《数据库系统及应用》复习提纲

made by cmx

第一章 数据库管理概述

1、数据库、数据库管理系统和数据库系统的概念

数据库（Database,简称DB）：长期储存在计算机内、有组织的、可共享的大量数据的集合

数据库管理系统DBMS（Database Management System）：是计算机程序的集合，用于创建和维护数据库

数据库系统DBS（DataBase System）指在计算机系统中引入了数据库后的系统，即采用了数据库技术的计算机系统

2、数据库系统与文件系统的对比

文件系统：数据冗余和不一致、数据孤立和访问困难、完整性问题、原子性问题、数据并发操作问题、安全性问题

数据库系统：数据共享、减少冗余、避免不一致、提供事务支持、保持完整性、增强安全性、提供并发控制、标准化

3、DBMS的分类

- 按所支持的用户数：单用户DBMS（目前已经很少见）、多用户DBMS
- 按允许数据库可以分布的站点数：集中式DBMS、分布式DBMS
- 按用途：通用DBMS，如Oracle、Informix等；专用DBMS，如时态数据库、空间数据库等

第二章 数据库系统体系结构

1、数据库模式和实例

模式是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及类型的描述，不涉及具体的值。

模式的一个具体值称为模式的一个实例。

一个模式可有很多实例：
模式——反映数据的结构及联系
实例——反映的是某一时刻数据库的状态

模式相对稳定，而实例相对变动。

2、数据库的三级模式结构和两级映象的含义是什么？

三级模式结构：

外模式（子模式、用户模式）：单个用户所看到的局部数据的逻辑结构和特征的描述。

概念模式（模式、逻辑模式）：数据库中全体数据的逻辑结构和特征的描述。数据记录由哪些数据项构成；数据项的名字、类型、取值范围；数据之间的联系、数据的完整性等

内模式（存储模式）：数据物理结构和存储方式的描述。记录的存储方式：顺序存储、按B树组织还是散列存储？索引按什么方式组织？（排序or散列）数据是否加密？是否压缩存储？

二级映象：

外模式/模式映象：定义了外模式与概念模式之间的对应关系：属性名称可能不同；外模式的属性可能由模式中的多个属性运算而得。

模式/内模式映象：定义了概念模式与内模式之间的对应关系；概念模式中的逻辑记录和字段在内部如何存储

3、数据独立性包括哪两个方面？各自的含义是什么？

当概念模式发生改变时，只要修改外模式/模式映象，可保持外模式不变，从而保持用户应用程序不变，保证了数据与用户程序的逻辑独立性——数据的逻辑独立性

当数据库的内部存储结构发生改变时，只要修改模式/内模式映象，可保持概念模式不变，从而保持外模式以及用户程序的不变，保证了数据与程序的物理独立性——数据的物理独立性

第三章 关系数据模型

1、数据模型的概念和分类

概念：数据模型是描述现实世界实体、实体间联系以及数据语义和一致性约束的模型。

分类：根据模型应用的不同目的

- 概念数据模型（概念模型）按用户的观点对数据进行建模，强调语义表达功能；独立于计算机系统和DBMS；主要用于数据库的概念设计
- 结构数据模型（数据模型）按计算机系统的观点对数据进行建模，直接面向数据库的逻辑结构；与计算机系统和DBMS相关

2、数据模型的三个组成部分分别是什么？含义各是什么？

数据结构：现实世界实体及实体间联系的表示和实现

数据操作：数据检索和更新的实现

数据的完整性约束：数据及数据间联系应具有的制约和依赖规则

3、关系模型、元组、域、关系、关系模式、键等关系模型的基本概念

关系模型：用二维表格结构表示实体集，外码表示实体间联系，三类完整性规则，表示数据约束的数据模型

属性(Attribute)：二维表格的每一列称为关系的一个属性，列的数目称为度（degree）

元组(Tuple)：每一行称为关系的一个元组，元组的数目称为势或者基数（cardinality）

域(Domain)：一组具有相同数据类型的值的集合。每个属性有一个域

关系(Relation)：元组的集合

关系模式(Relation Schema)：关系的逻辑结构和特征的描述，对应于二维表格的表头，通常由属性集和各属性域表示，不关心域时可省略域：Student(Name, Age, Class)。

4、关系的性质有哪几个?

属性值不可分解：不允许表中有表

元组不可重复：因此一个关系模式至少存在一个候选码

没有行序，即元组之间无序：关系是元组的集合

没有列序，即属性之间无序：关系模式是属性的集合

5、关系模型中有三类完整性规则，分别指什么？具体含义?

实体完整性(Entity Integrity)：关系模式R的主码不可为空。

参照完整性(Referential Integrity)：参照关系R的任一个外码值必须等于被参照关系S中所参照的候选码的某个值，或者为空。

用户自定义完整性(User-Defined Integrity)：针对某一具体数据的约束条件，反映某一具体应用所涉及的数据必须满足的特殊语义。

6、关系模型的形式化定义

R（U，D，dom，F)
R为关系模式名，U是一个属性集，D是U中属性的值所来自的域，Dom是属性向域的映射集合，F是属性间的依赖关系。

7、关系代数的概念？关系代数中一元操作和二元操作各有哪些?

Relational Algebra=<A, O>
A：关系，关系代数中只存在一种类型，即关系
O：关系代数运算，也称关系代数操作，以关系为运算对象的一组运算集合

