

数据库知识点整理

第一章：数据库概论

知识点1 数据库的基本概念

- 1.数据库(DataBase): 为——一个共同的目的而而保存起来的所有数据的集合称为数据库。
- 2.数据库管理系统(DBMS - database management system): 数据库管理系统是——一套计算机程序(软件), 他把企业的数据以记录的形式在计算机中保存起来。
- 3.数据模型(data model): 是一套描述如何将现实世界的数据在概念上用电子信息表示的定义, 它还表示为用来操作这些数据的一类操作。数据模型:
 - Hierarchical Data Model/层次数据模型(有向树) → (原始概念)
 - Network Data Model/ 网状数据模型(有向无环图) → (相比树可以允许交叉)
 - (Relational Model/关系模型(表) + Object-Oriented Model) → 生成
 - Object-Relational Model/对象关系模型 (不受第一范式规则约束的关系模型、表中元素可以是集合)。
- 4.数据库用用户 (DataBase User)
 - 最终用户——交互式用用户
 - 临时用用户——用用SQL访问DBMS的用用户
 - 初级用用户——通过菜单程序访问DBMS的用用户
 - 应用程序员——编写菜单程序的程序员
 - 数据库管理理员 (DBA) ——管理理DBMS的专家 (负责设计和维护)

第二章：关系模型的基本概念与基本理论

知识点2 关系模型的基本概念

- 1.数据结构:
 - 表/关系(table/relation): file of records。表是以行和列的形式组织起来的数据的集合(wiki)。
 - 列/属性(column/attribute): field names of records。
 - 行/元组(row/tuple): records of a file。
 - 表头/模式(table heading/schema): 列名的集合。
 - 域(domain): 可以被用作表的属性值的常数的集合。Ex. domain(sno)={1,2,3}
 - 关系模型中, 第一行称为模式, 除第一行外每一行称为元组; 每一列称为属性, 每一列的第一行称为属性名
 - 用一个空值参与一个逻辑表达式的计算, 其计算结果是逻辑假
空值参与算术运算, 计算结果也是空值

数据独立性: 对某个问题的查询, 能够无视数据的变动给出答复

2.约束规则(relational rule):

- Rule 1. **第一范式规则**: 在定义的表中, 关系模型坚持不允许含有多值属性和含有内部结构的列。在关系模型中, 约束规则“First normal form rule”的含义是属性值的原子性——不能存在多值属性
- Rule 2. **只能基于内容取行规则**: 只可以通过行的内容, 即每一行中所存在的属性值来检索列。基于行获得内容规则: 含义是属性的无序性和元组的无序性
- Rule 3. **行唯一性原则**: 关系中的任何两个元组的值在同一时刻不能是完全相同的。
- Rule 4. **实体完整性规则**: 表T中的任意行在主键列的取值都不允许为空值。

3.数据完整性约束:

- a) 实体完整性——主关键字不能有空值
- b) 参照完整性——外键

c) 用户定义完整性

可以通过**触发器**来进行数据完整性以及数据安全性的检查

4.超键(Superkey):表的任意两行数据在该列集合(属性的集合)上都有唯一的值。可通过该列的数值来区分任意两个元组

候选键/键(Key): 组成键的列集合中再也没有子集也是表的超键(也就是无多余属性)

(可以证明任何表都有至少一个Key)

主键: 被数据库设计者选择出来作为表中特定行的唯一性标识符的候选键。

空值(null value): 未知的或尚未定义的值。(计算平均值时被剔除\不能作为筛选依据\可以用!
=筛选掉Null值)

知识点3 关系代数(relation algebra)

1.表/关系在关系代数中的表示

在关系数据模型上进行数据访问, 其访问结果也构成了一个**关系**

2.关系代数中的运算符

集合运算: $R \cup S$ $R \cap S$ $R - S$ $R \times S$

兼容表: 表头相同且属性的域相同(只有**兼容表**可以进行并、交、差运算)。

PS: 兼容表中的表头相同不包括顺序相同

并(\cup)、交(\cap)、差($-$)是针对行的操作;

赋值($:=$)是针对表的操作;

PS: 可以认为笛卡尔积是基于行的组合

乘(\times)即对两个表的所有做笛卡尔乘积(组合)

其中: \cup (并), $-$ (差), \times (乘), 投影, 选择是五个基本运算。其他的运算都可以由这几个基本运算进行表示。



3.自然关系运算: **投影、选择、连接、除**

投影运算: R 在 A_{i1}, \dots, A_{ik} 上的投影用 $R[A_{i1}, \dots, A_{ik}]$ 表示, 如CUSTOMER[name]或者 $CN := \rho_{cname}(C)$ 即把某几列从表中选出, **重复的值会被合并。**

选择运算: S where C , C 也可以是 C and C' , C or C' 或者not C 。

连接运算: $R \bowtie S$, 即选出属性重叠部分值相同的元组。没有共同属性时连接的结果与笛卡尔乘积相同。

除运算: 乘积的逆运算(每行除后取交集, 注意到 R/S 的情况下, 如果 S 中有多个元组, 则应该对于每一个元组除运算结果取交集), **用于处理“所有”问题。**

$R/S=T$ 但是 $T*S$ 不等于 R , 讲道理是 R 的子集。

4.扩充运算: 外连接、 θ 连接

*** 外连接**: $R \bowtie_0 S$, 即将两个表完全“并”起来, 无对应值时补null。

(左外链: $R \bowtie_{L0} S$, 即保留左边未匹配的行, 右边补null。)

(右外链: $R \bowtie_{R0} S$, 即保留右边未匹配的行, 左边补null。)

*** θ 连接**: θ 可以是 $\{>, <, >=, <=, =, \neq\}$ 中的一个。表示为 $R \bowtie_{A_i \theta B_j} S$ 。若 A_i 和 B_j 同名, 则表示为 $R \bowtie_{R.A_i \theta S.B_j} S$ 。(在两个表存在同名属性时使用, 相当于把where语句中的内容放到了 \bowtie 之后和 S 之前的部分写)

ex: $(ORDERS \bowtie_{Orders.qty > Products.quantity} PRODUCTS)[ordno]$

等价于: $(ORDERS \times PRODUCTS \text{ where } O.qty > P.qty)[ordno]$

第三章：关系数据库语言SQL

知识点4 数据访问命令的基本结构

1.建立表，例：

```
creat table customers (  
cid char(4) not null,  
cname varchar(13),    --最大长度为13的可变长字符串 (--后面为注释)  
city varchar(20),  
discnt real,  
primary key(cid));
```

CHAR(n)	定长
VARCHAR(n)	变长
LONG	超级变长

NUMERIC(p,s)

p:总数字个数

s:小数点后几位 (可能为负)

当以NUMERIC(x,x)形式声明默认为0

否则默认全保留

知识点5 基本的数据查询命令

1.在SQL查询语句中，**select**子句和**from**子句是一条映像语句中必不可少的两个组成部分

2.单表查询

例：select distinct o.ordno, o.cid, o.aid, o.pid, o.qty*p.price-(c.discnt+a.percent)*(o.qty*p.price)
as profit

from custommers c, orders o, agents a, products p

where c.cid = o.cid and o.aid = a.aid and p.pid = o.pid and a.city

= 'New York';

