元组，且相容

- difference$-$：

$$r - s$$

- Cartesian product$\times$：

$$r \times s$$

- rename ρ：对E关系的属性进行重命名

$$\rho_X(E)$$
$$\rho_{x(A_1,A_2,\cdots,A_n)}(E)$$

Additional operations

- Notation$\cap$

$$r \cap s = \{t|t \in r \text{ and } t \in s\}$$

- natural-join$\bowtie$

$$r \bowtie s = \prod_{r.A,r.B,r.C,s.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

- outer-join$=\bowtie$
  - left outer join $=\bowtie$：保留左边的表格

$$(r \bowtie s) \cup (r - \prod_R (r \bowtie s) \times \{(null, \cdots, null)\})$$

  - right outer join $\bowtie=$：保留右边的表格

$$(r \bowtie s) \cup (\{(null, \cdots, null)\} \times (s - \prod_R (r \bowtie s))$$

  - full outer join =⋈=：两边表格都保留

- assignment←赋值操作
- division÷除法操作

$$tmp1 \leftarrow \prod_{R-S}(r)$$

$$tmp2 \leftarrow \prod_{R-S}((temp1 \times s) - \prod_{R-S,S}(r))$$

$$result = temp1 - temp2$$

Extended Algrbra

- aggregation聚合函数

  $avg、min、max、sum、count$

$$_{分组标准 G_1,G_2,\cdots,G_n}\mathcal{G}_{聚合函数 F_1(A_1),\cdots,F_n(A_n)}(E)$$

---

chapter3 Introduction to SQL

SQL全称：Structured Query Language

multiset:可以存在相同元组

DDL(data-definition language)

- the schema for each relation
- the domain of values associated with each attribute
- integrity constraint
  - 索引
  - 安全与授权
  - 物理储存结构

```sql
insert into instructor values(10211,null,'Biology',66000)
delete from instructor where ID=10211
update instructor set salary=salary*1.1
```

- 完整性约束

  ```sql
  create table instructor(
      ID          char(5),
      name        varchar(20)not null,
      dept_name   varchar(20),
      salary      numeric(8,2),
      primary key(ID),
      foreign key(dept_name)references department)
  ```

- 外键

```
foreign key(dept_name)references department
    on delete cascade|setnull|restrict|setdefault
    on update cascade|setnull|restrict|setdefault
```

  - cascade：被引用的删除，则引用处也删除
  - setnull：被引用的删除，则引用处置null
  - restrict：被引用的删除是被限制的，不允许进行删除

基本操作语言

data-manipulation language(DML)

- distinct去除重复

```
select distinct dept_name from instructors
```

- rename重命名

```
select ID,name salary/12 as monthly_salary from instructors
```

- 字符串通配符
  - %：like "%dar%"代替一个字符子串
  - _：代替一个字符
- 升序、降序排列

```
select distinct name from instructor order by name desc ##降序
select distinct name from instructor order by name asc  ##升序
```

- 集合操作(去重)
  - union==or
  - intersect==and
  - except==but not in
- 集合操作（不去重）：即加上all，如union all
- where筛选行，having筛选分组

```
select dept_name, count(*) as cnt
from instructor
where salary>=1000
group by dept_name
having count(*)>10
order by cnt
```

- 嵌套语句
  - set membership

```sql
select distinct course_id
from section
where smester='Fall' and
      course_id (not) in(select course_id
                        from section
                        where semester='Spring')
```

- set comparison

```sql
select name
from instructor
where salary > some / all (select salary
                      from instructor
                      where dept_name='Biology')
```

- correlation variables

```sql
select course_id
from section as S
where semester='Fall' and
    (not) exists(select*
                from section as T
                where semester='Spring')
```

- 是否有重复的元组

```sql
select dept_name
from department
where unique(
            select name
            from student
            where student.dept_name=department.dept_name)
```

- case语句

```sql
update instructor
set salary=case
            when salary<=100000 then salary*1.05
            else salary*1.03
            end
```

chapter4 Intermediate SQL

- 定义数据类型

```sql
create type Dollors as numeric(12,2)final
```

- large-object type

        - blob:binary large object
        - clob:character large object
    - 完整性约束条件
    - assertion断言：判断一张表整体的特征
    - views让不同的用户可以看到不同的数据

      ```
      create view v as <query expression>
      ```

        - update view:对于view插入、修改是可行的，即通过view向真正的表插入修改
        - materialized view:物质化的临时表
    - transaction事务：一整件事只有全做和全不做
        - begin：implicity
        - end：commit work、rollback work

          ```
          set autocommit=0#关闭自动提交功能
          ```

        - ACDI性质
            - atomicity原子性
            - consistency一致性
            - isolation隔离性
            - durability持久性
    - authorization授权

      reference引用也需要授权

      ```
      grant select on instructor to User1,User2,User3/public
      revoke select on instructor from User1,User2,User3/public
      ```

