**1.SQL基础语句 13 7.树索引结构 103 13.数据库设计 180**

**2.关系模型 28 8.哈希索引结构 113 14.冲突控制原理 192**

**3.关系代数语句 44 9.模式细化和范式 122 15.两阶段锁 211**

**4.SQL语句 53 10.查询页面读取计算 138 16.数据库日志 285**

**5.硬盘和文件存储 71 11.查询执行 150 17.数据库恢复 318**

**6.数据库存储 83 12.查询优化 166**

**关系代数语句(relational algebra)**：选行σ，选列π，叉乘x，作减-，并U，重命名ρ，自然连接⋈（公共列相同才保留）

π sname, rating(σ rating > 8 (S2)); π sname(σ bid = 103 (Reserves ⋈ Sailors));

ρ(Temp, (π sname((σ bid = 103 Reserves) ⋈ Sailors))));

**SQL语句**： **create** table Students (sid char(20), name char(20), login char(10), age integer, gpa float);

**select** \* from Stedents S where S.age = 18; **delete** from Students S where S.name = ‘Smith’;

**insert into** Students (sid ,name, login, age, gpa) **values** (‘53688’,’Smith’,’smith@ee’,18,3.2,);

**update** Students **set** age = 18 where name = ‘Smith’; **distinct**用于去重，**order by** A,B，按A排序，A相同则按B排序

select sname from Sailors where sname **like ‘B\_%B’**; ‘\_‘表示可以任意一个的字符，’%‘表示数量任意的字符，例如BOOB

select \*\*\*\* intersect select \*\*\*\* 两个表取交集, select \*\*\*\* union select \*\*\*\* 两个表取并集

select \*\*\*\* except select \*\*\*\* 两个表相减， select sname from Sailors where sid **in** (select sid from Reserves where bid = 103)

select sname from Sailors **S** where **exists** (select \* from Reserves R where **S.sid** = R.sid and R.bid = 103); **子查询**

select \* from Sailors S where S.rating **> any** (select S2.rating from Sailors S2 where S2.name = ‘xx’); 大于任意一个

select \* from Sailors S where S.rating **> all** (select S2.rating from Sailors S2 where S2.name = ‘xx’); 大于所有值

select sid, name, bid from Sailor S **inne**r **join** Reserves R **on** S.sid = R.sid; 内连接

**left outer join** 左外连接，即使不满足选择条件的左表也会被选择输出，输出留空

**right outer join** 右外连接，即使不满足选择条件的右表也会被选择输出，输出留空，

**full outer join** 全外连接 count(),计数 avg(),求平均值 sum(),求和 max(),最大值 min(),最小值

select S.sname ,S.age from Sailors S where S.age = (select Max (S2.age) from Sailors S2);

select S.rating, min(S.age) from Sailors S where S.age > 18 **group by** S.rating **having** count(\*) > 1; group by等于分组循环,having是符合having条件的分组才会被输出 create **view** A **as** select \*\*\*\*\*; 创建视图

create table Sailors (sid integer,. rating integer, **constraint** chk\_rating **check** (rating >= 1 and rating <= 10)); check约束了表的元素的条件，还可以使用constraint命名约束chk\_rating。

create table enrolled (sid char(20), cid char(20), grade char(2), **primary key(sid, cid), foreign key(sid) references Students**)

主键sid, cid，外键sid，该外键的取值范围参照(references)表Students的sid

**超键**(super key): 在关系中能唯一标识元组的属性集称为关系模式的超键 **候选键**(candidate key): 不含有多余属性的超键称为候选键。也就是在候选键中，若再删除属性，就不是键了！ **主键**(primary key): 用户选作元组标识的一个候选键程序主键 **外键(**foreign key)：如果关系模式R中属性K是其它模式的主键，那么k在模式R中称为外键

**超键**：只要含有“学号”或者“身份证号”两个属性的集合就叫超键，例如R1(学号,性别), R2(身份证号,身高), R3(学号,身份证号)等等都可以称为超键！ **候选键**：不含有多余的属性的超键，比如(学号), (身份证号)都是候选键，又比如R1中学号这一个属性就可以唯一标识元组了，而有没有性别这一属性对是否唯一标识元组没有任何的影响！ **主键**：就是用户从很多候选键选出来的一个键就是主键，比如你要求学号是主键，那么身份证号就不可以是主键了！ **外键**：宿舍号就是学生信息表的外键