字符串用单引号。

需删除select后的重复行时应使用distinct，保证重复行不被删除

使用all，select默认不删除重复行。

as用于定义表的别名，可被省略。

内层子查询能使用来自外层select语句的变量，反之则不然。

3.多表查询

(1) UNION运算符（逻辑并操作），自动删除重复行，例：

select city from customers union select city from agents;

在IN谓词、量化比较谓词以及EXISTS谓词所包含的子查询里，不能使用子查询的UNION形式。

(2) UNION ALL：大致同UNION，但允许出现重复行。

(3) INTERSECT运算符（逻辑交操作），自动删除重复行，例：

select distinct x.cid from pid = 'p01' （选择既买p01又买p07的顾客的cid）

intersect select cid from orders where pid = 'p07'

(4) INTERSECT ALL：

大致同INTERSECT，但允许出现重复行。

(5) EXCEPT运算符（逻辑差操作），自动删除重复行，例：

select c.cname from customers c

except select c.cname from customers c, orders x

where c.cid = x.cid and x.aid = 'a05';

(6) EXCEPT ALL：大致同EXCEPT，但允许出现重复行。

(7) JOIN：连接查询，必须显式指定参加连接的列（on或using）。例：

(1) select cname, city, latitude, longitude

from customers c join cities x on c.city = x.cityname

(2) select * aname, aid, total

from sales join agents using (aid);

连接可以在SELECT 语句的FROM子句或WHERE子句中建立。

4.扩展的查询谓词

Relational Algebra	SQL Predicate
natural join	IN
	= SOME
	EXISTS
difference	NOT IN
	<> ALL
	NOT EXISTS

1. (NOT) IN 谓词，例：(NOT IN 等价于 <>ALL)

```
select distinct cid from orders
  where aid in ( select aid from agents
                where city = 'Dulth');
```

2. 量化比较谓词(<, <=, =, <>, >, >=), {SOME, ANY, ALL}, 例：

```
select aid from agents where percent <= all
(select percent from agents);
```

* = SOME与IN含义相同，<> ALL与NOT IN含义相同。

* 小于任何一个不是用<=ANY而是<=ALL

* 记住ALL一定要写

3. (NOT) EXISTS谓词 (对每一行检查子查询的检索结果是否为空集)：

```
select distinct cid from orders x
  where pid = 'p01' and exists ( select * from orders
                                where cid = x.cid and pid = 'p07');
```

4. (NOT) BETWEEN谓词，例：

```
select * from scott.emp where sal not between 2000 and 3000;
```

5. IS (NOT) NULL谓词，例：

```
select * from scott.emp where job is not null;
```

6.(NOT) LIKE谓词，例：(检索不以M开头的工作信息) like 关键字不能用于子查询

```
select * from scott.emp where job not like 'M%'
```

7.下划线 (_)：任意单个字符的通配符。

8.百分号 (%)：包含零个或多个字符的任意序列的通配符。

9.转义字符：用在需要按字面含义引用的字符之前，可自定义，例：

```
select cid (寻找以'TIP_ 开头的')
from customers
where cname like 'TIP\_%' escape '\';
```

转义字符只对紧跟在其后面的字符有效。应在同一行里使用两次转义字符。

知识点5 复杂的数据查询命令

1.统计查询 (集合函数)

COUNT：参数可以是任何类型的列或表，计算行数。(默认是计算重复的值的，注意若不计重复必须显式加上COUNT (distinct city)，假使列名是city)

SUM: 参数必须是数字类型的列或表, 计算各行之和。

AVG: 参数必须是数字类型的列或表, 计算各行的平均数。

MAX: 参数必须是数字或字符类型的列或表, 计算各行中最大的值。

MIN: 参数必须是数字或字符类型的列或表, 计算各行中最小的值。

* 集合函数不能出现在where子句中, 除非是在子查询的选择列表中。

* null值不等于包括null值在内的任何值, 所有函数都忽略null值。

* 对空集使用统计查询时, COUNT()返回0, 其它函数均返回NULL。

2. 分组统计查询 (GROUP BY)

* GroupBy的列集合要和select中的非函数列集合对应, 允许内部顺序不一致

```
例: select aname, a.aid, pname, p.pid, sum(qty)
      from orders x, products p, agents a
      where x.pid = p.pid and x.aid = a.aid and x.cid in('c002','c003')
      group by a.aid, a.aname, p.pid, p.pname;
```

语句的执行顺序为:

- 1、对from语句中的表做笛卡尔乘积;
- 2、删除不满足where子句的行;
- 3、根据group by对剩余进行分组;
- 4、求出选择列表中的表达式的值。

3. 分组选择统计查询 (GROUP BY ... HAVING)

```
例: select pid, aid, sum(qty), as total
      from orders
      group by pid, aid
      having sum(qty) > 1000;
```

语句的执行顺序为:

- 1、对from语句中的表做笛卡尔乘积;
- 2、根据group by对剩余进行分组;
- 3、删除不满足having子句的行;
- 4、求出选择列表中的表达式的值。

放在group by 和having 语句的后面

ORDER BY qty DESC (按照qty列的值进行降序输出)

ORDER BY qty ASC (按照qty列的值进行升序输出)

4. 多层嵌套的NOT EXISTS查询 (除法)

例: o[cid, aid]÷(a where city='New York')[aid]

```
SELECT c.cid
FROM customers c
WHERE NOT EXISTS (
  SELECT *
  FROM agents a
  WHERE a.city = 'New York' and NOT EXISTS (
    SELECT *
    FROM orders o
    WHERE o.cid = c.cid and o.aid = a.aid ) )
```

被除数在里层，除数在外层。

SQL不允许集合函数内部包含子查询，也不允许from子句包含子查询。

知识点6 数据更新命令

1.INSERT语句，例：

(1) 直接插入：insert into orders (ordno, month, cid, aid, pid)

values (1107, 'aug', 'c006', 'a04', 'p01');

(2) 子查询插入：INSERT INTO swcusts

select *

from customers

where city in ('Dallas', 'Austin');

insert中显式表示的列名序列决定了values () 中的序列

insert后列名可缺省，这时按照原来表的序列进行插入。

insert后未指定的列缺省值为空值。

insert into表名后的子查询不写在括号里面。

UPDATE语句，例：

UPDATE agents

SET percent = 1.1 * percent

WHERE city = 'New York';

2.DELETE语句，例：

DELETE FROM agents

WHERE city = 'New York';

第四章：对象-关系SQ

对象关系数据库在数据类型方面的扩充主要是：对象类型和集合类型。

知识点7 Oracle中的对象类型

1.创建对象类型，例：

CREATE TYPE name_t AS OBJECT (

lname varchar(30),

fname varchar(30),

mi char(1));

可复用，生成后与内置数据类型一致

2.对象的定义（不需要指定主键）

CREATE TYPE person_t AS OBJECT(

ssno int,

Pname name_t, (name_t是对象类型)

Age int);

3.使用对象类型直接创建表（需要指定主键），例：

CREATE TABLE people OF person_t(

PRIMARY KEY(ssno));

4.访问对象中的数据，例：

Select t.tid, t.tname.fname, t.tname.lname

from teachers t

where t.room = 123;