满足封闭性：任何关系代数运算的结果仍是关系

一元操作（Unary Operation）只有一个变元的代数操作，如选择 σ、投影 π

二元操作（Binary Operation）具有两个变元的代数操作，如并、交、差、笛卡儿积、连接 $\bowtie$、除

8、关系代数的五个基本操作

基本运算有5个：并、差、积、选择、投影。重命名*ρ

其它操作都可以通过这些基本操作来表示。

交：$R \cap S = R - (R - S)$

自然连接：$R \bowtie S = \pi_{X,R.Y,Z}(\sigma_{R.Y=S.Y}(R \times S))$

θ 连接：$R \bowtie_{A\theta B} S = \sigma_{A\theta B}(R \times S)$

除：$R \div S = \pi_X(R) - \pi_X(((\pi_X(R)) \times S) - R)$

9、掌握附加的关系代数操作：扩展投影、聚集和分组、排序、赋值。

重命名*：ρx(E)：将关系代数表达式E重命名为X；ρx(A1，A2，……，An)(E)：将关系代数表达式E重命名为X，并且各属性更名为A1，A2，……An

扩展投影：a+b→x 作为一个列表元素表示a和b属性的和，并被重命名为x。元素 c||d→e 表示连接串类型的属性c和d，并重命名为e。

聚集函数：SUM（汇总）COUNT（计数）AVG（求均值）MAX（求最大值）MIN（求最小值）

分组$\gamma_L(R)$：该组的分组属性值。该组中所有元组对列表L的属性聚集操作的结果。

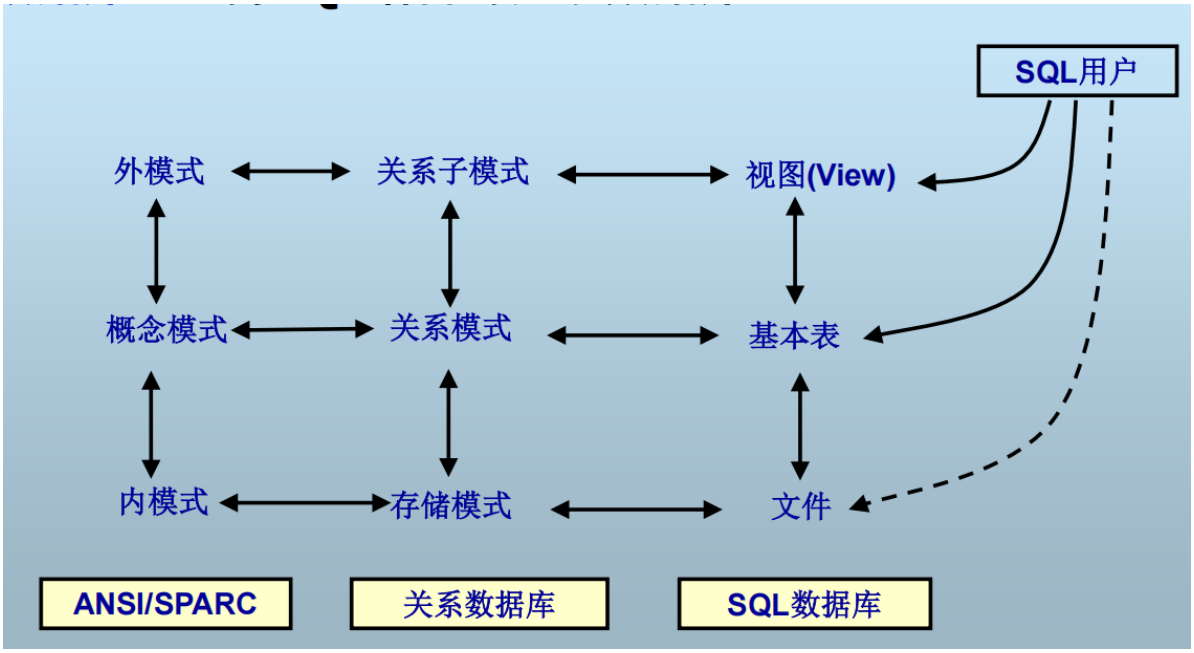
排序 $\tau_L(R)$：返回按属性列表L排序的关系R，结果中的所有元组是按照L排序。如果L是A1，A2，…，An，那么R 的元组就先按属性A1的值先排序，对于A1属性相等的元组，就按A2的值排序，依此类推。

赋值：R←E：把关系代数表达式E的结果赋给R。值操作并不把结果显示给用户，最后一句表示表达式 = 结果被作为结果显示。

10、熟练掌握：根据查询要求写出关系代数表达式

第四章 关系数据库语言SQL

1、SQL数据库的三级体系结构是什么?



2、SQL的组成

DDL（模式）、DML（数据）、DCL（权限）

3、熟练掌握：CREATE TABLE语句——DDL

```
Creat Table <基本表名>(
    列名1 列类型1 [DEFAULT <默认值>] [[NOT] NULL] [[Constraint <约束名>] <约束类型>],
    列名2 列类型2 [<列约束2>],
    ...
    [表约束]
);
```

约束类型：Primary Key 主键、Unique 唯一性、Foreign Key 外键、Check 检查

4、SQL中的约束有哪几种？与关系模型中的三类完整性约束有何关联？

主键约束（Primary Key）定义主码——实体完整性

唯一键约束（Unique）定义候选码

外键约束（Foreign Key）定义外码——参照完整性

检查约束（Check）——用户自定义完整性

5、熟练掌握：INSERT、UPDATE和DELETE语句的使用——DML

```
Insert Into <表名> (列名1，列名2，……，列名n)
Values（值1，值2，……，值n）
```

```
Update <表名>
Set <列名1>=<值1>，<列名2>=<值2>，……
[Where <条件>]
```

```
Delete From <表名>
[Where <条件>]
```

6、熟练掌握：根据要求写出SELECT查询语句（包括基本查询、连接查询和嵌套查询）

```
Select <列名表> ——指定希望查看的列
From <表名列表> ——指定要查询的表
Where <条件> ——指定查询条件
Group By <分组列名表> ——指定要分组的列
Having <条件> ——指定分组的条件
Order By <排序列名表> ——指定如何排序
```

基本查询：

```
# 查询全部记录
Select * From Student

# 查询特定的列
Select sno, sname From Student

# 使用别名
Select sno AS 学号, sname AS 姓名 From Student

# 使用表达式
Select concat(sno,':', sname) AS 学生，2003－age AS 出生年份 From Student
Select sno, format_date(birth, '%m-%d-%Y') AS birthday From Student
Select Count(sno) As 学生人数 From Student

# 检索特定的记录
Select sno AS 学号, sname AS 姓名 From Student Where age > 20
# WHERE子句中的关系运算符
# 算术比较符：>, <, >=, <=, =, <>
# IN
Select * From Student Where sno IN ('s001','s003','s006','s008')
# IS NULL和IS NOT NULL
Select * From Student Where age IS NULL
# LIKE: %：任意长度的字符串  _：单个字符
Select * From Student Where sname LIKE 'R%'
# EXISTS
# 多个比较式可用NOT、AND和OR连接
Select * From Student Where age IS NULL and sname LIKE 'R%'

# 去除重复记录
Select Distinct sname From Student

# 排序查询结果：ASC表示升序，DESC表示降序
Select * From Student Order By age ASC, sname DESC

# 聚集函数：
# Count(列名)：对一列中的值计数
# Count(*)：计算记录个数
# SUM(列名)：求一列值的总和（数值）
# AVG (列名)：求一列值的平均值
```

```
# MIN (列名)：求一列值的最小值
# MAX (列名)：求一列值的最大值
# 除聚集函数外的属性必须全部出现在Group By子句中
Select sex, AVG(age) as Average_age From Student Group By sex

# 返回满足特定条件的分组结果
Select age, COUNT(*) as students From Student
Group By age
Having COUNT(*) > 5

# limit
Select sno, avg(score) as avg_score
From SC
Group By sno
Order By avg_score DESC
Limit 4,6 # 从第5行开始，返回6行 offset, rows

# All, Some, Any
# All：要求子查询中的所有条件都满足
Select eno from employee
where dept='finance'
    and salary > ALL (select salary from employee where dept='sales')
# Some和Any：要求子查询中的某个条件满足即可
Select eno from employee
where dept='finance'
    and salary > SOME (select salary from employee where dept='sales')

# Outer Join 返回不匹配的结果

# If(条件表达式，true时的值，false时的值)
Select student.sno, If(count(cno) is null, 0, count(cno)) as c_count
From student LEFT OUTER JOIN SC on student.sno=sc.sno
Group by SC.sno
```