B+树索引有利于优化范围查询，并且范围越小越优，也可以用于等值查询，但哈希不适用于范围查询

**ISAM**（索引顺序访问树）:静态结构，使用溢出页面。 **B+树**:动态结构，在插入和删除时动态地调整

B+树使用bulk loading，先排序再建立树，可以减少IO次数。使用多次插入，速度较慢

**静态哈希**：页面数量是确定的，可以使用溢出页面，会导致长溢出链。 **可扩展哈希**：如果桶满了，桶进行分裂，00->100,000，避免溢出页面。 **线性哈希**：可以不使用目录处理溢出链的问题桶。0-Next-1已完成分裂，Next后的未分裂，未分裂的桶如果满了，使用溢出页面。只有某个桶满了，触发分裂，Next所在的桶才会进行分裂，分裂完成后，Next加一。如果新插入的数据哈希后的桶小于Next，该桶继续使用新的哈希函数h(level + 1)再次哈希，如果大于等于Next，说明此桶还未分裂，直接插入。当Next分裂了最后一个桶，旧的哈希h(level)作废，level加一，使用新的哈希函数h(level + 1)，Next归0。

**数据库三大范式**：**1NF**: 每一列属性都是不可再分的属性值，确保每一列的原子性，两列的属性相近或相似或一样，尽量合并属性一样的列，确保不产生冗余数据，可以使用拆分列满足。 **2NF**：2NF基于1NF，需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关，这是为了减少冗余，可以使用拆分表满足。 **3NF**：数据不能存在传递关系，即每个属性都跟主键有直接关系而不是间接关系。像：a->b->c属性之间含有这样的关系，是不符合第三范式的。比如Student表（学号，姓名，年龄，性别，所在院校，院校地址，院校电话），就存在上述关系。学号->所在院校->(院校地址，院校电话) ，这样的表结构我们应该拆开来，（学号，姓名，年龄，性别，所在院校）和（所在院校，院校地址，院校电话）

满足2NF一定满足1NF，满足3NF一定满足1NF和2NF，第二范式消除非主属性对键的部分依赖，第三范式消除传递依赖

**BCNF范式**：满足BCNF必须满足3NF，主属性之间不存在依赖关系，没有非平凡依赖

**函数依赖**：1.A->BC A->B, A->C 2.A->B, A->C A->BC 3.A->B, B->C A->C 4.A ⊇B A->B

5.A->B AC->BC 6. A/AB->表中所有元素(ABCD….) A/AB是表的主键

**平凡函数依赖**：如果A ⊇B，则A->B，整体决定部分

**闭包**：闭包就是由一个属性直接或间接推导出的所有属性的集合。如果表R中所有属性都在X的闭包中，X是表R的超键

**无损分解**：表A拆分后得到得两个表，这两个表连接得到得表B与表A相同。如果R中W->Z且W∩Z为空，则R拆分为R-Z和WZ为无损分解。如果R中X->Y违反BCNF，R拆分为R-Y和XY为无损分解。

**聚簇索引**：找到了索引就找到了需要的数据，那么这个索引就是聚簇索引，所以主键就是聚簇索引，修改聚簇索引其实就是修改主键。 **非聚簇索引**：索引的存储和数据的存储是分离的，也就是说找到了索引但没找到数据，需要根据索引上的值(主键)再次回表查询，非聚簇索引也叫做辅助索引。

**自然连接查询花费**：Sailors 500页，每页80个元组，Reserves 1000页，每页100个元组，两个表进行自然连接

多种连接方式：嵌套循环中，最好使用页面数量少的作为outer循环，buffer数量多使用块嵌套循环

**元组嵌套循环**：100\*1000\*500+1000，外层元组数X外层页面数X内层页面数+外层页面数

**页面嵌套循环**：1000\*500+1000，外层页面数X内层页面数+外层页面数

**块嵌套循环**：有102个buffer需要2个buffer进行IO，1000+(1000/100)X500，外层页面数+外层页面数/块数X内层页面数

**索引嵌套循环**：外层页面数+外层页面数X外层元组数X索引寻找内层元组的花费，假如内层元组使用B+树索引，树的层数为2，cost = 1000 + 1000\*100\*(2+1)

**外排序归并**：R排序花费+S排序花费+R页面数+S页面数，假设有35个buffer，计算两个外排序所需要的趟数，1000/35<35，要2趟外排序，500/35<35，也是2趟，2X2X1000+2X2X500+1000+500

如果有20个buffer，2000个页面，PASS0: 2000/20 = 100 runs, PASS1: 100/(20-1) = 6 runs; PASS2: 1 run; 一共需要3躺外排序， Total cost: 2000 x 2 x 3 + 2000 = 14000