      权限是否能够传递，取决于语句

      ```
      grant select on department to Amit with grant option#给予授权的权限
      revoke select on department from Amit cascade#回收Amit的权限及Amit授权的权限
      revoke select on department from Amit restrict#如果Amit有对外授权未收回，则无法回收Amit的权限
      ```

    - role

      ```
      create role teaching_assistant
      grant teaching_assistant to instructor
      ```

    - Trigger触发器
        - ECA规则：Event(insert,delete,update),Condition,Action

        -
          ```
          create trigger account_trigger after update pn account(balance)
          referencing new row as nrow
          referencing old row as orow
          for each row#每次触发会对表中每一行进行检查
              when nrow.balance-orow.balance>=200000 or
                  orow.balance-nrow.balance>=50000
              begin
                  insert into account_log values(..,..,..)
              end
          ```

        - 也可以维护数据库信息完整性

          ```
          create trigger score_check after update on register(score)
          referencing new table as ntable
          for each statement#对一个语句触发一次
              when some(select average(score)
                       from ntable
                       group by cno)<60
             begin
              rollback
          end
          ```

---

chapter5 Advanced SQL

- API(Application Program Interface)与Embedded SQL
- ODBC(Open Database Connectivity):C,C++,C#
    - 通过函数调用
- JDBC(Java Database connectivity):Java

  ```
  connection conn=DriverManager.getConnection(
      "jdbc:oracle:thin:@db.yale.edu:2000:univdb",userid,password);
  statement stmt=conn.createStatement();
  try{
      stmt.executeUpdata("insert into instructor values(...,...,...)");
  }catch(SQLException sqle){
      System.out.printIn("Error");
  }
  stmt.close();
  conn.close();
  ```

  ```
  DatabaseMataData dbmd=conn.getMetaData();
  ```

- embedded SQL in C
- SQLJ:embedded SQL in Java
- JPA(Java Persistence API)
- host language:宿主语言

  host value:既出现在SQL又出现在宿主语言

chapter7 Entity-Relationship Model

Entity Relationship Model

- **entity**:an object that exists and is distinguishable
- **relationship**:association among several entities
- 联系的度
  - binary relationship二元联系
  - ternary relationship三元联系
- 属性的类型
  - simple and composite
  - single-valued and multivalued
  - Derived:可以由其它属性派生出来的属性
- mapping cardinality constraints
  - one to one:$A \leftarrow r \rightarrow B$
  - one to many:$A \leftarrow r - B$
  - many to many:$A - r - B$
  - total participation全部参与这个关系$A - r = B$
  - partial participation
- PK for relationship Set: **A.PK+B.PK**
- weak Entity Set弱实体集：没有完整的PK

  依赖于一个标识性实体集identifying entity set
- 特征
  - Specialization:父类->子类
    - Top-down design process
    - Attribute inheritance
  - Generalization：子类->父类
    - bottom-up process

chapter8 Relational Database Design

- 信息重复Information repetition

  插入异常Insertion anomalies

  更新困难Update difficulty
- Lossless-join⇔两部分的公共属性要么能决定r1，要么能决定r2
- First Normal Form：没有强限制

函数依赖

- augmentation增补率

$$\text{if } \alpha \rightarrow \beta, \text{then } \gamma\alpha \rightarrow \gamma\beta$$

- reflexivity自反率

$$\text{if } \beta \subset \alpha \text{,then } \alpha \to \beta$$

- transitivity传递率

$$\text{if } \alpha \to \beta \text{ and } \beta \to \gamma \text{ ,then } \alpha \to \gamma$$

- 属性的闭包：属性A能决定的属性集合

> $R(A, B, C, D)$
> $F = A \to B, B \to C, B \to D$
>
> 则：
>
> $$A^+ = ABCD$$
> $$B^+ = BCD$$
> $$C^+ = C$$

