第一章 数据库基本介绍

2021年3月20日 21:24

传统"基于文件的数据管理"的缺点：

- 数据是独立且分离的
- 数据的复制易导致数据失去健全性
- 数据-程序具有依赖性，数据结构改变会要求程序也进行相应的改变
- 文件格式不相容问题：不同的编程语言对存储数据的文件要求不同
- 非常依赖程序开发者，否则数据的安全性、易恢复性和共享性难以得到保证

数据库的定义：

定义一 数据库是一个对 逻辑相关的数据 的描述以及共享集合，其设计满足特定组织机构的数据需求

定义二 数据库是有自我描述的完整记录的集合

解释：

数据的描述 又称系统目录（system catalog）、数据词典（data dictionary）或 元数据（meta-data, 有关数据的数据）

数据库的自我描述性质，决定了其可以实现 程序-数据独立性（脱耦合）

数据库管理系统（DBMS）：

DBMS是一个能使用户 定义、创建、维持数据库 以及 控制对数据库的访问 的软件

DBMS的组成：

- 数据定义语言 DDL：确定数据类型、数据结构以及数据限制
- 数据操作语言 DML：完成从数据库中增删改查的操作
- 控制数据库访问的组件：保障数据的完整性、安全性、控制数据同步、数据恢复以及存储用户可访问性目录
- 视图机制：用于展示数据库的特定子集

数据库中的结构被称为 **模式 schema**

一个完整的DBMS环境包含哪些组件?

- 硬件：计算机、网络设备等
- 软件：DBMS及其附加软件、操作系统、网络驱动等
- 数据：数据库中包含的数据，包括可操控数据和元数据
- 流程：对数据库进行操作的步骤和规范，如登入登出DBMS，修改数据库结构，备份数据库等
- 人：对数据库和DBMS进行操作的主体

人 在数据库环境中的角色：

- 数据管理者 DA：负责管理数据源，数据源包括 数据库计划、数据库开发、维护数据库的标准、政策、流程以及概念层面的数据库设计
- 数据库管理者 DBA：负责管理物理层面上数据库的实现，包括物理数据库的实现，保证数据库的安全和完整性，维护操作系统以及保证用户对数据库软件的使用体验
- 数据库设计者
  - 逻辑数据库设计者：定义数据，确定数据之间的关系，以及在数据库中以何种方式存储
  - 物理数据库设计者：决定逻辑数据如何在物理硬件上实现
- 应用程序开发者：为用户提供可以满足功能需求的数据库程序
- 末端用户：数据库的"客户clinets"，提供需要管理的数据的人

DBA的职责

- 1)决定DB的信息内容，结构
- 2)决定DB的存储结构，策略
- 3)定义数据的安全性要求，完整性约束条件
- 4)监督控制DB的运行
- 5)DBMS的改进，重组重构

DBMS的优缺点：

优点：

- 能够管理数据冗余
- 数据具有连续性

- 数据可以分享，并且方便分享
- 提高数据的健壮性
- 保障数据安全

缺点：

- DBMS往往非常复杂
- DBMS的成本较高
- DBMS对性能的要求更高
- 一旦出现错误或意外，DBMS造成的影响更大

第二章 数据库环境

2021年3月22日 9:20

ANSI-SPARC 三级模式结构：

目的：对于每个单独的用户，将其看到的数据库视图与实际数据库的物理表现相分离。

内容：

外部层 external level：又称用户层，即用户看到的数据库

包括一系列不同的数据库视图（View）

概念层 conceptual level：描述所有存储在数据库的数据以及它们之间的关系，

包括整个数据库的所有逻辑结构，与DBA所管理的一致。

具体来说包括：

- 所有的条目，以及它们的属性和关系
- 对数据的限制
- 数据的语义信息
- 有关安全性和健壮性的信息

内部层 internal level：在计算机物理层面上数据库的实现，其描述了数据库中的数据如何存储在硬件上，包括在存储硬件上需要的数据结构和文件组织。

具体来说有：

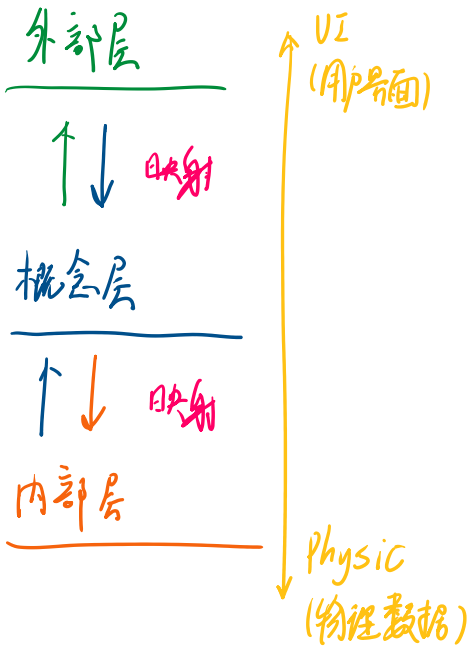
分配数据和其索引在存储器上空间的信息

存储信息的记录（如数据大小等）

存储位置的记录

数据压缩或加密信息

在内部层以下，数据的管理交由操作系统而非DBMS



三级模式的两级映射：

对整个数据库的描述被称为数据库的 模式schema，根据三级模式，可以将schema分为

外部模式、概念模式和内部模式。

三级模式之间有着互相映射的关系。

外部-概念层映射：将外部模式映射到概念模式

概念-内部层映射：将概念模式映射到内部模式

在一个确定的时间点，称此时数据库中的数据构成一个 数据库实例 database instance

数据库模式(schema) 又称 数据库的内涵(intension)

数据库实例(instance) 又称 数据库的外延(extension)或状态(state)

三级模式两级映射提供了 逻辑数据独立性 和 物理数据独立性：

- 逻辑数据独立性：指概念层模式被改变时（如添加删除条目），外部层模式不需要改变的特性
- 物理数据独立性：指内部层模式被改变时（如改变存储文件的结构或使用不同的存储设备），外部层和概念层模式不需要被改变的特性

数据库语言

包括 DDL（数据定义语言）和DML（数据操作语言）

DDL的定义：

一种允许DBA（或用户）来描述和命名条目、属性及联系，以及与其相关的完整性约束和安全性约束的语言。

- DDL可以用来定义或修改模式（schema）
- 编译后的DDL语句应为一系列存储在特定文件中的表，它们被统称为系统目录 system catalog，即代表元数据

DML的定义：

提供对数据库中数据的一系列基础操作的支持的语言

包括“增删改查”

- DML中关于查询数据的部分也被称为 查询语言 query language

DML的分类：

- 过程型(procedural)DML：
  - 用户会告诉系统需要取用什么数据以及（准确地）如何取用数据
- 非过程型DML：
  - 用户值告诉系统需要取用什么数据但不关心系统如何取用数据

网络和结构型DBMS的DML是过程型的，而关系型DBMS常用非过程型DML

第四代语言（4GL）：指用户在不知道程序操作原理的情况下就可以实现功能的语言。因此它代表非过程型语言。

数据模型：

描述和操纵组织机构内的数据、数据间的联系以及数据的约束的模型

数据模型的三个要素：

- 结构部分：描述创建数据库的规则
- 操作部分：描述并分类 允许对数据进行的操作
- 完整性规则集合： 保证数据的准确性

数据模型的分类：

- 基于对象的数据模型：使用 实体、属性、关系这样对象的数据模型。

  实体 entities：数据库中可区分的对象（人、地点、事务、概率、事件）

  属性 attributes: 对象的性质

  关系 relationships：实体间的关联

  由此可以构造的数据模型有：ER，语义(semantic)，函数(functional), 面向对象（OO）

  现在常用的数据库设计很多都基于 实体-联系的ER模型

- 基于记录的数据模型：数据库由若干不同类型的固定格式记录组成
  - 包括：
    - 关系数据模型relation：数据和联系均以表格形式表示，列名称是唯一的
      - 适用于外部层和概念层，而不适用于物理层
    - 网状数据模型network：数据表示成一组记录record，联系被表示为络set
      - 可以用图结构来实现物理层
    - 层次数据模型hierachical：同网状模型，但规定每个节点只能有一个父节点
      - 可以用树结构来实现物理层
- 物理数据模型：描述数据如何存储在硬件中
  - 包括 统一数据模型 和 帧存储模型

DBMS的功能：

1. 数据存储、检索和更新
2. 提供用户可访问的目录
3. 事务支持，确保执行数据操作的一致性
4. 并发控制，支持多个用户并行更新数据库
5. 恢复服务，能够恢复数据库
6. 授权服务，保障只有授权用户可以访问
7. 数据通信，提供网络远程访问
8. 完整性服务，对数据的存储和修改遵循一定的规则
9. 提高数据独立性
10. 实用程序服务，其他提高数据库管理效率的功能

多用户DBMS的结构：

- 远程处理结构：一台主机服务器+n个终端
- 文件服务器结构：
  - 一台"文件服务器"用来连接数据库
  - DBMS安装在各个客户端上，用来连接文件服务器
- 客户服务器结构
  - DBMS安装在服务器端上，连接数据库以及处理各个客户端的请求
    - 2层：客户端直接对接数据库服务器
      - 缺点：客户端的开销依然很大
    - 3层：客户端通过一个 应用服务器 Transaction Processing Monitors 作为中介 连接数据库服务器
      - 客户端只负责用户界面
      - 应用服务器负责 业务逻辑 和 数据逻辑处理
      - 数据库服务器只负责 数据访问和确认
    - n层：客户端在连接应用服务器前还需经过一个或多个Web服务器
      - 优点是可以获得高效的 负载均衡