连接查询：

```
Select student.sno, student.sname,sc.cno
From student,sc
Where student.sno = sc.sno # 连接条件

# 表别名
Select b.cno, c.cname
From student a, sc b, course c
Where a.sno=b.sno and b.cno=c.cno and a.sname='sa'
```

嵌套查询：

```
# 无关子查询
Select sno,sname
From student
Where sno NOT IN (select distinct sno From sc)

# 相关子查询
Select sno, sname
From student
Where EXISTS (Select * From sc Where sc.sno= student.sno)

# 联机视图
```

查询结果的连接

```
# UNION
(Select sno From student where age<20)
UNION
(Select sno From
    (Select sno, AVG(score) From SC group by sno having avg(score)>90) SC2
)
# UNION操作自动去除重复记录 --Set Union
# Union All操作不去除重复记录 --Bag Union

# Minus 差
Select sno From Student) Minus (Select distinct sno From SC)

# Intersect 返回两个查询结果的交集
(Select sno From student where age<20)
Intersect
(Select sno From (Select sno, AVG(score) From SC group by sno having
avg(score)>90) SC2)
```

7、视图的概念？视图在SQL数据库中有什么作用？

概念：视图是从一个或几个基本表中导出的虚拟表，其数据没有实际存储，但可以和表一样操作。

作用：逻辑数据独立性：用户程序与数据库结构；简化了用户眼中的数据，使用户可以集中于所关心的数据上；同一数据库对不同用户提供不同的数据呈现方式；安全保护

8、视图的更新有何限制？

不是所有视图都是可更新的：基于连接查询的视图不可更新；使用了函数、表达式、Distinct的视图不可更新；使用了分组聚集操作的视图不可更新。

只有建立在单个表上，而且只是去掉了基本表的某些行和列，但保留了主键的视图才是可更新的。

第五章 过程化SQL

1、过程化SQL的主要作用

可以完成一些SQL不能完成的复杂计算，并且封装处理逻辑；

客户机计算任务少；

服务器计算任务加重；

网络传输少。

2、了解过程化SQL对SQL的主要扩展

*输入输出：输出：select，输入：NA

程序块定义：Begin……End

变量

流程控制：顺序结构/分支结构/循环结构

出错处理

游标

过程：存储过程/函数、触发器

3、游标、存储过程、触发器的概念和作用

游标：游标是客户机或数据库服务器上开辟的一块内存，用于存放SQL返回的结果。

游标可以协调过程化SQL与SQL之间的数据处理矛盾；

过程化SQL程序（存储过程/函数）可以通过游标来存取SQL返回的结果。

存储过程：存储在数据库中的过程，可以随时运行，也可以被SQL或外部程序调用。

作用：增强了SQL的功能和灵活性，可以完成复杂的判断和运算。

可增强数据库的安全性。通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。

可增强数据库的完整性。在运行存储过程前，数据库已对其进行了语法和句法分析，并给出了优化执行方案。 由于执行SQL语句的大部分工作已经完成，所以存储过程能以较快的速度执行。

可以降低网络的通信量。使体现企业规则的运算程序放入数据库服务器中，以便集中控制。

触发器：特定表关联的存储过程。当在该表上执行DML操作时，可以自动触发该存储过程执行相应的操作。

触发操作：Update、Insert、Delete。通过触发器可以定制数据库对应用程序的反应。

一个触发器只能属于一个表，一个表可有多个触发器。

作用：强化约束：触发器能够实现复杂的约束。

跟踪变化：触发器可以侦测数据库内的操作，可以用来实施审计，以及不允许数据库中未经许可的更新和变化。

级联运行：触发器可以侦测数据库内操作，并自动地级联影响整个数据库的各项内容。

第六章 关系数据库的模式设计

1、函数依赖、完全函数依赖、传递依赖、无损连接的概念

函数依赖（FD，Functional Dependency）：一个关系模式中一个属性集和另一个属性集间的多对一关系。

形式化定义：关系模式 R(A1,A2,...,An) 或简记为 R(U)，X 和 Y 是 U 的子集。r 是 R 的任意一个实例（关系），若 r 的任意两个元组 t1、t2，由 t1[X] = t2[X] 可导致 t1[Y] = t2[Y]，即如果 X 相等则 Y 也相等，则称 Y 函数依赖于 X 或称为 X 函数决定 Y，记作 X→Y。即 R 的X属性集上的值可唯一决定 R 的 Y 属性集上的值，也即对于 R 的任意两个元组，X上的值相等，则 Y 上的值也必相等。

FD 是相对于关系模式而言的，因此关系模式R的所有实例都要满足FD，FD是关系模式的一部分。

平凡 FD 和不平凡 FD：X→Y，且 Y ⊆ X，则 X→Y 是平凡 FD，否则是不平凡 FD。

函数依赖的逻辑蕴含：设 F 是关系模式 R 的一个函数依赖集，X 和 Y 是 R 的属性子集，若从 F 的函数依赖中能推出 X→Y，则称 F 逻辑蕴含 X→Y，记作$F \models X \to Y$
函数依赖集的闭包：被函数依赖集F逻辑蕴含的函数依赖的全体构成的集合称为F的闭包，记做F+。

完全函数依赖：对于函数依赖 W→A，若不存在 X⊂W，并且 X→A 成立，则称 W→A 为完全函数依赖，否则为局部函数依赖。

传递依赖：若Y→X，X→A，并且X→Y，A不是X的子集，则称A传递依赖于Y

无损连接：设R是关系模式，分解成关系模式 ρ = {R1,R2,...,Rk}，F 是 R 上的一个 FD 集，若对 R 中满足 F 的
每个关系 r，都有：$r = \pi_{R1}(r) \bowtie \pi_{R2}(r) \bowtie \ldots \bowtie \pi_{Rk}(r)$，则称这个分解p相对于F是“无损连接分解”。

记 $m_\rho(r) = \bowtie_{i=1}^{k} \pi_{Ri}(r)$，则对于关系模式R关于F的无损连接条件是 $r = m_\rho(r)$。

2、模式设计中可能出现哪些问题？如何解决？

数据冗余、更新异常、插入异常、删除异常

解决：模式分解。

3、模式分解的概念和含义

概念：设有关系模式 R（U）和 R1(U1)，R2(U2)，...，Rk(Uk)，其中U = U1 ∪ U2 ... ∪ Uk，设ρ = {R1, R2,..., Rk}，则称 ρ 为R的一个分解。