- 正则覆盖Canonical Cover：不存在冗余，既不包含多余属性也不包含多余关系，而且箭头左边不相同

BC范式BCNF

- 要么左边都包含key，要么$\alpha \to \beta$是平凡的，不一定能保证依赖保持

函数依赖保持Dependency Preservation：关系分解之后的集合，各个集合的关系并集与推导和原关系集合等价

Third Normal Form（3NF）：为了依赖保持

- 要么是BCNF要求的，要么在右边但不在左边的每个属性都被包含在某ckey中

Fourth Normal Form（4NF）

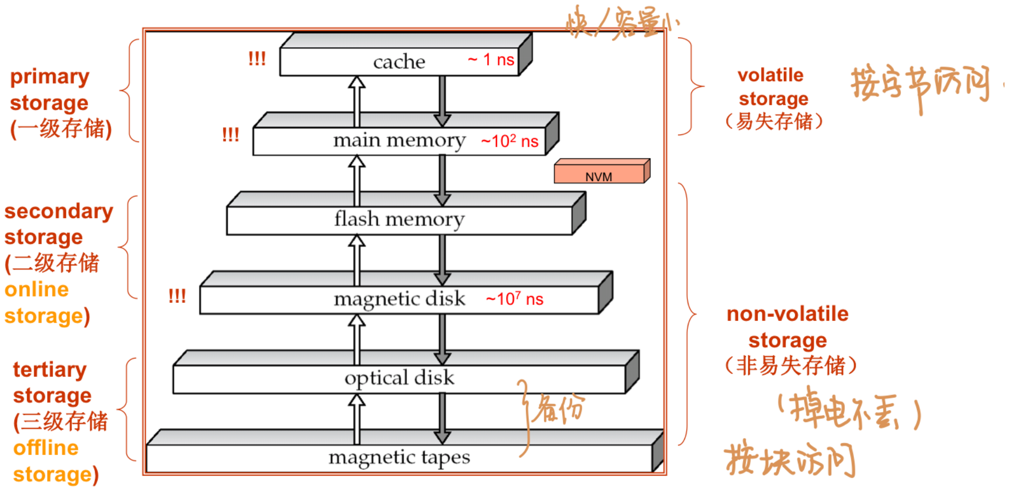
- 多值依赖：$\alpha \to\to \beta$

  函数依赖是一种特殊的多值依赖

- 要么$\alpha \to\to \beta$是平凡的，或者已经退化为一种函数依赖

---

chapter10 Storage and File Structure


快/容量小
primary storage (一级存储)
secondary storage (二级存储 online storage)
tertiary storage (三级存储 offline storage)
!!!
cache
~ 1 ns
!!!
main memory
~10² ns
NVM
flash memory
!!!
magnetic disk
~10⁷ ns
optical disk
备份
magnetic tapes
volatile storage（易失存储）
按字节访问
non-volatile storage（非易失存储）
(掉电不丢)
按块访问

- speed,cost,reliability

Magnetic Disk磁盘

- access time组成

  1. seek time：毫秒级
  2. rotational latency：旋转延迟，毫秒级

- Data-transfer rate数据传输率：MB/sec
- IOPS：每秒钟磁盘可以支持读的块数量（进行IO操作的次数）
- MTTF平均故障时间：3-5年
- 按块来存放，每块可能为4kbyte

磁盘优化

- 非易失性写缓存Nonvolatile write buffers：battery backed up RAM, flash memory
- 日志磁盘Log disk

SSD：由很多的闪存组成

- typical 4KB read: 10000 IOPS
- typical 4KB write:40000 IOPS

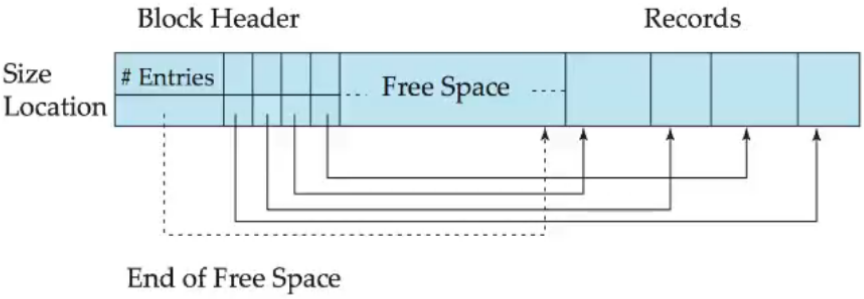
定长记录的存放

- 记录的插入：第i条记录（每条记录长度为n）的位置：$n*(i-1)$
- 记录的删除

变长记录的存放

- slotted page分槽页：

  1. number ofrecord entries
  2. end of free space in the block
  3. location and size of each record



LRU example:least recently used

1. 最近访问的放在最外面
2. 每次内存满了的时候，把最下面的记录写入磁盘

---

chapter11 Indexing and Hashing

- Dense index稠密索引：一个索引对应一个值
- sparse index稀疏索引：有些值没有索引

B+Tree Index

- 每一个内节点都有$\lceil n/2 \rceil$到n个孩子

- 根节点如果不是叶子节点，至少有2个孩子
- 叶子的孩子数：$\lceil n/2 \rceil$到$n-1$个值
- 如果有K个索引值，树的高度不会大于$\lceil \log_{\lceil n/2 \rceil}(K/2) \rceil + 1$

> ```
> person(pid char(18),name char(8),age smallint,address char(40),primary key(pid))
> Block size:4K
> 1000000 persons
> ```
>
> Records perblock=4K/(18+8+2+40)=60
>
> num of blocks to store 1M persons: 1000000/60=16667
>
> B+ tree n(fan-out): (4K-4)/(18+4)+1=187
>
> 叶子节点大小：$\lceil (n-1)/2 \rceil \rightarrow n-1$
>
> 非叶子节点大小：$\lceil n/2 \rceil \rightarrow n$
>
> key value个数：$2*94*94*94 \rightarrow 187*187*187*186$