第三章 关系模型

2021年3月25日 8:01

关系型数据库管理系统（RDBMS）

关系：一组域上的笛卡尔积的有限子集

简单解释：关系是由行和列组成的表

属性：属性是关系中的列

元组：元组是关系中的一行

域：一个或多个属性的取值集合

维度：关系中包含属性的个数

基数：关系中包含元组的个数

关系又称 文件

元组又称 记录

属性又称 字段

关系模式：用一组 属性和域 进行一一对应从而形成的 关系

一个关系模式可以对应出n个关系，其中n表示构成关系模式的（属性:域）元素对的数量。

关系模式可以看作关系的“模板”

关系数据库模式：若干 关系模式 组成的集合

关系的性质：

- 关系模式中的所有关系不能重名
- 关系中的每个元素（每个单元格的值）都是原子的，不可再分的
- 一个关系中的所有属性不能重名
- 一个关系中不存在重复元组
- 属性 没有顺序
- 理论上，元组也没有顺序（但顺序会影响物理上的访问效率

关系中的关键字：

超关键字 superkey： 唯一标识关系中的每个元组的 一个属性或属性集合

包含多个属性的关键字称为“合成关键字”

候选关键字 candidate key：本身是超关键字但其任何子集都不再是超关键字的 属性集合

主关键字 primary key（PK）：从候选关键字中下主动选择的一个关键字

因为关系没有重复元组，因此每个关系一定有一个主关键字，在有多个候选关键字的情况下才主动选择PK，未被选择的CK称为 可替换关键字

外部关键字 Foreign key：某一个关系中的SK与另一个关系（包括自己）的CK匹配时，称该SK为 外部关键字

关系中的"完整性约束"：

实体完整性 entity integrity	基本关系中，主关键字PK的值不能为空
引用完整性 referential integrity	如果关系中存在外部关键字FK，则FK的值要么为空，要么与外部关系中某个元组的CK取值相等
一般性约束	其他用户需要加入的约束（如元组数量上限等

空 NULL：表示一个元组的属性值为 不知道 或 不可用 的情况

基本关系 base relation：

可以与**概念模式 conceptional schema** 中的一个实体 相对应的 一个（有名称的具体）关系

- 基本关系的元组都一定存储在在数据库的物理结构中

视图 View:

对一个或多个基本关系进行操作得到的结果

- 视图是动态的，可以根据用户需求随时生成
- 视图是 **虚关系**，或者是 呈现在用户面前的关系，而没有物理存储
- 用户可以对视图进行动态操作，操作会作用到对应的基本关系上
  - 对视图的更新或操作可能要遵循限制，因此视图可以分为：
    - 不可更新（理论上
    - 可更新（理论上
    - 部分可更新
  - 一般的限制为：
    - 可以更新：一个基本关系的视图，并且包括PK或CK
    - 不可更新：视图中包含多个基本关系，或是生成视图时涉及到

分组操作

第四章 关系代数及其运算

关系代数是一种纯理论语言，是DML的基础

关系代数的运算在 关系的闭包 上的运算即：关系代数的操作数和结构都属于关系集合

关系代数的八种运算：

基本（5）

选择	Selection
投影	Projection
笛卡尔积	Cartesian product
集合并	Union
集合差	Set Difference

复合运算（3）

连接	Join
集合交	Intersection
除	Division

- 其中一元运算：选择 投影
  其他为二元运算

[一元运算] Unary:

① Selection 选择运算

符号：$\sigma_F(R) = R'$

F：条件（又称"谓词"）　R：作用于关系R　R'：结果（新关系）

例：

R：

No	I	B	C
A1	20	1	w
A2	40	1	h
A3	60	0	t

S：

I	B	E
20	1	3.6
20	0	2.7
40	1	0.4
40	0	8.1

$\sigma_{I>40}(R)$ =

No	I	B	C
A3	60	0	t

② projection 投影运算

符号：$\Pi_{C_1, C_2, \dots, C_n}(R) = R'$

$C_1, C_2, \dots, C_n$：指定的属性（列）　R：作用对象　R'：新关系

[二元运算] Set (集运算)

③ 集合并 Set-Union

表达式：$R \cup S = R'$

$(x \in R \mid x \in S \rightarrow x \in R')$

④ 集合差 Set-Difference

表达式：$R - S = R'$

$(x \in R \ \& \ x \notin S \rightarrow x \in R')$

⑤ 集合交 Set-Intersection

表达式：$R \cap S = R - (R - S)$

$(x \in R \ \& \ x \in S \rightarrow x \in R')$

⑥ 笛卡尔积 Cartesian Product

表达式：$R \times S = R'$

元组 i n i×j →元组新维

属性 j m m×n

$\sigma_{No,B,C}(R)$ =

No	B	C
A1	1	w
A2	1	h
A3	0	t

$R \times S$：

	No	I	B	C	I	B	E
▶	1	20	1	w	40	1	0.4
	1	20	1	w	20	0	2.7
	1	20	1	w	20	1	3.6
	1	20	1	w	40	0	8.1
	2	40	1	h	40	1	0.4
	2	40	1	h	20	0	2.7
	2	40	1	h	20	1	3.6
	2	40	1	h	40	0	8.1
	3	60	0	t	40	1	0.4
	3	60	0	t	20	0	2.7
	3	60	0	t	20	1	3.6
	3	60	0	t	40	0	8.1

重命名：$\rho_S(E)$ 表达式

$\rho_{S(a_1, a_2, \dots a_n)}(E)$

S：新的名称　$a_1, a_2, \dots a_n$：新的属性名称

⑦ 连接 Join

1. θ-连接

$R \bowtie_F S = \sigma_F(R \times S)$

条件格式：$a_i \ \theta \ b_i$，$\theta \in \{<, \leq, >, \geq, =, \neq\}$　a_i ⇒R的属性　b_i ⇒S的属性

（θ为"="时，称为"等值"）

2. 自然连接：

$R \bowtie S = \sigma_{\text{All } R.x = S.x}(R \times S)$

R和S在所有公共属性上的等值

(左)

3. 外连接

$R \ \overline{\bowtie} \ S = \sigma_{\text{All } x \in R}(R \times S)$

保留R中所有元组，若有元组不含S中的属性，将R⟗S中该属性置为空

4. 半连接

$R \bowtie_{R.B=S.B} S$：

N_O	I_R	C	I_S	B	E
1	20	w	40	1	0.4
1	20	w	20	1	3.6
2	40	h	40	1	0.4
2	40	h	20	1	3.6
3	60	t	20	0	2.7
3	60	t	40	0	8.1

$R \rhd_{R.I=S.I} S$：

⇒

N_O	I_R	B_R	C
1	20	1	w
2	40	1	h

$R \bowtie S$：

No	I	B	C	E
1	20	1	w	3.6
2	40	1	h	0.4

$R \overline{\bowtie}_I S$：

No	I_R	C	I_S	B_R	B_S	E

4. 半连接

$R \ltimes_F S = \Pi_A(R \bowtie_F S)$

$A = \{x \mid x \in R\}$

（先θ连接，再投影仅取R的属性）

$R \ltimes_I S$：

No	I_R	C	I_S	B_R	B_S	E
1	20	w	20	1	0	2.7
1	20	w	20	1	1	3.6
2	40	h	40	1	1	0.4
2	40	h	40	1	0	8.1
3	60	t	NULL	0	NULL	NULL

⑧ 除法 Divide

$R \div S = T_C = T_1 - \Pi_C((T_1 \times S) - R)$

（要求：S的属性集合⊆R的属性集合）其中 $T_1 = \Pi_C(R)$

$C = \{x \mid x \in R, x \notin S\}$

前缘的属性集合.