**哈希**：N个buffer，表M分成N-1个桶，每个桶不超过N-2个页面，所以必须要求表M的页面数的开方<N，所以表M通常为页面数最少的表，3X(1000+500)，3X(R页面数+S页面数)

哈希内存充足时，速度快，块嵌套循环简单，但花费额外内存，外排序归并内存使用少。

**条件连接优化（5个buffer）**：Select S.sname from Reserves R, Sailors S where R.sid = S.sid and R.bid = 100 and S.rating > 5;

先选择后连接，可以减少IO次数，如果两个表都进行提前条件选择，两个表都要先读入再根据条件筛选。提前条件选择需要使用额外的一个buffer，嵌套循环就需要3个buffer。

**页面嵌套循环**：先读入两个表500+1000，读入筛选后的外层表1000/100，筛选后两个表进行嵌套循环(500/2)X(1000/100)

**外排序归并**：先读入两个表1000+500，读入筛选后的两个表1000/100+500/2，筛选后两个表进行外排序归并(2X2X10)+(2X4X250)+10+250

**块嵌套循环**：先读入两个表1000+500，读入筛选后的两个表1000/100+500/2 ,筛选后两个表进行块嵌套循环10+10/3X250

col=value RF = 1/NKeys(I), col1=col2 RF = 1/MAX(NKeys(I1), NKeys(I2)), col>value RF = (High(I)-value)/(High(I)-Low(I))

**建立索引的选择**：select S.sid from Sailors S where S.rating = 8;

在**rating**上建立索引，**基数索引**：(1/NKeys(I)) \* NTuples(R) = (1/10) \* 500\*80 **聚簇索引**：(1/NKeys(I)) \* (NPages(I)+NPages(R)) = (1/10) \* (50+500) = 55 **非聚簇索引**：: (1/NKeys(I)) \* (NPages(I)+NTuples(R)) = (1/10) \* (50+500\*80) = 4005

如果在**sid**上建立索引，那必须检查所有页面/元组，聚簇索引：50+500，非聚簇索引：50+500\*80

Select E.a, D.b from E,D where E.c between 1000 and 2000 and E.d = ‘x’ and E.f = D.f;

点查询优于范围查询，选择点查询在E.d建立B+树索引。如果选择范围过大，也应该选择在点查询建立聚簇索引

**冲突操作**: 在并行调度中至少有一个操作为写: 读写，写读，写写。**冲突可序列化**: 如果在并行调度与串行调度中，冲突操作顺序相同则正确，P201。**依赖路径**：线程A和线程B有一对冲突操作，则有一条依赖路径，从执行时间早指向执行时间晚。

如果调度依赖图是有向无环图，则该调度是冲突可序列化的，可进行拓扑排序，实现串行调度，P204。

**视图等价**：对同一事物，只要是有一个线程读了它的初始值，另外一个线程也必须读它的初始值。如果在一个线程里，一个操作是读了一个写操作后的值，另一个线程也必须读同样写操作后的值。如果在一个线程里最后进行了写操作，则另一个线程也要在最后进行同样的写操作。**视图可串行化**：如果某个调度视图等价于一个串行调度，则称该调度是视图可串行化的。

如果一个调度是冲突可序列化，那么也是视图可串行化的。

**S-LOCK共享锁**：事务T对A加上共享锁后，则其他事务只能对A再加共享锁，不能加互斥锁，直到已释放所有共享锁，共享锁的事务只能读数据，不能修改数据。 **X-LOCK互斥锁**：事务T对A加上互斥锁后，则其他事务不能再对A加任何类型的锁，直到将A上的互斥锁释放为止，互斥锁的事务既能读数据，又能修改数据。

**两阶段锁**：第一阶段成长(加锁)，每个事务都从锁管理器请求它需要的锁，锁管理器授予/拒绝锁请求。第二阶段收缩(解锁)，事务只允许释放它以前获得的锁，不允许获取新锁。两阶段锁可以确保冲突可序列化，有可能导致级联中止，也有可能导致**脏读**(事务B去查询了事务A修改过的数据，但是此时事务A还没提交，所以事务A随时会回滚导致事务B再次查询就读不到刚才事务A修改的数据了)，也有可能导致死锁。 **严格的两阶段锁**：所有锁都在最后被释放，一个数据被事务A进行写，其他事务不能对该数据进行读或写直到事务A结束，不会导致级联中止。