LSM-Tree:按照序列形式进行追加，短期内不进行合并update

---

chapter12 Query Processing

查询代价的衡量

1. number of seeks
2. number of blocks read
3. number of blocks written

教材使用number of block transfers from disk and the number of seeks作为衡量指标

$$b\text{个}blocktransfer\text{和}S\text{次}seek\text{的时间}$$
$$b*t_T+S*t_S$$

File scan：linear search

$$\text{worse case} = b_r * t_T + t_s$$
$$\text{average case} = (b_r/2)t_T + t_s$$

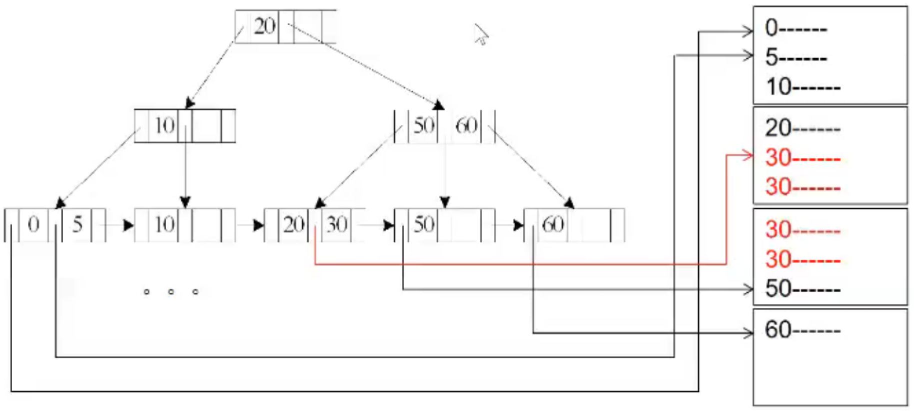
Index scan：primary B+-tree index,equality on key

$$cost = (h_i+1)*(t_T+t_S)$$

Index scan：primary B+-tree index,equality on nonkey

$$cost = h_i*(t_T+t_S)+t_S+t_T*b$$

b=包含了相关记录的内存块的个数（下图中为2）



Index scan： secondary B+-tree index,equality on key

$$cost = (h_i + 1) * (t_T + t_S)$$

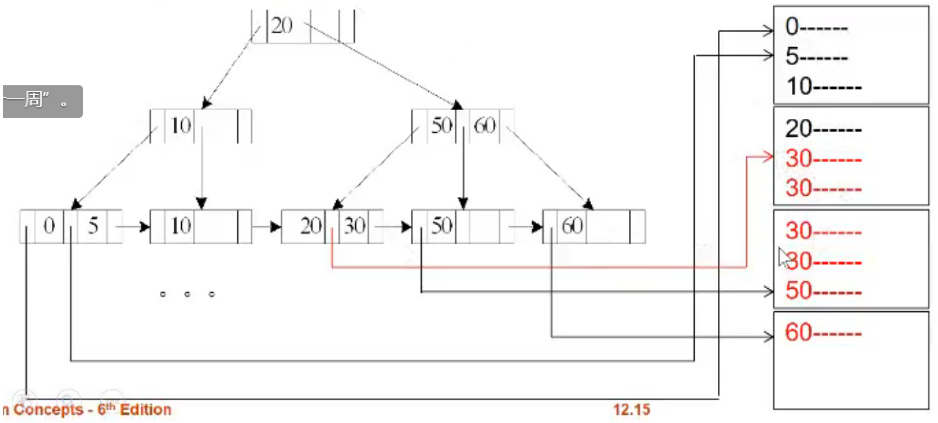
Index scan： secondary B+-tree index,equality on nonkey

$$cost = (h_i + m + n) * (t_T + t_S)$$

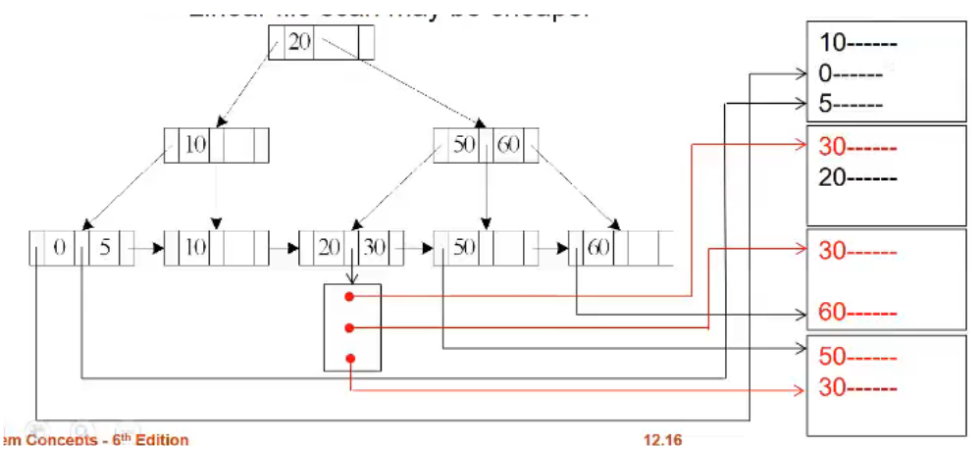
each of n matching records may be on a different block