含义：属性集的分解；函数依赖集的分解。

4、模式分解的原则有哪几个？

既具有无损连接，又要保持函数依赖

5、掌握无损连接的测试方法

1. Chase

   造一个 k 行 n 列的表格，每行对应一个模式 Ri (1 ≤i ≤ k)，每列对应一个属性 Aj（1≤ j≤ n），若 Aj 在 Ri 中，则在表格的第 i 行第 j 列处填上 aj，否则填上符号 bij。

   检查 F 的每个 FD，并修改表格中的元素，方法如下：对于 F 中的函数依赖 X→Y，若表格中有两行在 X 分量上相等，在 Y 分量上不相等，则修改 Y：若 Y 的分量中有一个 aj，则另一个也修改为 aj；如果没有 aj，则用其中一个 bij 替换另一个符号（i 是所有 b 中最小的行数），一直到表格不能修改为止。

   若修改后，表格中有一行是全 a，即 a1a2...an，则 p 相对于 F 是无损连接的分解，否则不是。

2. p是无损连接的分解当且仅当下面之一满足：
   （R1 ∩ R2） → （R1 - R2）
   （R1 ∩ R2） → （R2 - R1）

6、掌握一个分解是否保持函数依赖的判定方法

$$\left(\bigcup_{i=1}^{k} \pi_{R_i}(F)\right)^{+} = F^{+}$$

7、掌握求最小函数依赖集的方法

当且仅当函数依赖集F满足下面条件，F是最小函数依赖集：
F的每个FD的右边只有一个属性
F不可约：F中的每个X→Y，F - {X→Y}与F不等价
F的每个FD的左部不可约：删除左边的任何一个属性都会使F转变为一个不等价于原来的F的集合

1-将右边写出单属性并去除重复FD（分解律）

2-消去左部冗余属性

3-消去冗余函数依赖

8、1NF、2NF、3NF、BCNF的概念

1NF：对于关系模式R的任一实例，其元组的每一个属性值都只含有一个值，则R∈1NF。

2NF：（假定 R 只有一个候选码/主码）当且仅当 R 属于 1NF，且 R 的每一个非主属性都完全函数依赖于主码时，R ∈2NF。

3NF：(假定 R 只有一个候选码，且该候选码为主码)当且仅当 R 属于 2NF，且 R 的每一个非主属性都不传递依赖于主码时，R∈3NF。

BCNF：如果关系模式R的所有不平凡的、完全的函数依赖的决定因素（左边的属性集）都是候选码，则R∈BCNF

9、码的形式化定义

关系模式 R(U)，F 是 R 的一个FD集，X 是 U 的一个子集，若 X→U ∈ F+，则 X 是 R 的一个超码；

如果同时不存在X 的真子集 Y，使得 Y→U 成立，则 X 是 R 的一个候选码。

10、掌握无损并且保持函数依赖分解到3NF的算法

保持函数依赖地分解到3NF的算法：

1. 求出 R<U,F> 的最小函数依赖集（仍记为F）
2. 把所有不在F中出现的属性组成一个关系模式R'，并在 U 中去掉这些属性(剩余属性仍记为U)
3. 若 F 中存在 X →A，且 XA = U, 则输出 R(U) 和 R'，算法结束，否则
4. 对 F 按相同的左部分组，将所有 X →A1, X →A2,..., X →Ak 形式的 FD 分为一组，并将每组涉及的所有属性作为一个关系模式输出。若某个关系模式 Ri 的属性集是另一个关系模式的属性集的子集，则在结果中去掉Ri。设最后得到关系模式R1，R2，...，Rk，则 p={R1,R2,...,Rk,R'} 一个保持函数依赖的分解，并且满足3NF。

无损连接且保持函数依赖地分解到3NF：

1. 先用算法1求出R的保持函数依赖的3NF分解，设为 q={R1,R2,...,Rk}
2. 设 X 是 R 的主码，求出p=q ∪ {R(X)}
3. 若 X 是 q 中某个 Ri 的子集，则在 p 中去掉 R(X)
4. 得到的 p 就是最终结果

11、掌握无损分解到BCNF的算法

1. p:={R};
2. 检查p中各关系模式是否都属于BCNF，若是，则算法终止
3. 设 p 中 S(Us) 非 BCNF 关系模式，则必存在 X→A ,其中 X 不是 S 的超码;
   ① 将S分解为S1(XA) 和 S2(Us - A)，此分解是无损联接的 // ({XA} ∩ {Us - A}=X)→(A={XA} - {Us - A}
   ② p:={p - S} ∪ {S1, S2}; // 用 S1 和 S2 替换 p 中的 S
   ③ 转到第2步;
4. 由于 U 的属性有限，因此有限次循环后算法终止

第七章 数据库设计

1、数据库设计分为几个阶段？各个阶段的主要工作是什么?

需求分析、概念设计、逻辑设计、物理设计、数据库实施、数据库运行与维护

2、概念设计一般采用什么方法？主要步骤是什么?