访问对象属性时必须使用对象的别名，即：

select tid, tname.fname, tname.lname

from teachers

where room = 123; 的写法不被允许。

5.插入元组(有对象时会使用对象构造函数)

insert into teachers values

(1234, name_t('Einstein','Albert','E'), 120)

insert语句不支持表的别名

6.清除对象，例：

drop type person_t; 

7.删除表，例：

drop table people;

8.修改元组中对象的属性值，例：

update people p

set p.pname = name_t('Gould', 'Ben', null);

其中pname是类型为name_t的属性。

也可以修改成员对象的成员属性

update people p

set p.pname.mi = 'A' ;

还可以修改整个元组(people是一个对象类型)

update people p

set p=peron_t(123, name_t('Gould', 'Ben', null), 55)

update语句允许使用表的别名

* value()方法返回对象取值 (而不是元组取值)，例：

select value(p)

from people p

where age > 25;

知识点8 对象引用类型 (使用引用仍需保留原来的数据)

1.定义对象引用，例：

create type order_t as object (

ordno int,

cid char(4),

ordcust ref customer_t);

其中customer_t为已定义对象。

之后先create table customers of customer_t (primary key (ordno)) ;

使用带有引用类型的对象直接创建表，例：

create table orders of order_t (

primary key (ordno),

scope for (ordcust) is customers);

其中customers为由customer_t直接创建的表。

这里的scope for 用来限定REF属性的取值范围

2.REF()函数返回对象 (元组) 的引用指针，例：

select c.cname

from customers c

where not exists (

select * from orders x

5.0

对象构造函数

typename(argument,...)

返回对象取值的函数

value()

引用类型可用于实现对象类型之间的嵌套引用

创建含有引用类型的关系表

1. 先通过预定义基本类型直接建立基本关系表
2. 用带有引用类型对象直接创建表

where x.ordcust = ref(c) and x.aid = 'a05';
引用只能与引用或指针比较。

3. **DEREF()**函数返回指针所指向的对象的值，例：

```
select value ( p ), deref ( p.partner )  
  from police_officers p;
```

引用本身只是一个指针。

4. **IS DANGLING**谓词，例：

```
select o.cid from orders o  
  where o.ordcust IS DANGLING;
```

用于判断所引用的元组对象是否存在。如果所引用的元组对象不存在，那么该谓词返回逻辑真(TRUE)，否则返回逻辑假(FALSE)。

主要用于检查错误的对象引用指针

5. **IS NULL**谓词：

查找取值为空(NULL)的REF属性。

IS NULL不等于 IS DANGLING

6. 对象类型不能递归定义，但**REF可以实现递归引用**。

```
create type police_officer_t as object (  
  badge_number integer,  
  partner ref police_officer_t );  
create table police_officers of police_officer_t(  
  primary key (badge_number),  
  scope for (partner) is police_officers );
```

7. 因为类型中的引用关系的影响，需要进行强制删除，例：

DROP TYPE customers_t FORCE;

8. 删除有引用关系的表和类型时：

- (1) 在删除类型(drop type)之前需要**先删除表(drop table)**；
- (2) 在删除类型(drop type)时需要采用**强制删除**的方式。

知识点9 Oracle中的集合类型

1. 嵌套表类型

创建表类型，例：CREATE TYPE **dependents_t AS TABLE OF person_t;**

创建嵌套表，例：

```
create table employees (  
  eid int,  
  eperson person_t,  
  dependents dependents_tab,  
  primary key (eid)  
) nested table dependents store as dependents_tab;
```

2. 使用一个create语句创建两个表，分别是employees和dependents_tab。一个表中可定义多个**nested table**，每一个table-type属性，都需要有一个对应的**nested table**。

3. 访问嵌套表，例：

```
select dependents
```



```

from employees
where eid = 101;
select eid
from employees e
where 6 < ( select count(*)
from table(e.dependents));

```

table()函数将嵌套表转化为表。

from语句后面必须是表，不能是嵌套表，因此必须使用table()函数进行转换。

统计函数的作用对象只能是表，不能是嵌套表。

在oracle数据库中，对象之间可以进行相等比较，但嵌套表之间不能。

4. 外连接，例：

```

select e.eid, d.ssno
from employees e, table(e.dependents) (+) d;

```

(+)表示空值将被填入另外一个表的保留行的那一侧。

5. 嵌套游标，允许增加一个扫描每一行中嵌套表的第二层循环。例：

```

select eid, cursor ( select count(*) from table(e.dependents) )
from employees e;

```

嵌套游标可以用来实现对于嵌套表的属性统计功能，但是也可以使用其他方法：

比如select eid, (select count(*) from table(e.dependents)) from employees e;

或者 select eid, count(*) from employees e, table(e.dependents) group by eid;

知识点10 数组类型

1. 创建数组类型，比如创建4个整型组成的数组：

```

create type extensions_t as varray(4) of int;

```

2. 使用数组类型定义表中属性，例：

```

create table phonebook (
  phperson person_t,
  extensions extensions_t);

```

3. 可以使用 table() 将一个 VARRAY 属性转换成一张嵌套表。

NESTED TABLE 和 VARRAY 的比较：

	Nested table	VARRAY
成员的排列次序	无序	有序
成员的最大数目	没有限制	确定的值
成员的存储组织	单独的存储表	直接存储在表中
访问模式	可执行insert操作 或通过update操作 修改成员的取值	不能通过执行insert操作 或直接对成员的update操作 修改成员取值，只能通过 update语句修改整个 VARRAY属性的取值

第五章：数据库的应用开发

SQL: Structured Query Language, 结构化查询语言。

ESQL: Embedded SQL, 嵌入式SQL。SQL嵌入在编程语言中

ISQL: Interactive SQL, 交互式SQL。使普通用户能够使用数据库

宿主变量: 逻辑程序可以使用的普通的程序变量, 无论何时使用都要加上“:”。

是引用主语言 (c语言) 来说明的程序变量, 主要用于数据输入输出。

知识点11 嵌入式SQL

1. 前缀: `exec sql` 后缀: 一定要加分号

2. 声明host variable, 例:

`exec sql begin declare section;`

`char cust_id[5]; //c语言的声明方式`

`char cust_name[14];`

`float cust_discnt;`

`char user_name[20], user_pwd[20];`

`exec sql end declare section;`

host variable 使用前必须先声明
原因是

1. 要分配内存 + 确定编译期类型
- 2.