第五章 SQL 数据操作语言

2021年4月1日 9:31

SQL语言基本概念说明

术语区别：

正式术语	SQL术语
关系	表
属性	列
元组	行

SQL语句： 保留字 + 用户自定义字;
（结束符为";"）

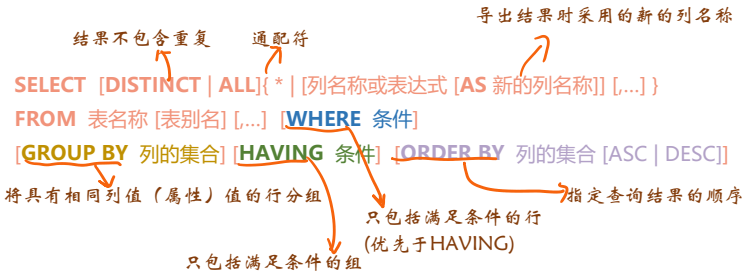
- 保留字：语言的固定部分，必须准确拼写，不能跨行拼写
- 用户自定义字：用户根据一定的语法规则自己定义
  P.S. 用户可以通过一些语句来更改结束符，由此方便程序式SQL的书写

SQL语句中除 字符数据常量 外，其他部分不区分大小写
对于字符变量，用单引号''存储，所以''内的字符是区分大小写的

数据操作语言(DML)详解

选择语句SELECT

BNF定义：



WHERE和HAVING的 条件 或 条件复合句：

相等： = 不等： <>，大小比较符号与一般编程语言相同
和：条件1 AND 条件2
或： 条件1 OR 条件2
范围： 列名称 BETWEEN 下限 AND 上限 【包括上下限】 |
列名称 NOT BETWEEN 下限 AND 上限
成员测试： 检查列名称中是否包含集合中的值
列名称 IN | NOT IN （值的集合）
模式匹配：
列名称 LIKE 模式串 | 列名称 NOT LIKE 模式串
模式串规则：
% 或 * : 通配符，匹配一个或多个字符
_ 或 ? : 匹配任意一个字符
: 转义符号，消除%和_的特殊含义
空查找条件：仅检查列值是否为空
列名称 IS NULL | 列名称 IS NOT NULL

SQL聚集函数：用来替代SELECT语句中的列名称表达式，来实现对数据库中数据的统计，SQL中的标准聚集函数：

- SELECT COUNT(列名称,...) : 返回指定列中数据的个数
- SELECT SUM(列名称,...) : 返回指定列中数据的和
- SELECT AVG(列名称,...) : 返回指定列中数据的平均值
- SELECT MIN(列名称,...) : 返回指定列中数据的最小值
- SELECT MAX(列名称,...) : 返回指定列中数据的最大值
  - 可以用 "AS 新的列名称" 来指定 返回的新数据列的名称

子查询（嵌套查询）：在SELECT 语句中嵌套一个SELECT语句（也可以在INSERT UPDATE 和 DELETE 语句中嵌套SELECT 语句）：

- 用小括号括起子语句

GROUP BY 列的集合：
分组操作：
在查询结果的行中：
如果有若干行 **除了GROUP BY 指定的列 以外** 其他的列的值均相同，则在结果中将其合并为同一行
因此分组操作常与SELECT聚集函数同时使用，将满足同一列属性的行的统计数据合并起来。
使用GROUP BY的注意事项：

- SELECT子句的列名称中，除出现在聚集函数中的外，必须在GROUP BY的列表中出现
- 同时使用GROUP BY 和 WHERE 子句时，WHERE必须先执行(因此要想筛选GROUP BY 后的结果行，请使用HAVING)
- 空值也看做相等的值，会被GROUP BY合并

HAVING 分组条件：
将GROUP BY 后的结果行 中 **不满足HAVING子句**的行过滤
HAVING中可以使用聚集函数作为条件的左侧，而WHERE子句中不行

ORDER BY 列的集合 [ASC| DESC]：
将选择结果排序：
如果参数是多个列，则靠前的列具有优先级，即如果出现该列相同的行，再按照靠后位置的列进行排序
用作排序的列集合 在一些SQL的实现中 要求必须出现在 SELECT列表中
排序选项：

ASC	升序(默认)
DESC	降序

子查询（嵌套查询）：在SELECT 语句中嵌套一个SELECT语句（也可以在INSERT UPDATE 和 DELETE 语句中嵌套SELECT 语句）：

- 用小括号括起子语句
- 子语句返回的值可以视为一个临时表
- ORDER BY 不能用于子查询
- 子查询的结果作为条件返回的时候，只能位于条件表达式右侧

条件式中，可以用IN,ANY(SOME),ALL从子查询的结果中筛选值：

- IN（子查询语句）：值是否在查询结果表中
- ANY(子查询语句)：查询结果表中的 任何值 进行比较为真，则WHERE语句为真
- ALL（子查询语句）：查询结果表中的 所有值 都进行比较，均满足条件才为真

EXISTS关键字判断返回子语句的查询结构是否为空：
EXISTS(【select子语句】) => 如果子语句的查询结构为空，返回假，否则返回真
NOT EXISTS(【select子语句】) => 返回结果与EXISTS相反

连接和多表查询：

- **简单连接：**

  方法1（选择多个表中指定的列）

  ```
  SELECT a.attrKey a.attrA attrB
  FROM tableA a , tableB b , ...
  WHERE a.attrKey = b.attrKey
  ```

  -> 三表连接：

  ```
  SELECT a.attrKey1 b.attrKey2 attrB attrC
  FROM tableA a , tableB b , tableC c
  WHERE a.attrKey1 = b.attrKey1 AND b.attrKey2 = c.attrKey2
  ```

  方法2（产生多个attrKey的列）

  ```
  SELECT (...)
  FROM tableA a JOIN tableB b ON a.attrKey = b.attrKey
  ```

  -> 三表连接：

  ```
  ...
  FROM (tableA a JOIN tableB b USING attrKey1) AS ab
                 JOIN tableC c USING attrKey2
  ```

  方法3（只产生一个attrKey的列）

  ```
  SELECT (...)
  FROM tableA  JOIN tableB USING attrKey
  ```

  或:

  ```
  SELECT (...)
  FROM tableA NATURAL JOIN tableB
  ```

  - 对连接结果排序：
    + ORDER BY a.attr_i , b.attr_j , ...

  - 使用 EXISTS 查询:

    ```
    SELECT (...)
    FROM (...)
    WHERE EXISTS(【select子语句】)
    ```

    如果子语句查询到结果，EXISTS返回结果true，否则为false

- **外连接：**

  左外连接:

  ```
  SELECT a.*,b.*
  FROM tableA a LEFT JOIN tableB b ON a.attrKey = b.attrKey
  ```

  右外连接：

  ```
  SELECT a.*,b.*
  FROM tableA a RIGHT JOIN tableB b ON a.attrKey = b.attrKey
  ```

表的合并（注意操作的表要有并兼容性）：

- 并运算：UNION

  (【select语句1】)
  UNION [ALL] [CORRESPONDING [BY 列的集合]]
  (【select语句2】)

  额外选项

指定 ALL	包括所有重复的行
指定 CORRESPONDING	仅在两个表的共同的列上合并
指定 CORRESPONDING [BY 列的集合]	仅在指定的列集合上合并

- 交运算：INTERSECT
- 差运算：EXCEPT

  (语法与UNION相同)

交和差运算的等价形式：

(SELECT key FROM A) INTERSECT (SELECT key FROM B) 等价于: SELECT DISTINCT a.key FROM A a , B b WHRER a.key = b.key
(SELECT key FROM A) EXCEPT (SELECT key FROM B) 等价于: SELECT DISTINCT key FROM A WHRER key NOT IN (SELECT key FROM B)

更改语句UPDATE

基本语法：

```
UPDATE 表名
    SET 列名称 = 更新的值或表达式 [,...]
    [WHRER 条件]
```

- 将满足条件的行的 某些列 更新到新的值，当不写WHRER子句时，更新整个列的数据

- 表达式可以包括数学运算等

插入语句INSERT

基本语法：

INSERT INTO 表名 [(列的集合)] VALUES (数据值的集合)

- 指定 列的集合：列集合中的名称必须与对应表中的列名称完全一致，并且数据类型与后面数据值的集合相对应，将对应的数据插入新的一行
- 不指定 列的集合：默认插入的数据值集合与对应表的 所有列 按照顺序一一对应

注意：插入值时，所有字符变量必须由单引号 '' 包括，日期类型用 DATE'日期' 写明

或:

INSERT INTO 表名 【select子语句】

使用SELECT语句将另一个表的一行或多行插入 目标表 中，注意所选出的列必须均可对应

删除语句DELETE

基本语法：

DELETE FROM 表名 [WHRER 条件]

指定WHRER子句时，删除满足条件的行，否则，删除所有行

SQL语句规范举例：

1	SELECT staffNo, fName, lName, position FROM Staff WHERE position='Manager' OR position='Supervisor';	
2	SELECT staffNo, fName, lName FROM Staff WHERE fName LIKE 'Nick#_%';	
3	SELECT propertyNo, type, rooms, rent FROM PropertyForRent ORDER BY type, rooms ,rent DESC;	PL94 Flat 4 400 PG36 Flat 3 375 PG4 Flat 3 350
4	SELECT COUNT(*) AS count FROM PropertyForRent WHERE rent > 350;	
5	SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum FROM Staff GROUP BY branchNo ORDER BY branchNo;	branchNo count sum B003 3 54000.00 B005 2 39000.00 B007 1 9000.00

第六章 SQL 数据定义语言

2021年4月22日 8:58

ISO标准数据库数据类型：

布尔类型	BOOLEAN
字符类型	CHAR（固定长度） VARCHAR（可变长度，只规定最大长度）
二进制类型	BIT BIT VARYING（新版标准已移除）
数字类型（精确）	NUMERIC DECIMAL（十进制小数，用(a,b)表示 总位长度 和小数位长度） INTEGER（大整数） SMALLINT（小整数，最大32767）
数字类型（非精确）	FLOAT（浮点类型，加入进度参数决定尾数的精度） REAL DOUBLE PRECISION
日期时间类型	DATE（日期，y/m/d的格式） TIME（默认精确到秒的时间，可以用参数指定时间的精度(0-6)和时区） TIMESTAMP（默认精确到微秒的时间（精度6））
时间间隔类型	INTERVAL
大型数据类型	CHARACTER LARGE OBJECT BINARY LARGE OBJECT