产生反映组织信息需求的数据库概念结构，即概念模型

概念模型独立于数据库逻辑结构、DBMS以及计算机系统

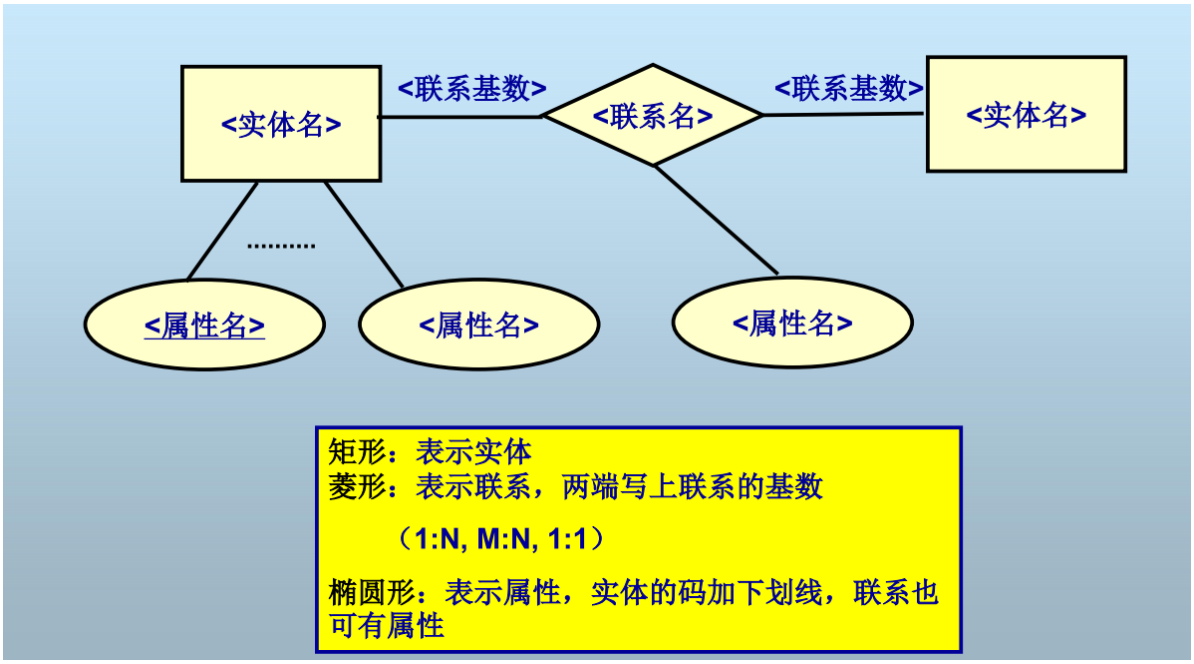
概念模型以一组ER图形式表示

概念设计侧重于数据内容的分析和抽象，以用户的观点描述应用中的实体以及实体间的联系

3、掌握ER设计的基本方法

自顶向下进行需求分析，自底向上进行ER设计：

- 分ER模型设计（局部ER图）：确定实体、确定实体属性、确定联系和联系属性（联系的基数）
- ER模型集成：
  - 确定公共实体
  - 合并分ER图
  - 消除冲突
- ER模型优化：
  - 合并实体类型：1:1合并，常同时处理合并。
  - 消除冗余属性：同一非码属性出现在几个实体中；一个属性值可从其它属性值中导出。
  - 消除冗余联系。

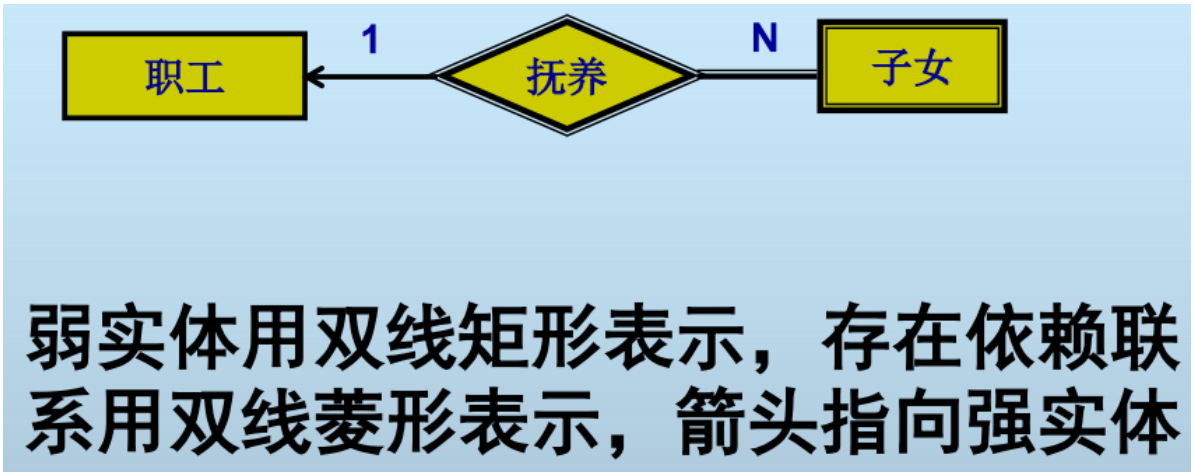


4、ER图的扩展：弱实体和子类的概念和设计

弱实体（weak entity）：一个弱实体的存在必须以另一实体的存在为前提。

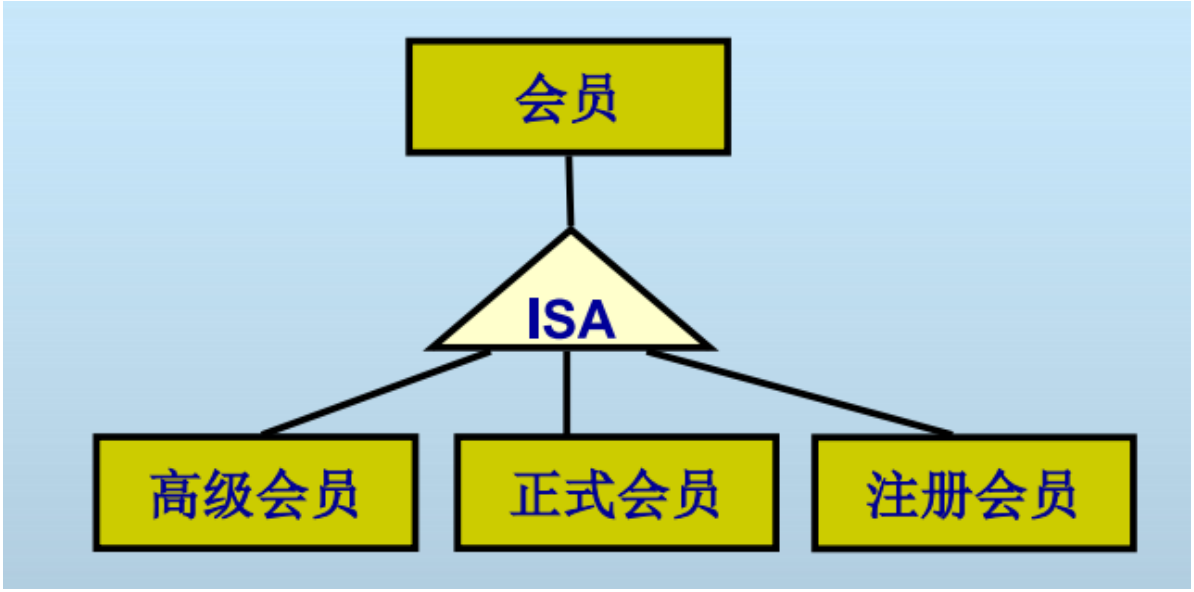
弱实体所依赖存在的实体称为常规实体（regular entity）或强实体（strong entity）

弱实体有自己的标识，但它的标识只保证对于所依赖的强实体而言是唯一的。在整个系统中没有自己唯一的实体标识。



子类（Subtype）和超类（Supertype）

- 两个实体A和B并不相同，但实体A属于实体B，则A称为实体子类，B称为实体超类
- 子类是超类的特殊化，超类是子类的一般化
- 子类继承了超类的全部属性，因此子类的标识就是超类的标识
    - 例如，研究生是学生的子类，经理是职工的子类
- 在ER设计时，可以根据实际情况增加子类，也可以根据若干实体抽象出超类



5、局部ER模式之间可能出现什么冲突?