n pointers may be stored in m blocks

Index scan： primary B+-tree,comparison



Index scan： secondary B+-tree,comparison



External soriting

假设内存块数为M

1. create sorted run
2. merge the runs

- normal cost,每次读一块

  total number of runs:$\lceil br/M \rceil$

  total number of merge passes required:$\lceil \log_{M-1}(br/M) \rceil$

  total number of transfer:$2br\lceil \log_{M-1}(br/M) \rceil + br$

  total number of seek:$2\lceil br/M \rceil + br(2\lceil \log_{M-1}(br/M) \rceil - 1)$

- advanced cost,每次可读bb块

  total number of transfer:$2br\lceil \log_{\lfloor M/bb \rfloor -1}(br/M) \rceil + br$

  total number of seek:$2\lceil br/M \rceil + (br/bb)(2\lceil \log_{\lfloor M/bb \rfloor -1}(br/M) \rceil - 1)$

Nested-Loop Join

$r \bowtie s$: r is outer relation, s is inner relation

- **worst case**:

$$\text{block transfer: } br * bs + br \\ \text{seeks: } 2 * br$$

- **best case**:

$$\text{block transfer: } bs + br \\ \text{seeks: } 2$$

有M块磁盘时候

- $$\text{block transfer: } \lceil br/(M-2) \rceil * bs + br \\ \text{seeks: } 2\lceil br/(M-2) \rceil$$

indexed nested-loop join

$$cost = br * (t_T + t_S) + n_r * c$$

merge join

bb:每次可以进去几块

$$\text{block transfer: } br + bs \\ \text{seeks: } \lceil br/bb \rceil + \lceil bs/bb \rceil \\ + \text{the cost of sorting if relations are unsorted}$$

hash join

nh:哈希函数生成的数的个数

$$\text{block transfer: } 3(br + bs) + 4n_h \\ \text{seeks: } 2(\lceil br/bb \rceil + \lceil bs/bb \rceil) + 2n_h$$

---

chapter13 Query Optimization

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. $\prod_{L1}(\prod_{L2}(\cdots(\prod_{Ln}(E))\cdots)) = \prod_{L1}(E)$
4. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
6. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

   $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
7. $\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\theta_0\text{只是E1上的属性}) = (\sigma_{\theta_0}(E1)) \bowtie_\theta E_2$

   $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$
8. $\prod_{L_1 \vee L_2}(E_1 \bowtie_\theta E_2) = (\prod_{L_1}(E_1)) \bowtie_\theta (\prod_{L_2}(E_2))$

performing the projection as early as possible

statistical information for cost estimation

n_r：number of tuples in a relation r

b_r：number of blocks containing tuples of r($b_r = \lceil \frac{n_r}{f_r} \rceil$)

I_r：size of a tuple of r

f_r：blocking factor of r——一个block中可以放多少个r元组

$V(A,r)$：number of distinct values that appear in r for attribute A(size of $\prod_A(r)$)

- $\sigma_{A=v}(r)$：$n_r / V(A,r)$
- $\sigma_{A \leq V}(r)$：$n_r * \frac{v - min(A,r)}{max(A,r) - min(A,r)}$
- **选择率selectivity**

  $\sigma_{\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n}(r)$：$n_r * \frac{S_1 * S_2 * \cdots * S_n}{n_r^n}$

  $\sigma_{\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n}(r)$：$n_r * (1 - (1 - \frac{S_1}{n_r}) * (1 - \frac{S_2}{n_r}) * \cdots * (1 - \frac{S_n}{n_r}))$

  $\sigma_{\neg\theta}(r)$：$n_r - size(\sigma_\theta(r))$
- **join**
  1. 如果没有重复属性，$sizeof(r \bowtie s) = r \times s$
  2. 如果R和S的公共属性是R的key，$sizeof(r \bowtie s) \leq r$
  3. 如果R和S的公共属性是S中依赖于R的外键，$sizeof(r \bowtie s) = s$
  4. 如果R和S的公共属性A不是key，$sizeof(r \bowtie s) = \min\{\frac{n_r * n_s}{V(A,s)}, \frac{n_r * n_s}{V(A,r)}\}$
- $\prod_A(r) = V(A,r)$
- ${}_A g_F(r) = V(A,r)$