数据库

创建数据库（模式）：

```
CREATE SCHEMA [模式名称 | AUTHORIZATION 创建者]
```

删除数据库（模式）：

```
DROP SCHEMA 模式名称 [RESTRICT | CASCADE ]
```

RESTRICT	受限，只有在模式为空的情况下才可以删除
CASCADE	递归删除模式下的所有内容

表

创建表：

```
CREATE TABLE 表名称
(
    {
        列名称 数据类型 [NOT NULL] [UNIQUE] [DEFAULT 默认值]
            [[CONSTRAINT 约束名称] CHECK 约束规则] ,...
        ,...
    }
        PRIMARY KEY (列的集合),
    {
        [UNIQUE (列的集合)]
        ,...
    }
    {
        [FOREIGN KEY (列的集合) REFERENCES 父表名称 [(父表列的集合)],
                                    [ON UPDATE 引用操作] [ON DELETE 引用操作] ]
        ,...
    }
    {
        [CHECK 约束规则]
        ,...
    }

)
```

完整性增强特性 IEF

SQL完整性增强特性（IEF）的五个方面：

- 非空值约束
- 域约束
- 实体完整性
- 引用完整性
- 一般性约束

非空值约束：不允许某些列为空

在定义列的结尾加上 NOT NULL

域约束语法：自行规定列的合法值集合

```
CHECK （列名 [对值进行规范约束]）
```

规范约束可以使用 VALUE IN (集合或SELECT子句返回的表) 等与判断关键字实现

自定义数据域约束规则（以该规则创建的域数据都要遵循该规则）：

```
CREATE DOMAIN 规则名称 AS 数据类型
    DEFAULT 默认值
    CHECK (VALUE [对值进行规范约束])
```

删除数据域约束规则：

```
DROP DOMAIN 规则名称 [ RESTRICT | CASCADE ]
```

RESTRICT	受限，只有在该规则没有被使用的情况下才可以删除
CASCADE	直接删除规则，如果规则已经被使用，将使用的域的数据类型更改为相应默认的类型

这样创建的规则名称在定义列名时代替数据类型使用

实体完整性：确定主关键字

每个表中只能使用一个关键字子句：

PRIMARY KEY(列名 ,...)

确定一个或多个列的组合 为 该表的主关键字，所列的值必须是唯一的非空值

若除PK外，仍需声明更多的列为唯一的非空值，可使用：

```
UNIQUE(列名 ,...)
```

引用完整性：外部关键字必须是父表中存在的有效元组

定义外部关键字：

```
FOREIGN KEY(外部列名) REFERENCES 外部表名
        [ON [DELETE | UPDATE] [CASCADE | SET NULL | SET DEFAULT | NO ACTION] ]
```

创建数据域类型约束：

DML语句	创建的类型名	实际类型	约束规则
CREATE DOMAIN StaffNumber AS VARCHAR(5) CHECK (VALUE IN (SELECT staffNo FROM Staff));	StaffNumber	可变字符串 最大长度为5	值在Staff表的staffNo列中
CREATE DOMAIN PNumber AS VARCHAR(5);	PNumber	可变字符串 最大长度为5	无
CREATE DOMAIN PRooms AS SMALLINT CHECK(VALUE BETWEEN 1 AND 15);	PRooms	小整形	在[1,15]范围内的整数
CREATE DOMAIN PRent AS DECIMAL(6,2) CHECK(VALUE BETWEEN 0 AND 9999.99);	PRent	十进制小数	在[0,9999.99]之间的小数，精度为小数点后两位

创建数据表：

CREATE TABLE PropertyForRent (

DML语句：	列名称	约束	解释
propertyNo PNumber NOT NULL,	propertyNo	非空	
rooms PRooms NOT NULL DEFAULT 4,	rooms	非空，默认值4	
rent PRent NOT NULL DEFAULT 600,	rent	非空，默认值600	
ownerNo OwnerNumber NOT NULL,	ownerNo	非空	
staffNo StaffNumber CONSTRAINT StaffNotHandlingTooMuch CHECK (NOT EXISTS (SELECT staffNo FROM propertyForRent GROUP BY staffNo HAVING COUNT(*)>100)),	staffNo	定义一个约束名词为StaffNotHandlingTooMuch，要求该列的值在查询PropertyForRent（即本表自己）中的结果总数不超过	即不允许同一个员工同时处理100个及以上的房屋

引用完整性：外部关键字必须是父表中存在的有效元组

定义外部关键字：

FOREIGN KEY(外部列名) REFERENCES 外部表名

[ON [DELETE|UPDATE] [CASCADE | SET NULL | SET DEFAULT | NO ACTION]]

引用操作：

CASCADE	删除或更新父表中的行，并自动删除或更新子表中匹配的行
SET NULL	删除或更新父表中的行，并设置子表中的外部关键字为NULL（外部关键字不能为NOT NULL）
SET DEFAULT	删除或更新父表中的行，并设置子表中的外部关键字为默认值（默认值必须已经指定）
NO ACTION	不允许对父表的删除或更新操作（ON DELETE 的默认操作）

一般性约束：自定义的，可应用到多个表的规则

CREATE ASSERTION 约束规则名称

CHECK (约束规则)

修改表的定义：

ALTER TABLE 表名

可用子句：

ADD 列名称 数据类型 [NOT NULL] [UNIQUE] [DEFAULT 默认值] [CHECK 约束规则]

=> 添加一个新列

ADD 【表约束定义子句】

=> 增加一个约束定义，子句为 PRIMARY KEY | UNIQUE | FOREIGN KEY | CHECK中的一个

DROP 列名称 [RESTRICT | CASCADE]

=> 删除一个列

执行方式标识:

RESTRICT	仅当该列没有被其他数据库对象引用时，才允许删除
CASCADE	删除该列，且从其他引用该列的数据库对象中自动删除该列

DROP CONSTRAINT 约束规则名称 [RESTRICT | CASCADE]

=> 删除一个约束规则

ALTER 列名 SET DEFAULT 默认值

=> 改变一个列的默认值

ALTER 列名 DROP DEFAULT

=> 删除一个列的默认值

删除表：

DROP TABLE 表名 [RESTRICT | CASCADE]

执行方式标识:

RESTRICT	仅在该表没有其他对象依赖时允许删除表
CASCADE	直接删除表，并自动递归删除依赖的对象

索引

创建索引：

CREATE [UNIQUE] INDEX 索引名称

ON 表名 (列名 [ASC|DESC] , ...)

唯一性标识：UNIQUE，将强制索引与列进行唯一组合，对于PK的索引，必须要是UNIQUE的

列的排序标识：

ASC	升序（默认）
DESC	降序

删除索引：

DROP 索引名称

视图

视图的优缺点：

优点	缺点
1 保障数据独立性 2 实时性，基表的改变会立刻反映到视图中 3 提高了安全性，限制了用户对数据库访问的范围 4 降低了复杂性，简化查询，将多表查询简化为单表查询 5 方便快捷，降低用户使用难度 6 用户化，用户可以定制数据库的呈现形式 7 保障数据完整性	1 具有更新局限性，在一些情况下视图不可更新 2 结构局限性，在视图创建后加入基表的列不能出现在视图上 3 使用视图会带来一定的性能开销

创建视图

CREATE VIEW 视图名称 [(列的别名 ,...)]

AS 【select子语句】 [WITH [CASCADED | LOACL] CHECK OPTION]

该语句会将满足 查询语句 （又称 定义查询）的行加入视图的基表中，并由此创建一个视图

执行方式标识:

WITH LOCAL CHECK OPTION	除非基表改变，否则不允许任何 更新视图 或 定义新视图等的操作使得视图中的行被删除
WITH CASCADED CHECK OPTION	（默认）任何时候都不允许视图中的行被 更新视图 或 定义新视图等操作删除

删除视图

DROP VIEW 视图名称 [RESTRICT | CASCADE]

执行方式标识:

RESTRICT	仅删除视图，视图依赖的对象不受影响
CASCADE	删除视图及其依赖的所有相关对象

视图的局限性： 不能对基于聚集函数的列 在创建视图的select子句中再使用聚集函数

创建视图举例

```
CREATE VIEW Staff3
    AS SELECT staffNo, fName, lName, position, sex
        FROM Staff
        WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

```
    CONSTRAINT StaffNotHandlingTooMuch
        CHECK ( NOT EXISTS ( SELECT staffNo FROM
        propertyForRent GROUP BY staffNo
                                    HAVING COUNT(*)>100)
                ),
```

		定义一个约束名称为StaffNotHandlingTooMuch，要求该列的值在查询PropertyForRent（即本表自己）中的结果总数不超过100	即不允许同一个员工同时处理100个及以上的房屋
branchNo BranchNumber NOT NULL,	branchNo	非空	
PRIMARY KEY (propertyNo),			指定主键propertyNo
FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL ON UPDATE CASCADE		当更新Staff表中含staffNo的行时，同时更新该表中对应的staffNo 当删除Staff表中含staffNo的行时，将该表中对应的staffNo设置为NULL	设置外部关键字staffNo

);

视图的可更新性：只有满足以下条件是，才能对视图进行更新（INSERT UPDATE 和 DELETE）

- 没有指定DISTINCT
- 定义中的SELECT子语句 列表中每个元素均为列名，且出现次数不多于一次
- 定义中的SELECT子语句 中没有GROUP BY 和 HAVING 子句
- WHERE子句中不能包括任何引用了FROM子句中的表 ，即不允许嵌套SELECT操作
- FROM 子句 只指定一个表

事务

事务是指 一段SQL语句在执行期间的 总称

其由执行事务初始化的SQL语句（如 SELECT INSERT UPDATE）触发开始

由几种方式触发结束：

COMMIT 语句提交事务，或程序式SQL成功终止

ROLLBACK 语句撤销事务，或程序式SQL异常终止

事务在结束之前 其操作数据库的变化，对其他并发的事务均不可见

配置SQL事务的属性：