属性冲突：类型冲突、值冲突，例如性别、年龄

结构冲突：实体属性集不同、联系类型不同、同一对象在不同应用中的抽象不同

命名冲突：同名异义、异名同义，实体命名冲突、属性命名冲突、联系命名冲突

6、逻辑设计的主要工作

ER模型向关系模型的转换、关系模型优化、关系模型修正。

7、掌握ER模型到关系模型的转换方法

实体转换

- 每个实体转换为一个关系模式，实体的属性为关系模式的属性，实体的标识成为关系模式的主码

联系转换

- 1:1：将任一端的实体的标识和联系属性加入另一实体所对应的关系模式中，两模式的主码保持不变
- 1:N：将1端实体的标识和联系属性加入N端实体所对应的关系模式中，两模式的主码不变
- M:N：新建一个关系模式，该模式的属性为两端实体的标识以及联系的属性，主码为两端关系模式的主码的组合

弱实体转换

- 每个强实体转换为一个关系模式，强实体的属性成为关系模式的属性，实体标识成为主码

- 每个弱实体转换为一个关系模式，并加入所依赖的强实体的标识，关系模式的主码为弱实体的标识加上强实体的标识

子类转换

- 父类实体和子类实体都各自转换为关系模式，并在子类关系模式中加入父类的主码，子类关系模式的主码设为父类的主码

8、物理设计的主要工作有哪些?

为关系模式选择存取方法、设计数据库的存储结构。

9、数据库实施阶段的主要任务?

- 建立实际的数据库结构
  - CREATE TABLE
  - CREATE INDEX
  - .....
- 初始数据装入
- 安全性设计和故障恢复设计
- 应用程序的编码和调试

第八章 数据库索引

1、密集索引、稀疏索引、多级索引、主索引、辅助索引的概念

密集索引：每个记录都有一个索引项，索引项按查找键排序。查找时：查找索引项，跟踪指针即可。

优点：记录通常比索引项要大；索引可以常驻内存；要查找键值为K的记录是否存在，不需要访问磁盘数据块。

缺点：索引占用太多空间

稀疏索引：仅部分记录有索引项，一般情况：为每个数据块的第一个记录建立索引。

优点：节省了索引空间；对同样的记录，稀疏索引可以使用更少的索引项。

缺点:对于“是否存在键值为K的记录？”，需要访问磁盘数据块。

多级索引：索引上再建索引。

好处：一级索引可能还太大而不能常驻内存；二级索引更小，可以常驻内存；减少磁盘I/O次数。当一级索引过大而二级索引可常驻内存时有效；二级索引仅可用稀疏索引；一般不考虑三级以上索引。

辅助索引：

主索引（Primary Index）：顺序文件上的索引，记录按索引属性值有序，根据索引值可以确定记录的位置。

辅助索引（Secondary Index）：数据文件不需要按查找键有序，根据索引值不能确定记录在文件中的顺序。辅助索引只能是密集索引。间接桶。

2、B+树的特点

一种树型的多级索引结构；树的层数与数据大小相关，通常为3层；所有结点格式相同：n个值，n + 1个指针；所有叶结点位于同一层。

B + 树插入：查找插入叶结点；若叶结点中有空闲位置（键），则插入；若没有空间，则分裂叶结点（叶结点的分裂可视作是父结点中插入一个子结点；递归向上分裂；分裂过程中需要对父结点中的键加以调整）例外：若根结点分裂，则需要创建一个新的根结点

B + 树删除：查找要删除的键值，并删除之；若结点的键值填充低于规定值，则调整（若相邻的叶结点中键填充高于规定值，则将其中一个键值移到该结点中；否则，合并该结点与相邻结点，合并可视作在父结点中删除一个子结点；递归向上删除）若删除的是叶结点中的最小键值，则需对父结点的键值加以调整。

B + 树的效率：访问索引的I/O代价 = 树高（B + 树不常驻内存）或者0（常驻内存）；树高通常不超过3层，因此索引I/O代价不超过3（总代价不超过4）；通常情况下，根节点常驻内存，因此索引I/O代价不超过2（总代价
不超过3）。

B树 vs. B+树：B-tree：所有节点都存储实际的数据（记录）；键值无重复存储；搜索有可能在非叶子结点结束；是数据存储的一种文件结构。B+-tree：节点本身不存储数据。

3、散列索引的特点

散列表(Hash Table)。

散列函数(Hash Functions)：h：查找键(散列键) → [0...B – 1]；桶(Buckets), numbered 0,1,..., B-1

散列索引方法：给定一个查找键K，对应的记录必定位于桶h(K)中；若一个桶中仅一块，则 I/O次数 = 1；否则由参数B决定，平均 = 总块数/B。

查找：对于给定的散列键值k，计算h(K)；根据h(K)定位桶；查找桶中的块。

插入：计算插入记录的h(K)，定位桶；若桶中有空间，则插入；否则，创建一个溢出块并将记录置于溢出块中。

删除：根据给定键值K计算h(K)，定位桶和记录，删除（回收溢出块）。

空间利用率：实际键值数 / 所有桶可放置的键值数。<50%：空间浪费；>80%：溢出问题；>50%到80%之间（GOOD！）。

4、动态散列索引：可扩展散列、线性散列

可扩展散列表：散列函数 h(k) 是一个 b (足够大)位二进制序列，前 i 位表示桶的数目；i 的值随数据文件的增长而增大。前 i 位构成一个桶数组。

优点：大部分情况下不存在着溢出块，因此当查找记录时，只需查找一个存储块。

缺点：桶增长速度快，可能会导致内存放不下整个桶数组，影响其他保存在主存中的数据，波动较大。

线性散列表：h(k)仍是二进制位序列，但使用右边(低) i 位区分桶；

桶数 = n，h(k) 的右 i 位 = m。若 m < n，则记录位于第 m 个桶；

若 $n \leq m < 2^i$，则记录位于第 $m - 2^{i-1}$ 个桶。

n 的选择：总是使n与当前记录总数 r 保持某个固定比例，意味着只有当桶的填充度达到超过某个比例后桶数才开始增长（否则将数据插入到溢出块中）。

总结：空间效率优于可扩展散列表；查找性能比可扩展散列表差；综合性能较好。

第九章 数据库应用开发（不考）

第十章 事务与恢复

1、事务的概念和ACID性质

事务(transaction)：一个不可分割的操作序列，其中的操作要么都做，要么都不做。

ACID性质：
原子性 Atomicity：事务是不可分的原子，其中的操作要么都做，要么都不做
一致性 Consistency：事务的执行保证数据库从一个一致状态转到另一个一致状态
隔离性 Isolation：多个事务一起执行时相互独立
持久性 Durability：事务一旦成功提交，就在数据库永久保存

2、SQL对事务的支持

3、事务的状态和原子操作

状态：< Start T>< Commit T>< Abort T>

原子操作：Input (x)、Output (x)、Read (x,t)、Write (x,t)

4、数据库一致性的概念

也称完整性约束，指数据库中的数据必须满足的谓词条件。事务开始前和结束后需保持一致，事物内部无需满足一致性。

5、数据库系统中的故障类型有哪几种?