嵌入式数据库和交互式数据库在命令格式上的差别

```
exec sql select count(*)
into :host_var
from customers ;
```

- a) 以‘`exec sql`’开始, 结束要加上‘;’
- b) 单行接收选择语句的结果
- c) 宿主变量, 前面要有冒号

host variable 可以干双向的事情,
是宿主语言和DBMS之间的桥梁

4. 错误处理, 例:

`exec sql whenever sqlerror goto report_error;`

需要避免死循环

`exec sql whenever not found goto notfound;`

错误种类包括: `SQLERROR`、`NOT FOUND`和`SQLWARNING`。

错误的处理方法包括:

- (1) `CONTINUE`
- (2) `GOTO label`
- (3) `STOP` (结束程序执行, 撤销未提交的事务并断开数据库连接)
- (4) `DO function` | `BREAK` | `CONTINUE` (执行指定的C函数, 函数返回后回到断点处继续执行)。或中断, 根据选了`BREAK` 还是`Continue`决定

5. 连接数据库, 例:

`EXEC SQL CONNECT TO :user_name IDENTIFIED BY :user_pwd ;`

6. 数据库访问, 例:

`exec sql select count(*)`
`into :host_var`

from customers ;

以“exec sql”开头，“;”结尾。

into语句将select的结果赋值给into后的主语言变量（加上“:”）。

7.提交/撤销（在断开连接之前需要进行的操作），例：

EXEC SQL COMMIT WORK; (SQL99)

EXEC SQL ROLLBACK WORK; (SQL99)

8.断开数据库连接，例：

exec sql commit release ; (ORACLE) --断开连接后自动执行一次commit

exec sql rollback release ; (ORACLE) --断开连接后撤销所有为提交的工作

在关系数据库语言SQL中，用于显式地结束一个事务的命令是commit和rollback

9.使用嵌入式SQL的数据库应用程序的基本程序流程

1、The Declare Section

2、Condition Handling

3、SQL Connect to Database

4、Main Body of Application Program

5、SQL Disconnect

知识点12 游标(多行的数据交换)

1.与游标有关的四条语句：

1、声明游标（游标名字是agent_dollars,指向（aid, sum（dollars））的数对）。例：

```
EXEC SQL DECLARE agent_dollars CURSOR FOR
select aid, sum(dollars)
from orders
where cid = :cust_id
group by aid ;
```

游标后的子查询不加括号。

2、打开游标，例：

EXEC SQL OPEN agent_dollars ;

打开游标会执行声明时的select语句。

打开游标后，游标的指针会指向结果集的第一行之前。



3、通过游标读取一行，例：

```
while (TRUE) {
    exec sql fetch agent_dollars
    into :agent_id, :dollar_sum;
    printf("%s %11.2f\n", agent_id, dollar_sum);
}
```

每次执行fetch都将指针向后移一行。

4、关闭游标，例：EXEC SQL CLOSE agent_dollars ;

游标被关闭后还可以被打开。

2.游标的作用：

- 1) 游标是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果
- 2) 每个游标区都有一个名字
- 3) 用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理

3.指示器的使用，可以在select中指示某个值是否为空，也可在update里面把某个值更新为空值
下面例子把某个cid的discnt赋值为null。例：

cd_ind=-1

exec sql update customers

set discnt = :cust_discnt INDICATOR :cd_ind

where cid = :cust_id;

Indicator 本是一个额外变量（超过select数）

INDICATOR值的含义：

1. =0：表示一个非空值被赋给host variable（cust_discnt）；
2. >0：一个被截断的字符串被赋值给host variable；
3. =-1：数据库中值为null，host variable中为一个无意义值。

第六章：数据库设计

知识点13 ER模型（实体-关系模型）

使用抽象化的方式描述一个数据

1.基本组成：实体(entity)，属性(attribute)，联系(relationship)

实体：具有公共性质的可区分的现实世界对象的集合

属性：描述实体或关系性质的数据项

联系：定义实体实例之间的对应规则（环：自己到自己）

2.属性类型：

* 超键：可以区分不同的实体实例的属性集合

* 标识符（候选键）：区分不同的实体实例的属性集合，且无冗余属性

* 主键：设计者选择的特定的标识符

* 描述符（descriptor）：非键属性

* 复合属性，例（student_name）：矩形 椭圆 直线

3.多值属性，例（hobbies）：矩形 椭圆 双线 直线

实体参与的基数，例：

max-card(E, R) = N

min-card(E, R) = 0

max-card(F, R) = 1

min-card(F, R) = 1

max-card(X, R) = 1 → single-valued participation（单值参与）

max-card(X, R) = N → multi-valued participation（多值参与）

min-card(X, R) = 1 → mandatory participation（强制参与）

min-card(X, R) = 0 → optional participation（可选参与）

1-1: 两个实体都是单值参与。

N-1: 一个实体是多值参与，另一个是单值参与。

N-N: 两个实体都是多值参与。

4.E-R模型到关系模型的转换规则：（即实体和二元联系分别怎么用表来表示）

四大问题：

1. redundancy 数据冗余
waste of space
2. abnormality of update 修改异常
waste of time
easy to wrong
3. abnormality of delete 删除异常
might lose inf
4. abnormality of insert 插入异常
unsuccessful insert
(减了个院系还没有招生，没法先把院系输入进去)
5. 更新异常
一列上有重复的列就会引起更新异常——更新表对应的某个实体实例的单个属性需要将多行更新

Rule 1. E-R图中的每一个实体映射到关系数据库中的一个表，并用实体名来命名这个表。表的列代表了连接到实体的所有简单单值属性（可能是通过复合属性连接到实体的，但复合属性本身并不变成表的列）。实体的标识符映射为该表的候选键，实体的主标识符映射为主键。实体实例映射为该表中的行。

Rule 2. 给定一个实体E，主标识是p。且E有一个多值属性a，那么a映射成自身的一个表，该表按照复数形式的多值属性名命名。这个新表的列用p和a命名，表的行对应(p,a)值对，表示与E的实体实例关联的a的属性值对。这个表的主键是 (p, a)。

Rule 3. (N-N联系) 当两个实体E和F参与一个多对多二元联系R时，在相关的关系数据库设计中，联系映射成一个表T。这个表包括从实体E和F转化而来的两个表的主键的所有属性，这些列构成了表T的主键。T还包含了连接到联系的所有属性的列。联系实例用表的行表示，相关联系的实体实例可以通过这些行的主键值唯一地标识出。候选键是双键

Rule 4. (N-1联系) 当两个实体E和F参与一个多对一的二元联系R时，这个联系在关系数据库设计中不能被映射自身的一个表。相反，如果我们假设实体F具有 $\max\text{-card}(F, R) = 1$ ，并表示联系中的“多”方，那么从实体F转化成的关系表T中应当包括从实体E转换出的关系表的主键属性列，这被称为T的外键。因为 $\max\text{-card}(F, R) = 1$ ，T的每一行都通过一个外键值联系到实体E的一个实例，如果F在R中是强制参与的，那么它必须恰恰与E的一个实例相联系，这意味着T的上述外键不能取空值。如果F在R中是选择参与的，那么T中不与E的实例相联系的行在外键所有列可以取空值。

外键：如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。由此可见，外键表示了两个关系之间的相关联系

Rule 5. (1-1联系，可选参与) 给定两个实体E和F，他们参与一对一二元联系R，二者的参与都是可选的。首先按照转换规则1建立表S来表示实体E；同样建立表T表示实体F。然后向表T中添加一组列（表S的主键作为作为T的外键）。如果愿意，还可以在表S中加入一组外键列（表T中主键的列）。

假使有一方是强制参与的，应该对其添加外键。

Rule 6. (1-1联系，两边均强制参与) 对于一个两方都是强制参与的一对一联系，最好将两个实体对应的两个表合并成为一个表，这样可以避免使用外键。

5.