```
SET TRANSACTION [READ ONLY | READ WRITE]
    [ ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED |
      REPEATABLE READ | SERIALIZABLE ]
```

读写标识符：

READ ONLY	READ WRITE
只读	读和写

隔离级：

READ UNCOMMITTED	允许污读、重读和幻读
READ COMMITTED	允许重读和幻读，不允许污读
REPEATABLE READ	允许幻读，不允许重读和污读
SERIALIZABLE	不允许污读、重读和幻读

污读：读取其他事务正在修改却未提交的数据
重读：重新读取已经被其他事务修改和提交过的数据
幻读：事务在不同时间执行同一查询时，会补充查询到 在此期间 其他事务已经增加并提交的 行插入操作

修改SQL语句完整性约束的检查时间：

```
SET CONSTRAINTS {ALL | 约束规则名称 ,... } {DEFERRED | IMMEDIATE}
```

IMMEDIATE	DEFERRED
立刻检查	延迟检查到事务提交时

权限控制

授予权限：

```
GRANT {权限列表 | ALL PRIVILEGES} ON 对象名称 TO { 被授权人列表 | PUBLIC} [WITH GRANT OPTION]
```

对象 可以是 数据库、表、视图等

权限列表里的权限标识(用逗号隔开):

SELECT	查询权限
INSERT [列名,...]	插入权限
UPDATE [列名,...]	更新权限
DELETE	删除权限
REFERENCES [列名,...]	引用指定表的列
USAGE	使用域、序列、字符集和转变规则
ALL PRIVILEGES	所有权限

被授权人标识:

PUBLIC	允许现在和未来的所有用户获得权限

额外可选标识：

WITH GRANT OPINION	允许被授权人将权限再转移给其他用户

撤销权限:

```
REVOKE [GRANT OPTION FOR] {权限列表 | ALL PRIVILEGES}
    ON 对象名称 FROM {授权身份列表 | PUBLIC} [RESTRICT | CASCADE]
```

执行方式标识:

RESTRICT	仅撤销权限
CASCADE	不仅撤销权限，而且删除所有之前通过授权创建的对象或其声明

额外可选标识：

GRANT OPINION FOR	允许被授权人传递的权限 独立地被撤销

第七章 数据库规划、设计和管理

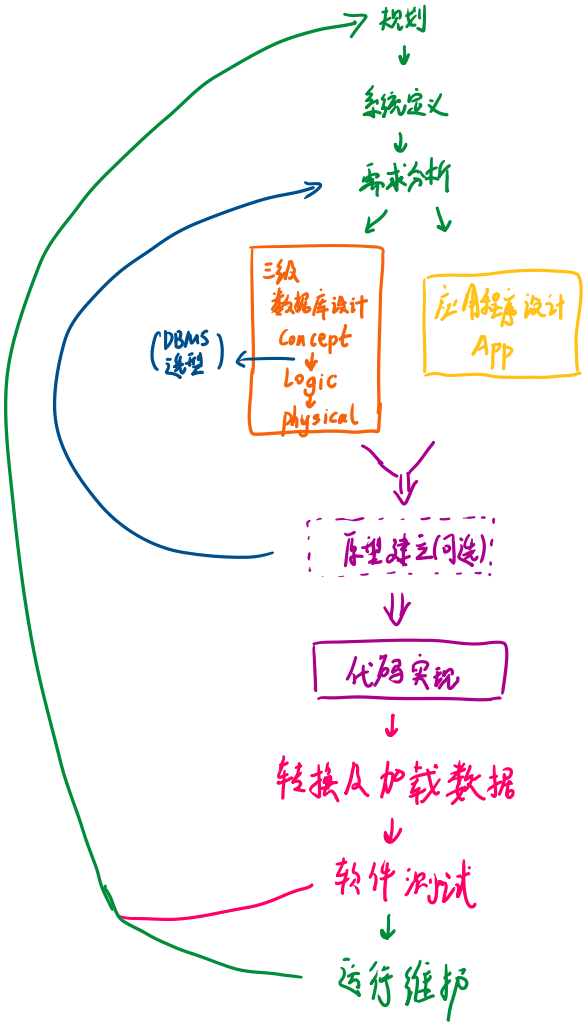
2021年5月6日 9:31

数据库是一种信息系统。

信息系统Information System 的定义：

在组织机构内用于收集、管理、控制和分发信息的所有资源

数据库系统的开发生命周期



规划 Database Planning	数据库规划的目的是: 尽可能高效及有效地展开开发生命周期的各个阶段 规划的步骤： 1 确定项目的任务描述 mission statement 2 确定任务目标 mission objective 3 建立相关标准、收集数据的方法和格式等其他内容
系统定义 System Definition	描述数据库的范围和边界，以及主要的用户视图 • 用户视图User views：从一个特定用户（或用户群体）的角度定义的数据库的需求 每一个用户视图都应该是整个数据库视图的子集
需求收集和分析 Require-ments Collection and Analysis	收集用户对于数据库系统的需求和要求，在这一阶段经常需要处理多个用户视图，常用的方法有（两种方法组合使用）： • 集中式方法 Centralized：合并所有视图的需求，形成一个统一需求，然后构建一个 全局数据模型global data model • 视图集成方法 View intergration：对每个用户的需求独立列出，然后设计若干独立的 局部数据模型local data model，最后加以整合形成全局数据模型
数据库设计 Database Design	设计满足组织机构的任务描述和任务目标的数据库的过程 **数据库设计的四种方案：** • 自底而上 Bottom-up：从基础属性开始设计，然后形成关系 • 自上而下 Top-down：先设计顶层的关系模型，再一步步设计下层的关系，直到确认每个关系的属性 • 由内而外 Inside-out：使用自底而上的方法，但是一开始只涉及一些主要实体的关系，设计好这些关系后再根据它们设计与其相关的实体的关系 • 混合式 Mixed：结合自底而上和自上而下的方法（两面包夹芝士） **数据库设计的步骤：** 1.概念设计：建立概念数据模型的过程，该模型与所有物理因素无关（使用的方法为：ER模型） 2.逻辑设计：根据概念数据模型模型，建立逻辑数据模型，逻辑数据模型与具体的DBMS无关 （使用的方法为：规范化） 3.物理设计：产生数据库在存储上实现的具体描述的过程，定义所有的基础关系，文件组织方式及索引，并包含所有完整性约束及安全措施
DBMS选型 DBMS selection	选择一种合适的DBMS作为数据库应用的载体 DBMS选型的主要步骤： • 确定研究考虑的方面 • 列出几个候选的DBMS产品 • 评估这些产品 • 给出建议产品选型的报告
应用程序设计 Application Design	设计数据库程序的UI界面，数据处理功能等 包含两个主要部分： • 事务设计：保障数据库能支持所系统涉及的事务处理 • 用户界面设计：[参见人机交互] (yt：哎~这我能讲一年)
原型建立 Prototyping	为数据库程序构建一个 虚拟的工作模型，以便开发测试程序 建立原型系统的常用策略： 需求原型 Requirements Prototyping：只确定数据库的需求，确定以后就不再使用。 进化原型 Evolutionary Prototyping：确定需求后在原型的基础上修改完善构成完整的数据库系统
实现 Implement ation	写代码（物理） 使用 所选DBMS的DDL 或使用 GUI图形界面 实现 应用程序APP可以用3GL或4GL语言实现，并且应该加入一些实用工具，如开始菜单，数据表输入器和报表生成器等
数据转换和	Data Conversion and Loading

CASE工具

CASE 指 计算机辅助软件工程 Computer-Aided Software Engineering

CASE的提供的支持有：

- 存储关于数据库系统数据信息的数据字典
- 数据分析的设计工具
- 企业数据模型、概念数据模型和逻辑数据模型开发的工具
- 建立原型系统的工具

CASE工具的分类：

- 上层工具（Upper-CASE）：支持从规划到设计的前期工作
- 底层工具（Lower-CASE）：支持实现、测试和维护的后期工作
- 集成工具：支持所有工作，即以上两种的结合

CASE的优点：

- 标准化

- 集成工具：支持所有工作，即以上两种的结合

CASE的优点：

- 标准化
- 集成化
- 支持标准化方法
- 保持一致性和方便进行一致性检查
- 自动化

[illegible]	应用程序APP可以用3GL或4GL语言实现，并且应该加入一些实用工具，如开始菜单，数据表输入器和报表生成器等
数据转换和加载	Data Conversion and Loading 将数据加载到实现好的数据库中，并将在以前的数据库上允许的数据管理程序（如果有）转移到新的DBMS中
软件测试	Testing 运行数据库系统，并试图找出错误 （yyy：哎~这我能讲一年 ）*2
运行维护	Operational Maintenance 实时监控并维持数据库程序正常运行

第八章 实体-联系（ER）建模

2021年5月18日 13:07

建模的目的：科学地规划和设计数据库对象的关系结构

实体Entity：

实体类型：代表现实世界中具有相同属性的一组对象。对象可以是物理存在的，也可以是抽象虚拟的。

实体出现 occurrence：每个实体类型中指定的一个唯一可被标识的对象

在没有歧义的情况下，实体类型和实体出现统称为实体。

实体类型的分类：

- 强实体类型 StrongEntityType：不依赖于其他实体类型存在
- 弱实体类型 WeakEntityType：依赖于其他实体类型存在

联系Relationship：实体类型间一组有意义的关联

同理，如果参与联系的都是实体出现，那么称该联系为"联系的实例出现"

参与联系的实体类型的数量，称为联系类型的度

递归联系：同一个实体类型在一个联系中出现了多次（大于1次），且参与的角色不同

联系的方式（结构化约束）：

联系的主要约束又称 多重性 multiplicity

多重性约束的定义：

- 二元多重性定义：参与联系的两个实体类型中，实体类型和联系类型之间对应关系的数量（或范围）
- n元多重性（又称 复杂联系）：参与联系的n个实体类型中，当其他$n-1$个实体类型的值固定以后，剩下的实体类型与参与联系类型实例 之间对应关系的数量（或范围）

多重性约束的内容:

- 基数约束Cardinality：实体能参与联系的最大数目，如（二元为例）：
  - 一对一（1：1）
  - 一对多（1：*）
  - 多对多（*：*）
- 参与性约束Participation：所有实体是否都参与了联系：
  - 是=>强制 参与
  - 否=>可选 参与

属性Attribute：实体或联系类型所具有的某一特征

属性的域Domain：对于一个或多个属性 的可被允许的 值的集合

属性的分类法：

一：

简单属性：独立的，不可再分的，又称原子属性

复杂属性：由多个部分组成，每个部分都可以独立存在

二：

单值属性：实体类型的每个 实例出现 都有且只有一个值的属性

多值属性：对于实体类型的某些 实例出现 可能会有多个值的属性

- ex:

导出属性Dervied attribute：属性的值由相关的一个或一组属性构成

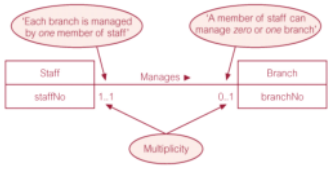
由属性定义关键字：

- 候选关键字：唯一确定 一个实体类型中任意实体出现 的最小属性集合
- 主关键字：从所有候选关键字中指定的 属性集合
- 复合关键字: 如果候选关键字集合里有两个或以上的属性，即称之为复合关键字

ER模型的问题：

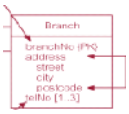
连接陷阱 Connection trap

- 扇形陷阱 fan trap：ER模型给出了两个实体类型之间的一种联系，但在某些实体出现之间存在着多条通路
- 断层陷阱 chasm trap：ER模型显示某些实体类型之间存在着联系，但某些实体出现之间却不存在通路



绘制ER模型对应的UML图

实体：用一个类似表格的方框表示，第一行填写实体类型的名称(首字母大写)，下方填写其属性



属性：写在对应实体的表格里

主关键字，后面注明 {PK}

复杂属性，用悬挂缩进的方式写明子属性

多值属性，后面注明 [x..y]，表示其允许的值的数量为x到y

导出属性，在名称前加上 / 以区分

联系：用实体间的连线表示，关系的名称（和方向）写在连线上，多重性约束写在连线的两边。

如A-B的联系，A对B的多重性约束写在B侧，B对A的多重性约束写在A侧，用 n..n来代替*：*这样的写法

参与性约束： 0 .. 1 可选参与

1 .. 1 强制参与

基数约束：

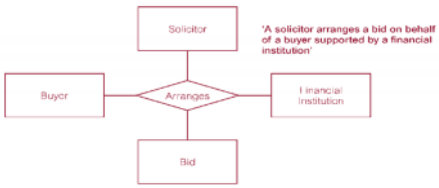