事务故障：发生在单个事务内部的故障。

介质故障：硬故障（Hard Crash），一般指磁盘损坏。

系统故障：软故障（Soft Crash），由于OS、DBMS软件问题或断电等问题导致内存数据丢失，但磁盘数据仍在。

6、数据库故障恢复的基本原则和方法是什么?

基本原则：冗余（Redundancy）。

实现方法：定期备份整个数据库；建立事务日志 (log)；通过备份和日志进行恢复。

8、先写日志原则的含义?

先写日志(Write-Ahead Log)原则：在数据被写到磁盘之前，对应此修改的日志记录必须已被写到磁盘上。

9、UNDO日志、REDO日志与UNDO/REDO日志

UNDO日志:

事务的每一个修改操作都生成一个日志记录 <T,x, old_value>；在x被写到磁盘之前，对应此修改的日志记录必须已被写到磁盘上；当事务的所有修改结果都已写入磁盘后，将 <Commit,T>日志记录写到磁盘上。

恢复：从头扫描日志，找出所有没有<Commit,T>或<Abort,T>的所有事务，放入一个事务列表L中；从尾部开始扫描日志记录<T,x,v>, 如果T∈ L，则write (X, v)，output (X)；For each T∈ L do write <Abort,T > to log。

总结： <T,x,v>记录修改前的旧值；写入<Commit,T>之前必须先将数据写入磁盘；恢复时忽略已提交事务，只撤销未提交事务（有<Commit,T>的事务肯定已写回磁盘）。

Redo日志:

x被写到磁盘之前，对应该修改的Redo日志记录必须已被写到磁盘上 (WAL)；在数据写回磁盘前先写<Commit,T>日志记录；日志中的数据修改记录<T, x, new_value>。

恢复：从头扫描日志，找出所有有<Commit,T>的事务，放入一个事务列表L中；从首部开始扫描日志记录<T,x,v>,如果T∈ L，则write (X, v)，output (X)；For each T∈ L do write <Abort,T > to log。

恢复的基础：没有<Commit,T>记录的操作必定没有改写磁盘数据，因此在恢复时可以不理会；有<Commit,T>记录的结果可能还未写回磁盘，因此在恢复时要Redo。

Undo基于立即更新 (Immediate Update)，Redo基于延迟更新 (Deferred Update)。

Undo：内存代价小，恢复代价高。
Redo：恢复代价小，内存代价高。

Undo/Redo日志:

在x被写到磁盘之前，对应该修改的日志记录必须已被写到磁盘上 (WAL)；日志中的数据修改记录<T, x, old_value，new_value > ；可以立即更新，也可以延迟更新。

恢复：正向扫描日志，将< commit>的事务放入 Redo 列表中，将没有结束的事务放入 Undo 列表；反向扫描日志，对于<T,x,v,w>，若 T 在 Undo 列表中，则 Write(x,v); Output(x)；正向扫描日志，对于< T, x, old_value, new_value > ，若 T 在Redo列表中，则Write(x,w)； Output(x)；对于Undo列表中的T，写入<abort,T>。（先 Undo，后 Redo）。

10、基于检查点的数据库恢复

检查点技术保证检查点之前的所有commit操作的结果已写回数据库，在恢复时不需REDO。

11、日志轮转的概念

采用Log Rotation节省存储。

第十一章 并发控制

1、并发操作可能带来什么问题?

丢失更新（Lost update）：事务T1提交的写操作被另一个事务T2的提交覆盖了

脏读（Dirty read / Uncommitted update）：务在内存中更新了但还未最终提交的数据

不一致分析 （Inconsistent analysis）：事务读了过时的数据，不是数据库的当前状态（不可重复读：事务内读到的数据被其它事务update或者delete了\幻像读：事务内读到的数据内容被其它事务的insert操作改变了）

2、并发事务可串化的概念

如果一个调度的结果与某一串行调度执行的结果等价，则称该调度是可串化调度，否则是不可串调度。

3、冲突可串的概念及判定

概念：如果调度中一对连续操作是冲突的，则意味着如果它们的执行顺序交换，则至少会改变其中一个事务的最终执行结果；如果两个连续操作不冲突，则可以在调度中交换顺序。

判定：冲突等价，冲突可串性；优先图无环；遵守2PL协议。

4、理解独占锁、共享锁、多粒度锁、意向锁等的含义

独占锁（X 锁）：若事务 T 对数据 R 加 X 锁，那么其它事务要等T释放X锁以后，才能获准对数据R进行封锁。只有获得R上的X锁的事务，才能对所封锁的数据进行修改。

共享锁（S锁）：果事务T对数据R加了S锁，则其它事务对R的X锁请求不能成功，但对R的S锁请求可以成功。这就保证了其它事务可以读取R但不能修改R，直到事务T释放S锁。当事务获得S锁后，如果要对数据R进行修改，则必须在修改前执行 Upgrade(R) 操作，将S锁升级为X锁。

- 事务在读取数据R前必须先获得 S 锁；事务在更新数据R前必须要获得X锁。如果该事务已具有R上的S锁，则必须将S锁升级为X锁；如果事务对锁的请求因为与其它事务已具有的锁不相容而被拒绝，则事务进入等待状态，直到其它事务释放锁。一旦释放一个锁，就不再请求任何锁。

更新锁（U锁）：如果事务取得了数据R上的更新锁，则可以读R，并且可以在以后升级为X锁；单纯的S锁不能升级为X锁；如果事务持有了R上的Update Lock，则其它事务不能得到R上的S锁、X锁以及Update锁；如果事务持有了R上的S Lock，则其它事务可以获取R上的Update Lock。

相容性矩阵：

	S	**X**	**U**
S	Y	N	Y
X	N	N	N
U	N	N	N

<S, U>是相容的：如果其它事务已经持有了S锁，则当前事务可以请求U锁，以获得较好的并发性;
<U, S>不相容：如果某个事务已持有U锁，则其它事务不能再获得S锁，因为持有U锁的事务可能会由于新的S锁而导致永远没有机会升级到X锁。

多粒度锁：同时支持多种不同的锁粒度（指加锁的数据对象的大小），允许多粒度树中的每个结点被独立地加S锁或X锁；对某个结点加锁，意味着其下层结点也被加了同类型的锁。

显式加锁：应事务的请求直接加到数据对象上的锁。
隐式加锁：本身没有被显式加锁，但因为其上层结点加了锁而使数据对象被加锁。
给一个结点显式加锁时必须考虑：该结点是否已有不相容锁存在；上层结点是否已有不相容的的锁（上层结点导致的隐式锁冲突）；所有下层结点中是否存在不相容的显式锁。

意向锁：

IS锁（Intent Share Lock，意向共享锁，意向读锁）：如果对某个结点加IS(IX)锁，则说明事务要对该结点的某个
下层结点加S (X)锁；对任一结点P加S(X)锁，必须先对从根结点到P的路径上的所有结点加IS(IX)锁。
IX锁（Intent Exlusive Lock，意向排它锁，意向写锁。

5、两阶段锁的含义？如何使用两阶段锁保证并发事务的可串行性?