- a) 用矩形代表实体
- b) 简单的属性用椭圆形表示，并用直线连接到实体
- c) 复合属性同样用椭圆形表示，并用直线连接到实体，组成复合属性的简单属性连接到复合属性上
- d) 多值属性用双线连接到实体
- e) 主标识符属性加下划线表示
- f) 关系用菱形表示
- g) 关系也会有属性，也用椭圆形表示，直线连接

6.弱实体：如果一个实体的所有实例都通过一个联系依赖另一个实体，那么就称之为弱实体（如员工家属，订单条目）。该实体的主键必须包含另一个实体的主键。

在E-R模型设计中，如果一个实体的存在必须依赖于其他的实体，则该实体集被称为弱实体

知识点14 规范化

1.规范化的目的与手段：降低数据冗余，消除操作（插入 / 删除）异常，增强数据库的稳定性和灵活性

2.函数依赖(Functional Dependency): $A \rightarrow B$: A函数决定B, 即A的所有值在B中有唯一的对应

3.Armstrong公理 用于通过已有FD推出其他一系列FD

基本规则:

Rule 1. **Inclusion Rule (自反/包含规则)** If $Y \subseteq X$, then $X \rightarrow Y$ 

平凡函数依赖: Y是X的子集的情况下, $X \rightarrow Y$

Rule 2. **Transitivity Rule (传递规则)** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Rule 3. **Augmentation rule (增广规则)** If $X \rightarrow Y$, then $XZ \rightarrow YZ$


部分函数依赖: W是X的子集, $W \rightarrow Y$, 则 $X \rightarrow Y$ 是部分函数依赖

扩充规则:

Rule 4. **Union Rule (合并规则)** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Rule 5. **Decomposition Rule (分解规则)** If $X \rightarrow YZ$, then $X \rightarrow Y$, and $X \rightarrow Z$

Rule 6. **Pseudotransitivity Rule (伪传递规则)** If $X \rightarrow Y$, and $WY \rightarrow Z$, then $XW \rightarrow Z$

Rule 7. **Set accumulation rule (聚积规则)** If $X \rightarrow YZ$ and $Z \rightarrow W$, then $X \rightarrow YZW$ 

4.函数依赖集F的闭包 (记为 F^+): 可以从F推导出得所有函数依赖的集合

5.函数依赖集的覆盖(FD set cover): G中的所有函数依赖都可以由F推导出来, 那么称F是G的覆盖, 或者F覆盖G, 即 $G \subseteq F^+$ 。

6.函数依赖集的等价: 如果F和G相互覆盖, 则称F等价于G

7.属性集闭包(Closure of a Set of Attributes): 属性集X能函数决定 (即推导出) 的最大属性集合Y, 称为属性集X的闭包

8.最小函数依赖集/最小覆盖(minimal cover) 

F的最小函数依赖集 / 最小覆盖M有以下性质: M覆盖F, 并且M中没有冗余的函数依赖, 并且每一个函数依赖的左边也没有多余的属性。

常见多余:


1. 平凡依赖
- 2.

最小函数依赖集算法:

Step 1:把函数依赖的右边全部先变成等价的单个属性 (即将 $A \rightarrow CD$ 拆成 $A \rightarrow C$ 和 $A \rightarrow D$)

Step 2: 即将多余的函数依赖删掉, 是否多余通过将其删掉后计算函数依赖集闭包判断 (即只要函数依赖集的闭包等价即是冗余的) 一般也不会真的算闭包, 能推出删掉的函数依赖就好了

Step 3: 对于左边有多个属性的函数依赖进行简化 (即尝试将形如 $AB \rightarrow D$), 方法是逐个去掉左边属性集中的属性 (如去除A), 每去掉一个都计算左边剩下的属性集闭包 (则计算B的属性集闭包), 只要该闭包等价于去除之前的闭包 (B的属性集的闭包) 即可以去除。

Step 4: 当对每一个函数依赖都进行了步骤3之后, 要再一次回溯步骤2 

Step5: 合并形如 $A \rightarrow B$ 和 $A \rightarrow C$ 的项, 写成 $A \rightarrow BC$

9.无损性分解(Lossless Decomposition): 表T被分解后, 将分解后的小表连接起来仍能恢复原始表的信息, 这应对原始表将来任何可能的内容都成立

一个表T会对应一个函数依赖集F, T的一个分解是表的集合 $\{T_1, T_2 \dots T_t\}$

无损性分解使得该集合有如下两个性质:

- (1) Head (T_i) 是Head (T) 的一个子集。
- (2) Head (T) = Head (T_1) \cup Head (T_2) $\cup \dots \cup$ Head(T_i)

与之对应的是有损分解 (Lossy Decomposition): 表T被分解后, 将分解后的小表连接起来不能恢复原始表的信息, 多了一些以前不存在的行

* 分解是否无损的判定:

若表T被分解为 T_1 和 T_2 ，则以下FD至少有一个成立时，分解一定是无损的：

$\text{Head}(T_1) \cap \text{Head}(T_2) \rightarrow \text{Head}(T_1)$

$\text{Head}(T_1) \cap \text{Head}(T_2) \rightarrow \text{Head}(T_2)$

通俗地讲，找出两个表的公共属性，若它成为某一张表的关键字，则为无损分解

10. 依赖保持性(FD Preserved)

若表T被无损分解为 $\{T_1, T_2, \dots, T_k\}$ ，对F中的函数依赖 $X \rightarrow Y$ ，有 $X \cup Y \subseteq \text{Head}(T_i)$ ，则说 $X \rightarrow Y$ 在此分解中具有依赖保持性或 $X \rightarrow Y$ 被保持在 T_i 上。如果有F的函数依赖集的闭包等于 $(F_1 \cup F_2 \cup \dots \cup F_k)$ 的闭包，则称该分解具有依赖保持性。



主关系键：非空且唯一

*寻找候选关键字



先对所有属性进行分类 (1) 只在右边 (2) 只在左边 (3) 两边都有

(2) 一定在关键字中，(3) 可能在关键字中，因此只要对(2)和(3)进行组合来产生不同的候选关键字。注意要保证关键字的真子集不是关键字（无冗余属性）。

11. 主属性：任何一个候选关键字中所包含的属性都是主属性。剩下的是非主属性



不一定是主键内的属性

12. 2NF范式：在F的闭包中找到任何一个 $X \rightarrow A$ (A是单个的非主属性)，若X不包含A属性，则X不能是某个候选关键字的真子集。

3NF范式（蕴含了2NF，要求更严格）：在F的闭包中找到任何一个 $X \rightarrow A$ (A是单个的非主属性)，若X不包含A属性，则X是一个超键。

BCNF范式（蕴含了3NF，覆盖更广，要求更严格）：在F的闭包中找到任何一个 $X \rightarrow A$ (A是单个属性)，若X不包含A属性，则X是一个超键。

如果关系R上的最小函数依赖集为空，那么该关系一定满足BCNF（对）

如果关系R上的非主属性集为空，那该关系一定满足BCNF（错）

2NF：消除了非主属性对码的部分依赖 —— 优化修改异常

3NF：消除了非主属性对码的传递依赖 —— 优化删除/插入异常

* 是否满足BCNF? BCNF：消除了主属性对码的部分依赖 —— 进一步优化三异常（解决具有反复性主属性）

对每一个分解，判断依赖的左边是不是都可以唯一确定这个关系中的每一行

是的话就满足BCNF；不是的话，就取出这个依赖右边的属性，将其从这个关系中剔除产生新分解的子表1，将这个依赖的左右两边合起来作为另一张子表

第七章：数据定义

知识点15 完整性约束

1. 基本的建表命令

```
CREATE TABLE customers (  
    cid char(4) not null,  
    cname varchar(13),  
    city varchar(20),  
    discnt real,  
    primary key(cid));
```