一对一：1 .. 1

一对多：1 .. n

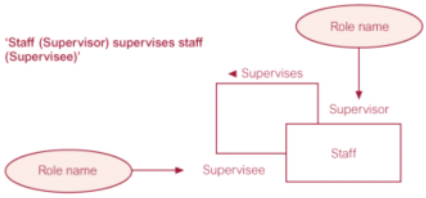
多对一：n .. n

(n可以为指定值)

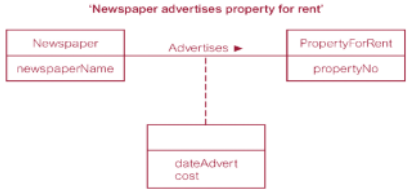
复杂联系：联系名称写在菱形框中，并与所有实体连接



递归联系：用指向自己的连线表示，双向不同的角色名称写在连线的两端



联系如果具有属性，可以用虚线连接一个表格框表示：



根据ER模型图生成逻辑数据模型

确定好完整的ER图后，按照以下步骤进行

1 处理强实体类型：

对于每个强实体，创建一个包含该实体的所有 简单属性 的关系

2 处理弱实体类型：

对于每个弱实体，创建一个包含该实体的所有 简单属性 的关系

在这一步创建的关系，不要设定主关键字

（处理了，但还没有完全处理）

3 处理 一对多（1：*）二元联系：

"一"的那一方称为父实体，"多"的那一方称为子实体。

将父实体的主关键字拷贝一份，放入子实体的关系中，

即每个子实体会增加一个外部关键字。

4 处理 一对一（1：1）二元联系：

根据参与性判断选择以下操作其一：

- 双方都强制参与（1..1）

  将联系中的 两个实体已经拥有的关系 **合并**到一个关系里，自主选择一方关系的主关键字作为新的主关键字
- 一方强制参与（1..0 / 0..1）

  认为可选参与的一方是父实体（即0为父实体，1为子实体），采用和步骤3一样的处理方案

  （现在知道0和1里面那个才是爸爸了吧）
- 双方都可选参与（0..0） （这个表情包好可爱）

  自主决定父子实体，有时候要根据实体的实际情况和方便后面的关系处理而指定父实体。然后采用和步骤3一样的处理方案

5 处理 一对一 递归联系：

根据参与性判断选择以下操作其一：

- 双方都强制参与（1..1）

  创建一个新的关系，将该实体的主关键字复制一份，重命名（改成能表示联系的含义的名称），然后将 主关键字 和 改名后的主关键字副本（设置为外部关键字） 加入这个新的关系
- 其他情况（1..0 / 0..1 / 0..0）

  可以采取与1..1一样的方案，也可以创建另一个新的关系：将原实体的主关键字复制两份，都重命名为表示关系含义的名称，然后加入新的关系，新的关系将只含两个外部关键字

6 处理 多对多（*：*）二元联系：

创建一个新的关系，将 参与这个二元联系的所有实体的主关键字 和 参与这个二元联系的所有属性 都复制一份加入新关系。新加入的主关键字变为外部关键字，从这些 外部关键字 或 （外部关键字+联系本身的属性）[这表示一个复合关键字] 中指定新关系的主关键字。

7 处理复杂联系

创建一个新的关系，将 参与该联系的所有实体的主关键字 和 参与该联系的所有属性 都复制一份加入新关系。新加入的主关键字变为外部关键字，从 自联系的"多"方拷贝来的外部关键字 或 （这些外部关键字+联系本身的属性）中指定新关系的主关键字。

注意：对于3-7，如果该联系本身也含有某些属性，这些属性也应该加入 子实体/合并后的实体/新创建的实体 的关系中。

8 处理多值属性

实体中每有一个多值属性，创建一个新的关系

将拥有多值属性的实体的主关键字复制到新的关系中，成为外部关键字。新属性的主关键字是 （多值属性+原实体的主关键字）的复合关键字（除非该多值属性原来是可替换的关键字）

简单的表格 总结一下对联系部分的处理（3-7）：

第八章 实体-联系（ER）建模 - 电子表格

	二元联系				复杂联系
	一对一	一对一（递归）	一对多	多对多	
1..1	合并两个实体的关系	新建关系，主关键字拷贝两份，一个改名变为外部关键字	父实体主关键字拷贝到子实体	新建关系，加入所有参与实体的主关键字和属性，然后再选择一个或组合一个作为新的主关键字	和左边这个一样，只不过选主关键字的时候只考虑从0/1..*方向加入的主关键字
1..0/0..1	0父实体，1子实体	新建关系，主关键字拷贝两份，两个都改名变为外部关键字			
0..0	父子实体任选				

第九章 规范化

2021年5月18日 14:30

规范化Normalization的目的：确定最佳的的满足需求的关系结构（类比于程序优化）

最佳的判定：

- ✓ 使用属性的数量最少
- ✓ 逻辑联系紧密的属性都在同一个关系中
- ✓ 冗余属性最少，或没有，即属性最多只出现一次

数据冗余 data redundancy：主关键字或候选关键字出现在多个表中（即外部关键字），数据冗余用来标识数据库之间的联系。

数据冗余可能带来的问题：更新异常 update anomalies：

- 插入异常：插入元组时，无法满足所有表中冗余数据的完整性约束而导致异常
- 删除异常：删除元组时，导致该元组相关的冗余数据信息丢失
- 修改异常：更新元组时，如果没有及时更新相关联的表，会导致冗余数据更新不一致。

函数依赖 Funcional Dependency：描述同一个关系中属性之间的联系

A -> B：B函数依赖于A，或A决定B，则A属性中的所有值都和B属性中的一一对应。

完全函数依赖：A -> B，且B不依赖于A的任何真子集，则称B完全函数依赖于A

传递函数依赖：A -> B, B -> C，称C通过B传递依赖于A

函数依赖的推导规则， 阿姆斯特朗公理Armstrong's axioms：

设A,B,C是关系R的属性，则：

- 若 B 是 A 的子集，则 A -> B （自反性）
- 若 A -> B，则 A,C -> B,C （增广性）
- 若 A -> B 且 B -> C，则 A ->C （传递性）

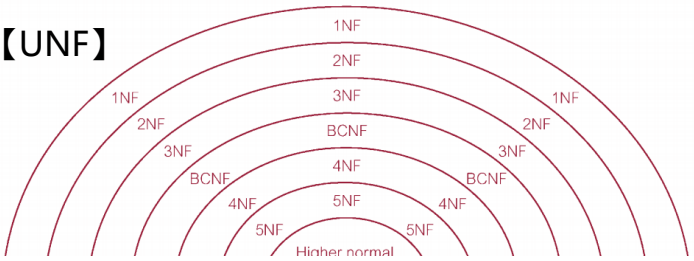
对一个函数依赖的集合X，其中所有的子集通过以上公理生成的（蕴含的）函数依赖，称为X的闭包，记为X^+

如果函数依赖Y 的每一个函数依赖 都是 X^+中的元素，则称Y被X覆盖，即Y中的每个依赖都能从X中导出

函数依赖的作用：确定（最小化的）主关键字

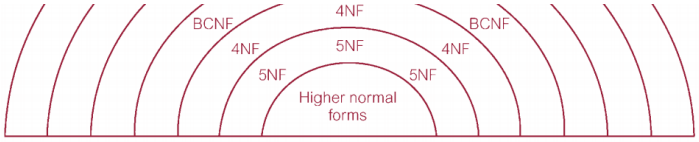
规范化过程

- ➢ 将属性（数据源）转化为表，确定依赖关系 【UNF】
- ➢ 移除重复组 【1NF】
- ➢ 移除部分依赖 【2NF】
- ➢ 移除传递依赖 【3NF】



- 移除部分依赖 【2NF】
- 移除传递依赖 【3NF】

  ……【更高范式】



规范化过程中的范式：规范化过程的每一步，都对应着某种已知性质的特定范式

非范式 UNF：包含重复组的表

第一范式 1NF：不包含重复组的表，即每行和每列的交点值有且仅有唯一值

第二范式 2NF：不包含重复组的表，且每个非主关键字的属性都 完全函数依赖于 主关键字的关系

第三范式 3NF：满足第二范式的要求，且每个非主关键字属性都不 传递依赖 于 主关键字的关系

（一般化的范式定义，只需将以上定义中的 主关键字 更改为 候选关键字）

Boyce-Codd范式 BCNF：满足第三范式的要求，而且所有函数依赖的决定方都是候选关键字（即对与任意A->B，A必须为候选关键字）

规范化过程的具体操作：都是关系代数投影，即无损连接，其逆过程为自然连接

- 从UNF -> 1NF：将非规范化的表中消除重复组的方法：
  1 平板化处理：将含有重复数据的行分离（复制几行），并在每行的对应列上填上合适的数据
  2 将重复的数据移植到一个新的关系中，并将原来关系中的 关键属性 复制到新关系中
  3 重复1-2步，直到所有表中都不含有重复组
- 从1NF -> 2NF：消除部分函数依赖的方法：
  1 找到所有的部分函数依赖（其满足 A->B, A'->B 且 A' ⊆A）
  2 将部分依赖的属性（组）B从原关系中移出，转移到一个新的关系。同时，若有其他的属性决定该部份依赖的属性A（A一定是主/候选关键字），将它们复制到新的关系中
- 从2NF -> 3NF：消除传递依赖的方法：
  1 找出所有传递依赖（其满足 A->B, B->C）
  2 将传递依赖的属性（组）B从原关系中移除，转移到一个新的关系，并将决定该属性的主/候选关键字A也复制到该关系中

从UNF规范到BNCF的规范化过程举例：参考教材p337-p341

第十章 事务管理

2021年5月27日 8:01

事务： 由单个用户或应用程序执行的，完成读取或更新数据库内容的一个或一串操作

事务是数据库的 逻辑操作单位

事务的四大特性：ACID

Atomicity：原子性

事务是一个不可分割的单元，事务里的所有操作，要么都执行，要么都不执行

Consistency：一致性

事务必须将数据库从一种一致的状态转换到另一种一致的状态

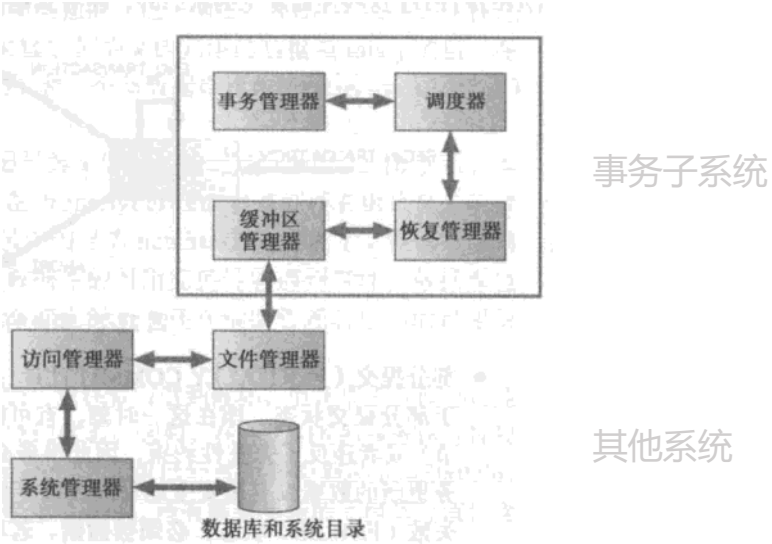
Isolation：可隔离性

不同事务的执行是相互独立的，一个事务不能看到另外一个正在执行中的事务处理的中间结果

Durability：持久性

成功提交的事务结果需要永久存储，并且在出现出现故障时可以随时恢复

DBMS事务子系统的结构：



事务的并发控制

当多个事务并发执行时，可能导致以下问题：

- 丢失更新问题：一个是事务的更新操作结果被另一个事务的操作结果取代
- 未提交依赖（污读）问题：一个事务读取了另一个事务操作中的数据，而另一个事务后来回滚撤销了操作，导致数据差别
- 不一致分析问题：
  - 不一致读取（不可重读）：在一个事务执行期间，需要多次读取同一数据，但由于受到另一个事务对数据修改的影响，导致该事务执行中读到的数据不一致