含义：事务在对任何数据进行读写之前，首先要获得该数据上的锁；在释放一个锁之后，事务不再获得任何锁。

两段式事务：遵守2PL协议的事务。
定理：如果调度S中的所有事务都是两段式事务，则该调度是可串化调度。

6、死锁的检测与预防

死锁检测 Deadlock Detecting：检测到死锁，再解锁。

Timeout 超时、Waiting graph 等待图（结点：事务，边：资源等待关系，环：死锁）

死锁预防 Deadlock Prevention：提前采取措施防止出现死锁。

- 优先顺序方法：按封锁对象的某种优先顺序加锁。
- 时间戳：每个事务开始时赋予一个时间戳，如果事务T被Rollback然后再Restart，T的时间戳不变，T_i 请求被 T_j 持有的锁，根据 T_i 和 T_j 的timestamp决定锁的授予。
    - 等待 - 死亡：timestamp(T) < timestamp(U)，T等待，U不影响；timestamp(T) > timestamp(U)，T死亡（回滚）。
    - 伤害 - 等待：timestamp(T) < timestamp(U)，T 伤害 U（U回滚并释放锁，T获得锁）；timestamp(T) > timestamp(U)，T等待。

7、乐观并发控制的概念

乐观并发控制假定不太可能（但不是不可能）在多个用户间发生资源冲突，允许不锁定任何资源而执行事务。只有试图更改数据时才检查资源以确定是否发生冲突。如果发生冲突，应用程序必须读取数据并再次尝试进行更改。

第十二章 数据库安全性

1、自主访问控制和强制访问控制的概念

自主存取控制（DAC）：同一用户对于不同的数据对象有不同的存取权限；不同的用户对同一对象也有不同的权限；用户还可将其拥有的存取权限自主地转授给其他用户。GRANT..To../REVOKE。

强制存取控制（MAC）：每一个数据对象被标以一定的密级；每一个用户也被授予某一个级别的许可；对于任意一个对象，只有具有合法许可的用户才可以存取。

2、了解TCSEC的安全等级

- 无保护级

  D，最低安全性；

- 自主保护级

  C1，主客体分离、身份鉴别、数据完整性、自主存取控制DAC
  C2，审计；

- 强制保护级

  B1，强制存取控制MAC - - 可信系统(安全系统)
  B2，良好的结构化设计、形式化安全模型；
  B3，全面的访问控制、可信恢复；

- 验证保护级

  A1，形式化认证。

3、基于视图的数据库安全增强

视图机制把要保密的数据对无权存取这些数据的用户隐藏起来；
视图机制更主要的功能在于提供数据独立性，其安全保护功能不够精细，往往远不能达到应用系统的要求；
实际中，视图机制与授权机制配合使用：首先用视图机制屏蔽掉一部分保密数据，视图上面再进一步定义存取权限，间接实现了用户自定义的安全控制。

第十三章 数据库完整性

1、了解数据库完整性的概念

数据库完整性防止合法用户使用数据库时向数据库加入不符合语义的数据。防止错误的数据进入数据库。

数据库的完整性是指保护数据库中数据的：
正确性：数据的合法性。如年龄由数字组成
有效性：数据是否在有效范围内。如月份取1 - 12
相容性：指表示同一个事实的两个数据应该一致。如一个人的性别只有一个

2、数据库完整性控制机制的主要功能

定义功能：提供定义完整性约束条件的机制
检查功能：检查用户发出的操作请求是否违背了约束条件。立即执行约束（一条语句执行完成后立即检查）；延迟执行约束（整个事务执行完毕后再检查）。
违约响应功能：如果发现用户操作请求使数据违背了完整性约束条件，则采取一定的动作来保证数据的完整性。

3、完整性规则的形式化定义

一条完整性规则是一个五元组（D,O,A,C,P）
D(Data)：约束作用的数据对象
O(Operation)： 触发完整性检查的数据库操作。即当用户发出什么操作请求时需要检查该完整性规则，是立即检查还是延迟检查。
A(Assertion)： 数据对象要满足的断言或语义规则
C(Condition)： 受A作用的数据对象值的谓词
P(Procedure)：违反完整性规则时触发的过程

4、完整性约束的分类

- 按约束的粒度：
  - 表级约束：若干元组间、关系上以及关系之间联系的约束。
  - 列级约束：针对列的类型、取值范围、精度等而制定的约束条件。
  - 元组级约束：元组中的字段组和字段间联系的约束。
- 按约束对象的状态：
  - 静态约束：数据库每一确定状态时的数据对象所应满足的约束条件
  - 动态约束：数据库从一种状态转变为另一种状态时，新、旧值之间所应满足的约束条件
- 按约束作用类型分：
  - 域完整性：域完整性为列或列组指定一个有效的数据集，并确定该列是否允许为空
  - 实体完整性：实体完整性要求表的主码不能为空
  - 参照完整性：参照完整性维护参照表中的外码与被参照表中候选码之间的相容关系。如果在被参照表中某一元组被外码参照，那么这一行既不能被删除，也不能更改

5、完整性实现方法

- 约束（Constraint）：限制输入到表中的值的范围。是实施数据完整性的首选方法。

  主键约束（Primary Key）唯一键约束（Unique）外键约束（Foreign Key）检查约束（Check）
- 触发器（Trigger）：与特定表关联的存储过程。当在该表上执行DML操作时，可以自动触发该存储过程执行相应的操作

  触发操作：Update、Insert、Delete。通过触发器可以实现复杂的约束，例如动态约束
- 规则（Rule）：一组用过程化SQL书写的条件语句（Where子句中合法的语句一般都可以用作规则：算术运算符；关系运算符；IN、LIKE、BETWEEN等关键字）；规则可以和列或用户定义类型捆绑在一起，检查数据完整性；多个列可以共用一个规则。
- 断言（Assertion）：Create Assertion*创建一个断言，对断言涉及的数据进行操作时会触发断言；断言为假时操作将被拒绝

第十四章 高级主题

1、了解分布式数据库的主要特点：物理上分布、逻辑上一体

物理分布性、逻辑整体性、站点自治性、数据透明性

2、面向对象数据库的特点：持久化OO语言

3、对象关系数据库的特点：支持ADT扩展（ORDB）

4、NoSQL与SQL数据库的对比

主要类型：键值数据库、列存储数据库、文档数据库和图数据库

NoSQL数据库大都以“KEY+VALUE”结构为基础进行数据表示和存储， KEY与VALUE都以字节流（byte string）存储