2. 一般约束

模式名：默认模式名为建表的用户名。

3.DEFAULT语句: DEFAULT { default_constant | NULL }

例: dollars float default 0.0

4.列约束: 在列的声明处使用

```
{ NOT NULL |  
  [ CONSTRAINT constraint_name ]  
  UNIQUE  
  | PRIMARY KEY  
  | CHECK ( search_condition )  
  | REFERENCES table_name [ ( column_name ) ]  
  [ ON DELETE CASCADE | RESTRICT | SET NULL ]  
  [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

1. **NOT NULL** (不能与DEFAULT NULL同时使用, 当DEFAULT语句和NOT NULL语句都不出现时, 默认为DEFAULT NULL) ;
2. **CONSTRAINT** (可用于给每个约束取名, 方便删除该约束) ;
3. **UNIQUE** (不允许出现重复元组, 允许在存在NULL值的时候使用, 表示所有非NULL值都唯一) ;
4. **PRIMARY KEY** (声明为主键, 包含NOT NULL和UNIQUE的意思, 包括表约束的PRIMARY在内, 一个表中只能有一个PRIMARY KEY语句, 不能与UNIQUE语句一起使用) ;
5. **CHECK** (要求该列必须满足括号中的条件) ;
6. **REFERENCES** (REFERENCES的值只能是NULL或指向表中的值, 默认指向单列的PRIMARY KEY) ;
7. **ON DELETE CASCADE|RESTRICT|SET NULL** (可选, 只能跟在REFERENCES后面使用, 表示当被指向的行被删除时要进行的操作, 默认为NO ACTION, 若不设置则删除将失败, CASCADE表示删除时将其从引用表中删除, RESTRICT表示不允许删除, SET NULL表示删除后将引用值改成NULL) 。

5.表约束: 在列声明结束之后使用

```
[ CONSTRAINT constraint_name ]  
  { UNIQUE ( colname { , colname ... } )  
  | PRIMARY KEY ( colname { , colname ... } )  
  | CHECK ( search_condition )  
  | FOREIGN KEY ( colname { , colname ... } )  
  REFERENCES table_name [ ( colname { , colname ... } ) ]  
  [ ON DELETE CASCADE | RESTRICT | SET NULL ]  
  [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

1. **UNIQUE** (单列时同列约束, 多列时表示多列的组合必须唯一) ;
2. **PRIMARY KEY** (可以是多列, 但包括列约束的PRIMARY KEY在内, 在一个表中最多有一个PRIMARY KEY语句) ;
3. **CHECK** (大多数数据库不支持在CHECK语句内的子查询) ;
4. **FOREIGN KEY ... REFERENCES ...** (声明表中某列为指向另一个表的某列的引用, 若被指向的表声明了PRIMARY KEY, REFERENCES语句可以不指定列名) ;
5. **ON DELETE CASCADE** (可选, 同列约束, 对整张表的引用有效) 。

6.引用约束:

the columns of F in any row of T1 must either have null values in at least one column that allows

null values, **or** have no null values and be equal to the value combination of P on some row of T2.

7. 外键约束

* 对于REFERENCE进行ON DELETE / ON UPDATE

[CASCADE | RESTRICT | SET NULL]

比如: **CREATE TABLE Emp (**

ssn char(8) PRIMARY KEY,

dno char(4)

REFERENCES Dept ON DELETE RESTRICT,}

(restrict表示若该部门有职工, 则不允许进行删除)

REFERENCES Dept ON DELETE CASCADE,}

(cascade表示若该部门有职工, 则将该部门的所有职工也删除)

REFERENCES Dept ON DELETE SET NULL,}

(cascade表示若该部门有职工, 则将该部门的职工的部门置为NULL)

外键约束是为了限制

对被引用表的修改

对引用表的修改无论如何都满足引用完整性

* **修改表** (加入或者删除列, 加入或者删除约束)

ALTER TABLE tablename

[Add] [Drop] [Modify] 对于列进行操作

[Add] [Drop] 对于约束进行操作

可以根据行、列级改动触发

REFERENCING OLD|NEW AS name

begin end块内可以用 :name 调用

8. 触发器

CREATE TRIGGER trigger_name { BEFORE | AFTER }

//缺省为AFTER。BEFORE表示先执行更新, 再触发。AFTER反之

{ INSERT | DELETE | UPDATE [**OF** colname { , colname ... }] }

ON table_name

[FOR EACH ROW | FOR EACH STATEMENT] --默认是STATEMENT

[WHEN (search_condition)] //不满足when条件会进入begin end

{ statement | BEGIN ATOMIC statement; { ... } END;

知识点16 视图 (view)

1. 定义: 是子查询产生的表, **但有自己的名字**

2. 特点: (1) 不拥有数据, 是**动态执行子查询**所得的结果

(2) 被称为是虚拟的表

3. 视图的定义命令

CREATE VIEW view_name [(col_name { , col_name ... })]

AS subquery [WITH CHECK OPTION]

具体指, 对子查询不可见的更新

当子查询中返回的列**中没有名冲突**时可以省略视图名后的列名。

添加了检查选项 (CHECK) 后, 将不允许向视图添加不符合子查询要求的行。

1. 列名: 当子查询列名不冲突时, 可以不用重新定义列名; 冲突时必须重新定义; 可以给列名任意重命名

2. 子查询: **不可以使用order by语句**

[With check option], 规定通过View的insert Update语句操作改变基本表的内容, 如果导致对视图的不可见的行, 它们将不被允许

3. With check option

WITHCHECKOPTION的作用?

CREATE VIEW custs AS

SELECT *

FROM customers

WHERE discnt <= 15.0

1. 对于update, 有withcheckoption, 要保证update后, 数据要被视图查询出来。

2. 对于delete, 有无withcheckoption都一个羊;

4. 对于insert, 有withcheckoption, 要保证insert后, 数据要被视图查询出来。

5. 对于没有where子句的视图, 使用withcheckoption是多余的。

WITH CHECK OPTION;

对视图的操作将作用于原表。

视图可以嵌套使用（子查询从视图中查询）。

4.视图的查询命令类似于普通的查询操作

5.删除表/视图：

```
DROP { TABLE table_name | VIEW view_name }  
    { CASCADE | RESTRICT };
```

视图被删除时不会删除任何行。

CASCADE表示所有使用到本视图的外键约束和视图都同时被删除；

RESTRICT表示在还有引用存在时删除将失败。

6.视图的作用

视图能够简化用户的操作

视图使用户能以多种角度看待同一数据

视图对重构数据库提供了一定程度的逻辑独立性

视图能够对机密数据提供安全保护

知识点17 安全性

1.授权语句

```
GRANT { ALL PRIVILEGES | privilege { , privilege ... } }  
    ON [ TABLE ] tablename | viewname  
    TO { PUBLIC | user-name { , user-name ... } }  
    [ WITH GRANT OPTION ];
```

privilege包括五种：

SELECT, DELETE, INSERT

UPDATE [column_name { , column_name ... }]