**死锁检测**：创建waits-for图，如果事务Ti等待事务Tj释放锁，则Ti到Tj有一条边，如果waits-for图是有环图，则有死锁

**死锁预防**：根据时间戳分配优先级，越早的时间意味着越高的优先级。Wait-Die（“Old等待Young”）：如果T1具有更高的优先级，则T1等待T2，否则T1中止。Wound-Wait（“Young等待old”）：如果T1具有更高的优先级，则T2中止，否则T1会等待。在等待锁时只允许一个方向。当事务重新启动时，其(新)优先级是其旧时间戳。

**意向锁**：是表级锁，意向共享锁(IS)表示事务意图在表中的单个行上设置共享锁，意向互斥锁(IX)表明事务意图在表中的单个行上设置互斥锁。事务对一张表的某行添加S/X锁前，要先获得IS/IX锁。意向锁IS和IX互相兼容，事务A加了表的IX或者IS锁，只代表事务A已锁定一行或者将要锁定一行，事务B也可以锁定其他的行，所以事务B也可以获得表的IS锁或者IX锁的。IS与其他事务的X不兼容，IX与其他事务的S/X不兼容。

**数据库ER图**：**实体**：用长方形表示，数据模型中的数据对象；**属性**：用椭圆形表示，唯一属性主键，用下划线表示；**关系**：用菱形表示，表现数据对象与数据对象之间的联系，1对1，1对N，M对N。弱实体：一个实体必须依赖于另一个实体存在，前者是弱实体，后者是强实体，弱实体和强实体的联系必然只有1:N或者1:1，之间的关系/弱实体用双线菱形/长方形

**计算关系依赖的最小覆盖**：设R<U,F>，U={A,B,C,D,E,G}，F={AB→C,D→EG,C→A,BE→C,BC→D,CG→BD,ACD→B,CE→AG}

1.将所有的函数依赖**右边**变成单个属性F={AB→C,D→E,D→G,C→A,BE→C,BC→D,CG→B,CG→D,ACD→B,CE→A,CE→G}

2.去掉F中多余的函数依赖，设AB→C为冗余的函数依赖，计算(AB)F1+，设X(0)=AB；计算X(1)，扫描F1中各个函数依赖，找到左部为AB或AB子集的函数依赖，因为找不到这样的函数依赖，故有X(1)=X(0)=AB，算法终止，不能去掉AB→C

3.设CG→B为冗余的函数依赖，计算(CG)F2+，设X(0)=CG；计算X(1)：扫描F2中的各个函数依赖，找到左部为CG或CG子集的函数依赖，得到一个C→A函数依赖。故有X(1)=X(0)∪A=CGA=ACG；计算X(2)，找到左部为ACG或ACG子集的函数依赖，得到一个CG→D函数依赖。故有X(2)=X(1)∪D=ACDG；计算X(3)，找到左部为ACDG或ACDG子集的函数依赖，得到两个ACD→B和D→E函数依赖。故有X(3)=X(2)∪BE=ABCDEG，因为X(3)=U，算法终止，CG→B是冗余的函数依赖。

4.计算得到CE→A也是冗余的函数依赖，CG→D不是，所以F={AB→C,D→E,D→G,C→A,BE→C,BC→D,CG→D,CD→B,CE→G}

**设**有一个员工数据库员工(eid:8字节，ename:16字节，did:4字节，电子邮件：12字节），其中eid和ename分别是员工的id和姓名，did是员工所在部门的id。假设有50000个员工记录和500个部门（即每个部门平均有100名员工）。页面大小为1000字节，指针大小为4字节。假设员工文件按did顺序排序，并且没有索引。估计检索在具有给定did的部门中工作的所有员工的记录的页面访问成本：每个元组的大小为8+16+4+12=40字节，每页元组个数1000/40=25，一共需要的页数50000/25=2000，一个部门100个员工，需要100/25=4个页面，cost=

**设**1000页，一个元组四个元素，选择其中一个元素，筛选条件后剩26%符合，结果得到的页面大小为: 0.26\*1000/4=65页

**设**F = {AB→A, AB→C, AB→D, C→A, D→B}，找出R的候选键：

,所以候选键为AB，AD，BC，CD

数据流图是在数据库的需求分析阶段完成的，设计关系模式是逻辑设计阶段的任务，sql语句中的view视图是数据库的外模式，属性不能取空NULL为实体完整性规则，事务的持久性是指事务一旦提交，对数据库的改变是永久的，事务的隔离性实至一个事务内部的操作及使用的数据对并发的其他事务时隔离的，只有出现并发操作时，才有可能出现死锁，sql语言具有数据定义、操作、控制的功能，插入异常是指应该插入的数据未被插入，删除异常是指应该删除的数据未被删除