动态规划求join的最优序列：n个关系的join，找到最优的时间复杂度为$O(3^n)$

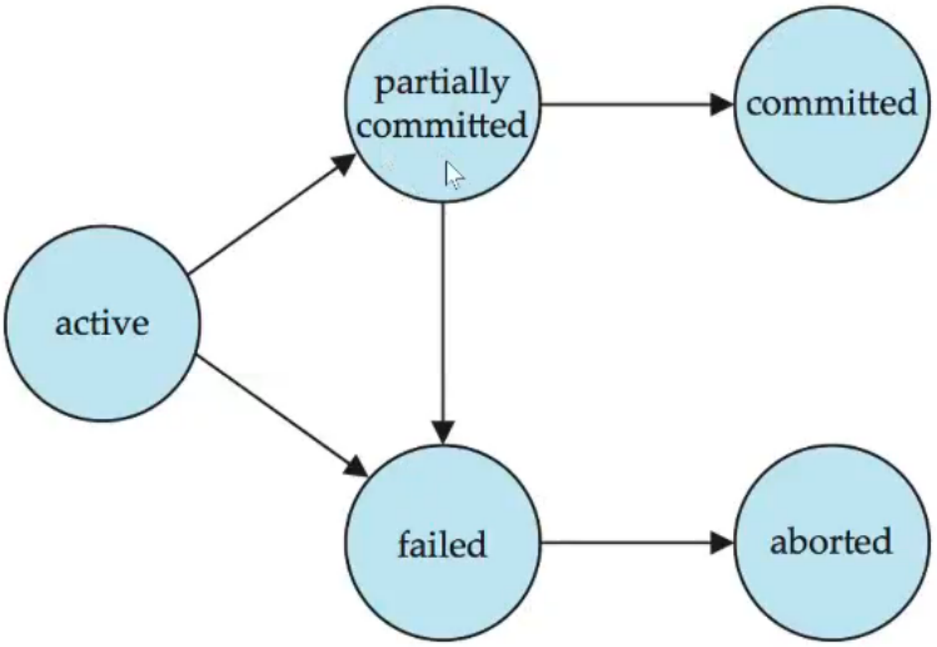
Left Deep Join Tree

---

chapter14 Transactions

commit+rollback

ACID：原子性、一致性、隔离性、持久性



异常：lost update, dirty read, unrepeatable read, phantom problem

Serializability可串行性：冲突可串行性/视图可串行性

precedence graph前驱图：没有环则可串行化

Recoverable schedule可恢复调度：前驱事务commit之后，后行事务再commit

cascading rollback级联调度：读脏数据

cascadeless schedules：前驱事务commit之后，后行才能read