REFERENCES [column_name { , column_name ... }]

例如：

```
grant select, delete, insert, update(cname, city)  
on custview to eoneil;
```

2.权限回收语句

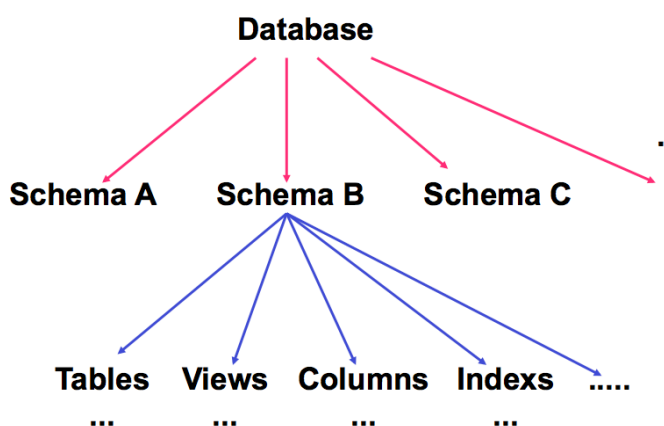
```
REVOKE { ALL PRIVILEGES | privilege { , privilege ... } }  
    ON tablename | viewname  
    FROM { PUBLIC | user-name { , user-name ... } }  
    { CASCADE | RESTRICT };
```

CASCADE表示删除与当前被撤销的权限有关系的视图或删除依赖于REFERENCES权限的

更新需要用户在基本表上有权限

可更新要求：

1. The FROM clause of the Subquery must contain only a single table, and if that table is a view table it must also be an updatable view table.
2. Neither the GROUP BY nor HAVING clause is present.
3. The DISTINCT keyword is not specified.
4. The WHERE clause does not contain a Subquery that references any table in the FROM clause, directly or indirectly via views.
5. All result columns of the Subquery are simple column names: no expressions, no column name appears more than once.



FOREIGN KEY约束；

RESTRICT表示如果有这样的依赖关系，不允许执行REVOKE语句。

3.Schema是模式，是数据库对象的集合，一个用户一般对应一个schema。可以用来区分数据库的同名对象。

知识点18 索引

1.提高select查询速度最常用的方法是创建索引

2.在关系数据库系统中，可以通过建立唯一键索引和非空约束来实现关键字的定义

第十章：事务管理

知识点19 事务(transaction)

1.是数据库提供的，把一系列数据库操作组合在一起作为一个整体执行。

2.ACID特性

Atomicity (原子性)：事务包含的一组更新操作时原子不可分的，要么全部成功完成，要么全部失败，即使是在系统崩溃的时候。

Consistency (一致性)：事务的成功完成将数据库从一个一致状态转变到另一个一致状态，它要求在事务并发执行的情况下事务的一致性仍然是满足的。

Isolation (隔离性)：又叫可串行性。系统允许的任何交错操作调度等价于某个串行调度。

Durability (持久性)：当事务发出提交语句后系统返回到程序逻辑时，它对数据所做的修改将是持久的，无论发生何种机器和系统故障。

对应的常见错误：

1. 创建不一致结果：
2. 线程不安全
3. 不确定持久化时机

3.事务的启动：

没有begin transaction这样的语句，事实上当系统处理进程中没有活动的事务，一个事务就可以开始。当事务进行中时任何更新对用户而言都是不可见的。

4.事务的结束：

Exec sql commit work; 事务已完成

Exec sql rollback work ; 异常终止

5.合适(well formed)事务：如果一个事务在访问数据库中的数据对象A之前按照要求申请对A的封锁，在操作结束后释放A上的封锁，这种事务被称为合适事务。

合适事务是保证并发事务的正确执行的基本条件

6.串行调度(seriaschedule)：所有事务的操作都是被依次调度执行的，不与其他任何事务交叉。

7.可串行化调度(Serial Schedule)：如果一系列操作等价于一个串行调度，则说它使可串行化调度。

8.调度器Scheduler：把来自各个事务的交错的访问操作输出一个可串行化的操作序列。

(实现方法比如推迟或者撤销某些操作)

同一个事务的访问操作，前后顺序确定
不同事务的访问操作，前后顺序不确定

9.冲突操作：两个事务的操作是冲突操作当且仅当



- 两个操作的对象是相同；
- 两个操作由不同的事务完成；

- 其中至少有一个是写操作。

10.冲突操作有两种类型：

(1) W-R丢失更新问题：也称之为“脏”写。每个事务都是先对一个数据进行读操作，之后再继续进行写操作。却没有意识到另一个事务也在进行相同的操作。

(2) W-W 盲写问题：每个事务都是直接写而不读

11.同一个事务所拥有的锁不会冲突，特别是在一个数据项上拥有读锁的事务可以进一步申请写锁，只要该数据项上没有其他读锁的存在

12.冲突的可串行性定理（存在冲突操作未必不可以串行化）

一个经历H存在等价的串行经历S(H)当且仅当对应的前趋图PG(H)中不存在环。

知识点20 封锁

1.封锁的前提：访问数据库的数据之前必须先获得锁

2.排他锁（eXclusive lock，简称X锁），即写封锁（WL – Write Lock）

申请规则：该数据对象没有被其他事务封锁

作用：对数据对象进行读写，并且禁止其他事务对于该数据对象的数据访问。

如果对象被施加了X锁，则其他事务都不能再施加任何类型的锁

锁必须维持到事务T的执行结束

3.共享锁（Sharing lock，简称S锁），即读封锁（RL – Read Lock）。

申请规则：该数据对象没有被封锁或者只有S锁

作用：进行读操作。允许多个事务的操作并发执行。

- 如果数据对象A没有被其它事务封锁，或者其它事务仅仅以 S锁的方式来封锁数据对象A时，事务T才能在数据对象A上施加 S锁
- 在持有封锁的事务释放数据对象A上的所有S锁之前，任何事务都不能对对象A写数据
- S锁不必维持到事务T的执行结束(依封锁协议而定)

4.排他锁与共享锁的锁相容矩阵：

		其它事务已持有的锁		
		X锁	S锁	—
当前事务申请的锁	X锁	No	No	Yes
	S锁	No	Yes	Yes

锁相容矩阵

5.加锁定理：遵循2PL原则的事务型操作必定是可串行化的

6.两阶段封锁(Two-Phase Locking, 2PL)

两段锁原理保证了一个可串行化的经历

- 锁申请规则：当事务 T_i 试图读取某个数据项时，需要 $RL_i(A)$ 。当事务 T_i 试图对某个数据项进行写操作时，需要加写锁， $WL_i(A)$ 。

- 锁申请的等待规则：得到锁的前提是没有冲突的锁在该数据上。
- 单个事务的锁申请和释放规则。加锁有两个阶段：增长阶段，在该阶段事务获得所要求的锁；收缩阶段，该阶段事务释放拥有的锁。调度器必须保证一旦收缩阶段开始，不允许该事务再申请新的锁，也就是说不允许一个事务释放了某个锁之后又去申请别的锁。