---

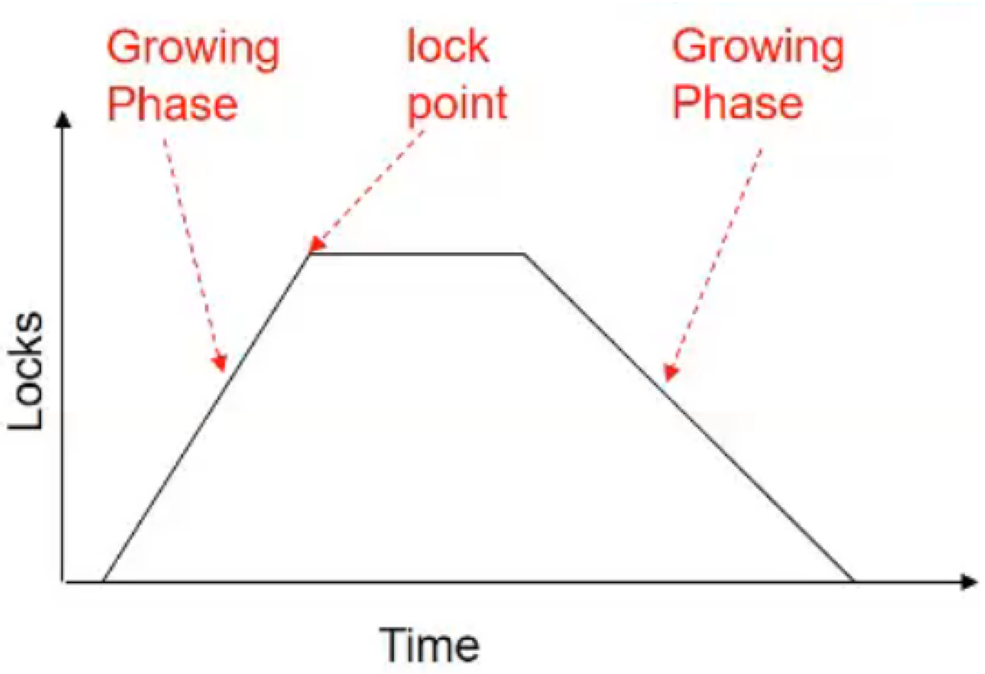
chapter15 Concurrency Control

Lock-Based Protocaols:用锁来控制

1. exclusive(X):排它性（写操作）
2. shared(S):读操作

访问同一个数据	**S**	**X**
S	true	false
X	false	false

两阶段封锁协议：释放锁之后不能再申请锁

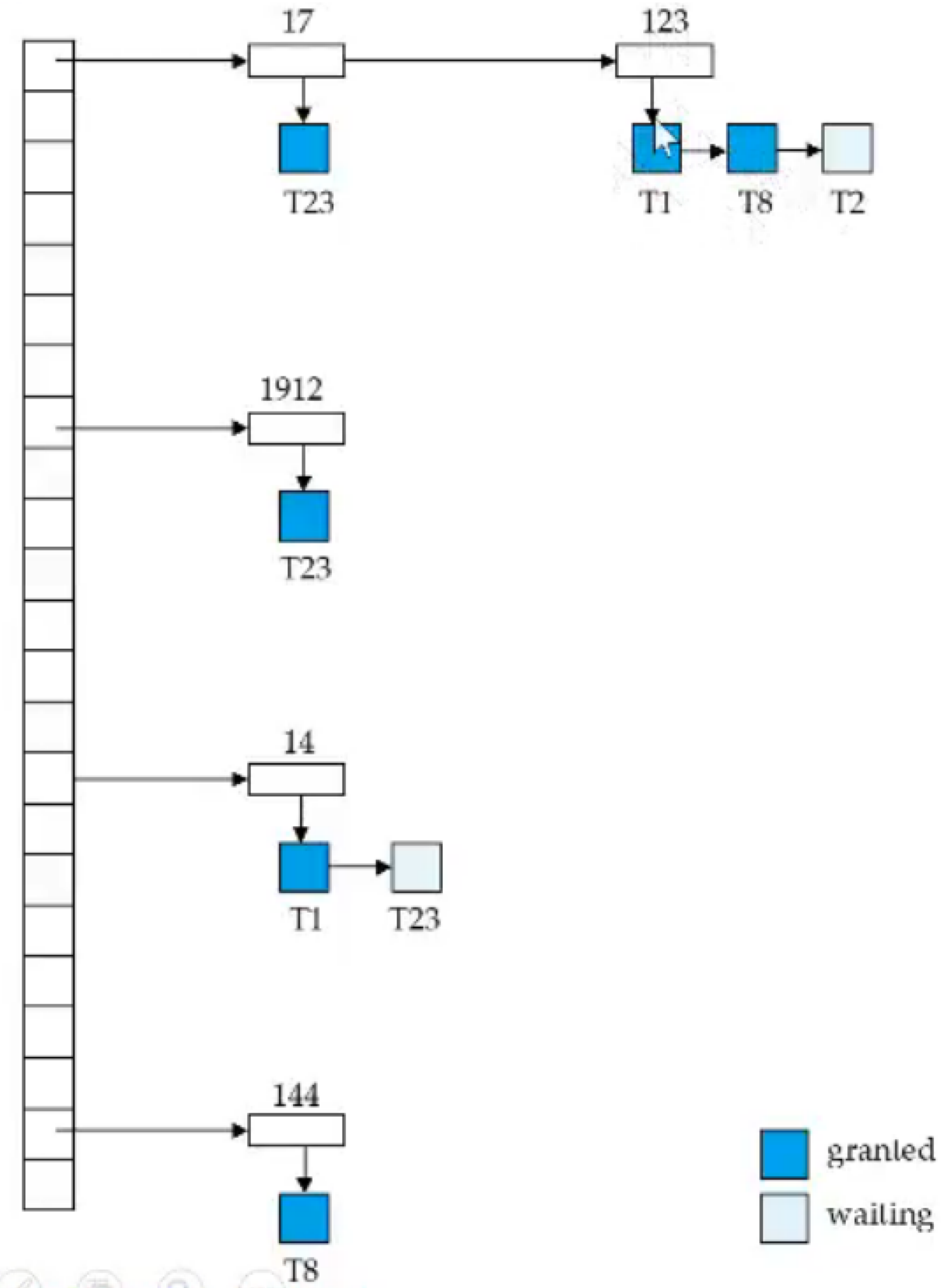


1. basic two-phase locking:保证可串行化
2. strict two-phase locking:事务只有在事务commit/abort的时候才释放X锁保证可恢复性
3. rigorous two-phase locking:所有锁都只有在commit/abort的时候才能释放