7.短期锁：（可以提高并发度）：读完或写完之后立即释放锁。

长期锁：事务提交后再释放锁。

8.死锁：是指两个或者两个以上的事务在进行操作时，由于资源竞争或彼此通信而造成的永久等待的现象。 一般是因为Synchronized的过分权责，即知道Commit 收缩才能释放锁，导致锁的无故占用

知识点21 事务的隔离级别（决定了该事务的封锁策略）

1.
 - a) Readundocommitted未提交读
只做读操作并且不用申请锁，不允许执行写操作。
在该方式下，当前事务不需要申请任何类型的封锁，因而可能会‘读’到未提交的修改结果
禁止一个事务以该方式去执行对数据的‘写’操作，以避免‘写’冲突现象。
 - b) Readcommitted提交读
读之前申请共享锁，读之后立即释放。
在‘读’数据对象A之前需要先申请对数据对象A的‘共享性’封锁，在‘读’操作执行结束之后立即释放该封锁。以避免读取未提交的修改结果。
 - c) Readrepeatable可重复读
读之前申请共享锁，并且直到事务结束再释放。在‘读’数据对象A之前需要先申请对数据对象A的‘共享性’封锁，并将该封锁维持到当前事务的结束。
可以避免其它的并发事务对当前事务正在使用的数据对象的修改。
 - d) Serializable可序列化（可串行化）
并发事务以一种可串行化的调度策略实现其并发执行，以避免它们相互之间的干扰现象。
- 2.不管采用何种隔离级别，在事务‘写’数据对象A之前需要先申请对数据对象A的‘排它性’封锁(write-lock)，并将该封锁维持到当前事务的结束。
- 3.隔离级别与封锁协议的关系


	Write locks	Read Locks (row)	Read Locks (Predicates)	Problem
Read Uncommitted	No (Read Only)	No	No	Dirty Reads
Read Committed	Yes	short-term	short-term	Lost Update
Repeatable Read	Yes	long-term	short-term	Update Anomaly
Serializable	Yes	long-term	long-term	No

因为没有限制整个表的插入和删除，可能在某些语句前后两次发现不存在的行或多出的行

知识点22 日志(log)

1.UNDO日志:

格式如下:

- a) 开始一个事务: <Start T>
- b) 提交事务T: <Commit T>
- c) 放弃事务T: <Abort T>
- d) 更新记录: <T, X, V> 

缺陷:

- 1. 事物的提交过程中需要执行许多写磁盘操作, 开销大

在数据库管理系统, 提交事务T并确保其更新结果的持久化实现的标志是: 将缓冲区中的<commit T>日志记录写入日志文件的磁盘

内容: 记录的是数据修改之前的值, 也就是before image

用于未结束事务的撤消(UNDO)

记载规则: a) 数据修改: 应该先更新磁盘日记磁盘

(把<T,V,W> flush log 到日记磁盘上)

b) 事务T提交操作, 则日志记录<Commit T>必须在事务T改变的所有DB元素已写到磁盘后再写到磁盘。

2.REDO日志: 纪录的是更新后的值 比UNDO的好处是output可以延后

记载规则:

- a) 更新记录<T,X,V> 和提交记录 <Commit T>都应该先被写到日记磁盘然后再进行相关的数据库磁盘操作。
- b) 记录的是修改项修改后的值, 用于已提交事务的重做(REDO)

3.UNDO/REDO日志:

记载规则:

a)数据修改: 必须先把更新<T,X,v,w>写到日记磁盘再更新数据库磁盘

b)提交: 每一条 <Commit T> 后面必须紧跟一条Flush Log操作。

既可用于事务的撤销, 又可用于事务的重做

4.使用日志进行数据库恢复的过程:

先逆向扫描日志, 对未提交事务做UNDO处理 (同时记录下已提交事务)

再正向扫描日志, 对已提交事务做REDO处理

5.在数据库系统中, 提交事务T并确保其更新结果的持久化实现的标志是:

将缓冲区中的<commit T>日志记录写入日志文件的磁盘

知识点23 检查点(checkpoint)

1.使得回滚操作只需要执行到检查点即可, 不需要回滚整个日志文件。

2.可以通过在数据库日志中设置检查点来减少故障恢复过程中需要扫描处理的日志范围

3.可以通过在事务中设置保存点来提供对事务的部分回滚功能

知识点18 索引 Index 补充 (迷)

1. Each card (entry) has: (keyvalue, row-pointer) 【数据对硬盘的一对一，表驱动】
keyvalue is for lookup
row-pointer (ROWID) is enough to locate row on disk (one I/O)

索引键(keyvalue): 由指定列直接拼接生产 (可能不唯一)

RowID: 根据磁盘存储页数+槽位生产的唯一磁盘标识

索引 (索引键, RowID) 的 (一般有序) 键值对数据结构

2. 索引创建

```
CREATE [ UNIQUE ] INDEX indexname
ON tablename (colname [ASC | DESC]
{ , colname [ASC | DESC] ...});

DROP INDEX indexname;
```

DEFAULT 对应数据表中多行

UNIQUE 一个INDEX 值对应数据表中的一行 (你可以用UNIQUE INDEX+NOTNULL代替主键)

数据表的修改会自动修改硬盘上对应的INDEX值, 不用重新构建

3. 数据库存储结构:

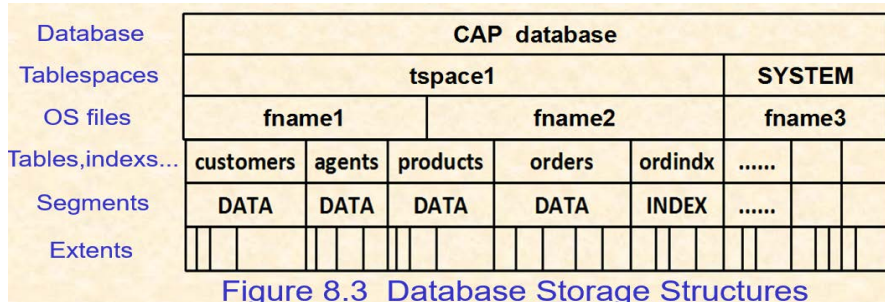


Figure 8.3 Database Storage Structures

(为什么不用RowID代替主键, 因为可能变而且有系统适配问题)

4. BitMap: 用于解决多索引重复问题 (对每一个属性的特定值做BITMAP)

A bitmap takes the place of a ROWID list, specifying a set of rows

one bit vs a ROWID (row)

1: the row have the keyval

0: the row don't have the keyval

a. 有N个行的表中每一行都有一个原始编号(0,1...N-1)

b. 且必然有一个函数能进行原始编号—>ROWID的转换

c. Bitmap (位图) 是一个有N位的序列, 当原始编号k的行拥有指定keyval, 在k位上置位1

5. 数据结构实现:

6. 1. 内部索引存成B树

7. 2. hash聚簇 (更快但不适用于order by 等范围搜索)

为什么无论是读请求还是写请求, 哈希类型的索引都比树型索引要更快一点, 树型还能成为主流呢?

因为对单行查询确实如此, 但是对于排序查询 (group by order by < >) 的SQL需求, 哈希的索引会退化回n²

8. 聚族索引

聚族索引就是存放的物理顺序和列中的顺序一样, 一般设置主键索引为聚集索引