2PL protocal一定是可串行的，但可串行的不一定是2PL的

- First Phase：锁越来越多，锁越来越强
- Second Phase：锁越来越少，锁越来越弱

lock table

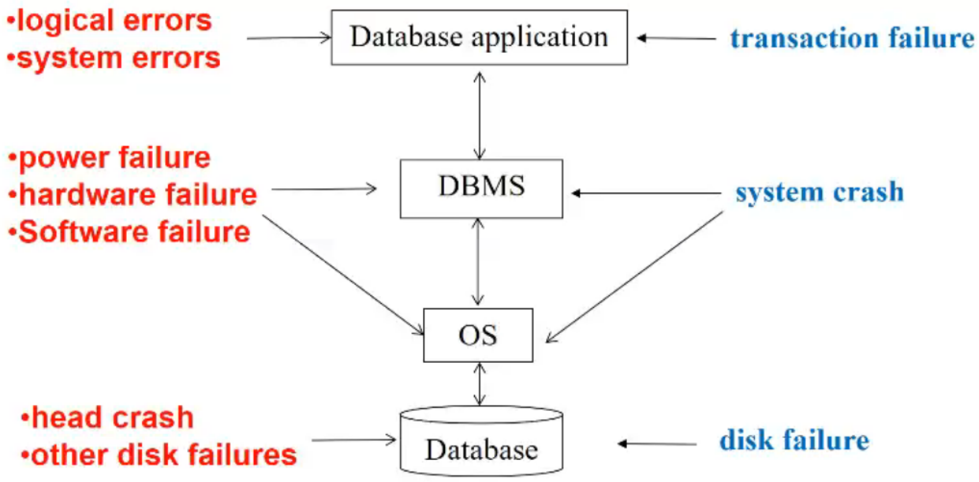


死锁：相互等待

- 预防方法：满足偏序关系可以避免死锁
- 超时方法：一个事务等待一个锁直到一个时间

	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

chapter16 Recovery System



- 保证原子性、持久性、一致性

幂等性Idempotent：执行1次和n次结果一样

Log-Based Recovery

- start:$< T_i start >$
- update:$< T_i, X, V_1, V_2 >$

  V_1是X的old value，V_2是X的new value
- commit:$< T_i commit >$
- rollback:$< T_i abort >$

先写日志原则：先把日志写入stable storage，再写数据

undo撤销：把原日志中的旧值重新写入

- 需要写补偿日志，同时完成后需要写$< T_i abort >$
- 事物有头无尾

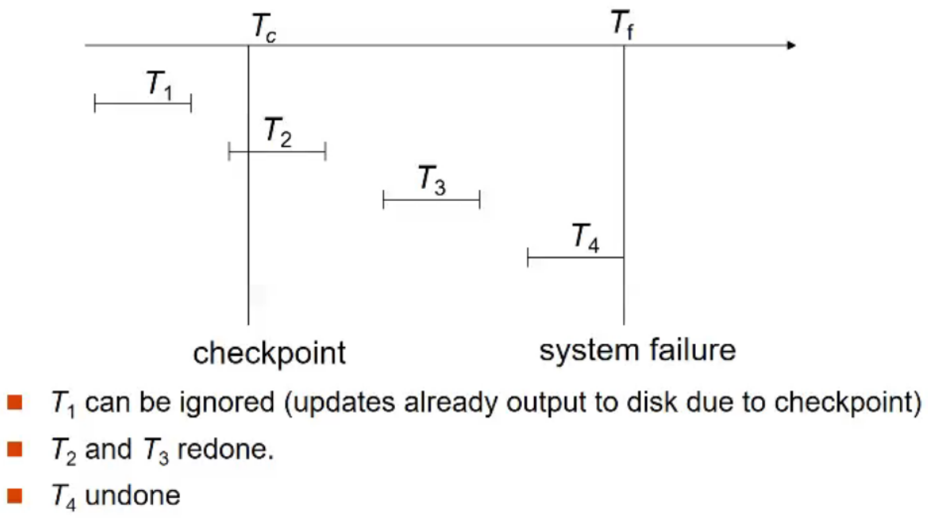
redo重做：再次写入原日志中的新值

- 不需要新写日志
- 这件事务有头有尾

repeating history:对这件事务从头到尾再做一遍，然后redo/undo

Checkpoint：

1. 停止当前所有事务
2. 把所有的日志文件写入stable storage
3. 把所有修改过的缓存区写入disk
4. 往stable storage写入$<$ checkpoint L $>$



- T_1 can be ignored (updates already output to disk due to checkpoint)
- T_2 and T_3 redone.
- T_4 undone

流程

1. 从最近的<checkpoint L>获得undo-list
2. Redo phase：从checkpoint处开始往后扫描
    1. 碰到$< T_i, X_j, V_1, V_2 >$，重做该操作
    2. 碰到$< T_i start >$，把T_i加入undo-list
    3. 碰到$< T_i commit >< T_i abort >$，把$T_i$从undo-list中删除
3. undo phase：从后往前直到找到undolist中所有的start
    1. 碰到undo-list中的事务，恢复并记补偿日志
    2. 碰到undo-list中的$< T_i start >$，记录补偿日志$< T_i abort >$，并把该事务从undo-list中删除

Logical undo operations

1. 当一个操作开始，$< T_i, O_j, \text{operation-begin} >$，Oj为这个操作的ID
2. 当操作在执行的时候，正常记录日志
3. 操作结束时，$< T_i, O_j, \text{operation-end}, U >$，U为对应事务undo时的操作

ARIES

LSN(log sequence number):日志编号

---

chapter23 XML