  - 不一致查询（幻读）：在一个事务执行期间，需要多次查询数据库，但由于受到另一个事务对数据库修改的影响，导致该事务执行中查询到的元组结果不一致

解决并发控制问题：

调度：确定所有事务的操作次序，以形成一个执行列表

调度的分类

- 串行调度：所有事务都按照顺序执行，没有任何交叉
- 非串行调度：事务的操作可以交叉执行的调度

可串行化： 对于一组特定的事务，其按照（一定的）顺序执行可得到一组结果A。当这些事务按照特定的非串行调度执行时，也得到同样的结果A，则称该非串行调度是 **可串行化的**

即可串行化的调度可以在并发执行的基础上，满足和串行执行一样的一致性

可串行化的具体分类：

- 冲突可串行化：对事务中的冲突操作执行顺序，其满足可串行化的条件。
  - 冲突操作：两个事务需要读或写同一个数据项
- 视图可串行化：事务的 调度视图 等价于串行调度。
  - 调度视图的等价性：对于两个调度序列，其满足以下条件：
    - 对于两个调度序列里事务操作，其读取同一个数据的初值 的操作相对应，修改同一个数据 的操作相对应，最后写操作同一个数据 的操作相对应

如果一组调度是 冲突可串行化 的，则其一定是 视图可串行化的，反之不成立。

可恢复性：对于调度中的每一对事务A和B，如果B读取了A的修改过的数据，则A必须在B之前提交操作

满足可恢复性的调度称为 **可恢复调度**，否则称为不可恢复调度

加锁方法（类似进程调度）：

锁 Locking：当一个事务正在访问数据库时，拒绝其他事务的访问请求

加锁类型

共享锁：具有共享锁的数据项只能被加锁的事务 读 而不能 修改

互斥锁：具有互斥锁的数据项可以被加锁的事务 读 或 修改

两段锁协议（2PL）：事务中所有的加锁操作出现在第一个解锁操作之前

两段锁协议要求事务包含两个阶段：

- 扩展阶段：事务获取它所需的所有锁（一般来说，事务要对什么数据项进行操作，就应该获取谁的锁），但不能释放锁
- 收缩阶段：事务可以释放其拥有的锁，但不能再获得更多的锁

2PL解决三个并发控制问题：

- 解决 丢失更新 和 未提交依赖：只需要先进行操作的事务加锁，后进行的事务在扩展阶段等待锁的释放即可
- 解决 不一致分析操作：进行读取或查询操作的事务必须先对数据加共享锁，而进行修改操作的事务必须先对数据加互斥锁；对同一个数据对象，后进行加锁操作的事务必须等待

级联回滚问题：如果先加锁的事务失败导致回滚，后面所有试图在该锁上进行操作的事务都需要回滚，导致的连锁回滚现象

避免级联回滚：

严格2PL：所有事务必须在完成后（提交结果之后）才能释放锁

如果所有事务都遵循严格2PL，则它们的调度一定满足按照提交顺序的串行调度

弱严格2PL：所有事务必须在完成后（提交结果之后）才能释放互斥锁

死锁问题：

死锁：两个或多个事务互相等待对方释放自己已经占有的锁

解决死锁问题的方法：

- 超时机制：为等待加锁的事务设置一定的等待时限，超时即撤销该事务并自动重启。回滚事务并重启不会对数据库产生什么严重的影响，因此超时方法简单易用，这和操作系统的死锁解决方案不同。
- 死锁预防：

  1、为事务加上时间戳，使其有序进行。（看下面的时间戳方法）

  2、采用保守的两段锁（conservative 2PL）：事务必须在开始之前获得所有的锁，否则就等待
- 死锁检测和恢复：

通过构造事务之间依赖关系的等待图 进行死锁检测（类似操作系统的死锁检测），检测到死锁后，DBMS撤销一个或多个事务，解决死锁。

撤销事务时，DBMS要注意防止某些事务饥饿，和尽可能用事务回滚代替撤销。

时间戳方法

时间戳：由DBMS创建的，标识事务相对启动时间的，具有唯一性的标识符

时间戳并发控制协议：按照以下规则确定事务的顺序（**基本时间戳排序**）

- 为每个数据项分配两个时间戳：读时间戳R，写时间戳W，默认值为最小值（可以理解为负无穷）
- 每个事务开始时，为其分配一个时间戳TS
- 满足以下条件的操作被允许：
  - 当事务要求读数据项时，其时间戳TS必须不小于数据项的写时间戳W
  - 当事务要求写数据项时，其时间戳TS必须不小于数据项的所有时间戳R和W
- 当有事务读或写数据项成功时，将数据项的R或W更改为对应事务的TS值
- 当有事务的操作被拒绝时，将其回滚到操作前的情况，并更新其时间戳到最新时间，然后重启事务

按照基本时间戳排序的规则进行的调度 一定是 冲突可串行的

多粒度锁方案

粒度 Granularity：受到保护的数据项的大小（包括的范围）

粗粒度：整个数据库，一个文件，一页等

细粒度：一条记录，一条记录里一个字段的值等

粗粒度包含细粒度，因此可以用树结构将数据库的粒度层次存储起来

多粒度加锁：当事务操作的数据项位于不同粒度时的加锁方案，满足以下规则：

- 当某一层粒度节点的数据项被加锁时，其所有祖先都要被加上 **意向锁 Intention lock**，根据加锁的方式，分为 意向互斥锁 和 意向共享锁
- 更改两段锁协议为：
  - 一旦有节点被解锁，就不再继续加锁
  - 只有在父节点有意向锁时，才能对其子节点加锁

    换句话说，想对子节点加锁，必须先对父节点加意向锁
  - 只有其所有子孙节点都被解锁时，父节点的意向锁才可以被解锁

    加锁自顶而下，解锁从底向上

锁之间的兼容性（√表示兼容，即某数据项是否可以被两个事务同时加上这两种锁）

	IS	S	IX	SIX	X
意向共享 (IS)	√	√	√	√	否
共享 (S)	√	√	否	否	否
意向互斥 (IX)	√	否	√	否	否
意向互斥共享 (SIX)	√	否	否	否	否
互斥 (X)	否	否	否	否	否

其中，意向互斥共享锁SIX 表示 先对数据项加共享锁（S），释放后再加IX锁

数据库恢复

定义：发生故障时，将数据库还原到正确的状态

数据库故障的原因：

- 系统崩溃
- 介质故障
- 应用软件错误
- 自然物理灾害
- 人为破坏（蓄意或无意）

事务是数据库恢复的基本单位

数据恢复时对事务的操作情况：

- 故障前事务已提交但未写入存储：重做事务（redo/rollforward）
- 故障前事务未提交：撤销事务（undo/rollback）

数据恢复机制：

- 备份机制：定期拷贝数据库和日志文件到其他存储介质
- 日志机制：事务执行时随时记录日志，日志中包括事务记录和检查点记录
- 检查点机制：主动定义检查点来方便确定恢复的范围
  - 检查点：数据库与事务日志文件之间的同步点，此时所有缓冲区数据都要强制被写入存储介质
- 恢复管理器：管理数据库的恢复过程

数据恢复技术：

- 延迟修改 Deferred Update：只有当事务提交成功后，才写入数据到数据库。如果发生故障，只要当做什么都没发生。

  “提前写”日志协议 write-ahead log protocol：

  - 事务执行前 写入事务开始到 日志
  - 事务执行过程中 写一份临时的日志，不放到真正的日志里
  - 事务提交后，写入事务提交以及临时日志里的所有内容到真日志里
- 立即修改 Immediate Update：更新一旦发生就立即修改数据库。如果发生故障，就撤销故障前的所有修改。
  - 随时写入事务执行的操作到日志里，粒度越细越好
- 影像页 Shadow Paging：事务启动时复制一份影像页表（操作系统内存级别的备份），其在事务执行过程中不再修改，以便恢复。

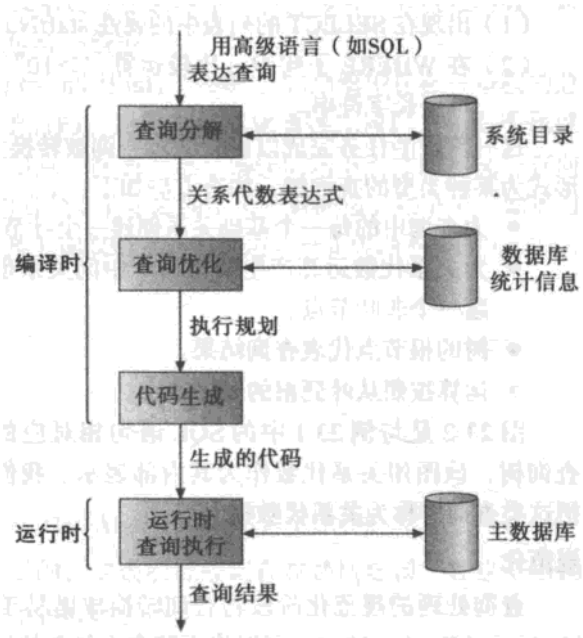
第十一章 查询处理

2021年6月17日 8:00

查询处理 Query Processing，包括语法分析、正确性验证、查询优化以及查询执行等活动，其中最重要的是查询优化。

查询优化 Query optimization：选择最佳的执行策略进行查询处理的过程

查询优化的目标：选择资源占用最少的等价查询形式，资源占用最少可以从 所有运算执行时间的总和 或 查询的响应时间 方面来量化



查询处理的四个主要阶段：

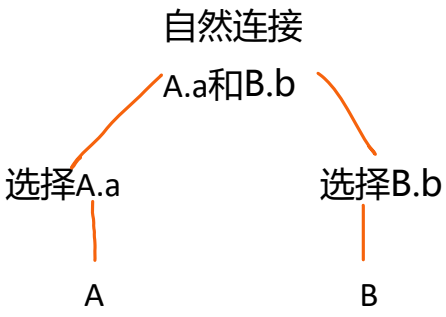
- 分解 decompostion：
  验证高级语言的语法，分析结构
- 优化 optimization：
  选择最佳的等价关系代数表达式
- 代码生成：code generation
  生成优化后的数据库执行代码
- 代码执行：execution

查询分解 / 查询解析

step1 验证语法是否正确，若否，拒绝执行

step2 创建查询树：

- 为查询中的每一个基础关系创建一个叶子节点
- 为每个关系代数运算产生的中间关系创建一个非叶子节点
- 最终查询结构作为根节点，连接各个节点
  这样就形成了关系代数查询树 telational algebra tree，语句的运算按照从叶到根的顺序执行



step3 规范化：将表达式里的谓词（SQL中的WHERE子句）转换为以下两种范式之一：

- 合取范式 (a ∧ b) ∨ (c ∧ d) ∨ ...

- 析取范式 (a ∨ b) ∧ (c ∨ d) ∧ ...

  没错这和离散相同

step 4 语义分析：判断是否有自相矛盾的谓词，简化它们为True或Flase

step 5 化简：根据布尔运算规律，去除公共子表达式；并考虑数据库的完整性约束、访问限制、视图定义等的语句要求

查询优化过程

两种主要的查询优化技术（常结合使用）：

1、启发式规则 heuristic rules：将语句通过关系代数表达式转换为更加高效的等价形式执行

2、比较各种策略的相对资源消耗，并选择资源消耗最少的方案

关系代数转换规则：

i. 串联式规则，合取选择运算转换为单个选择的运算的串联
ii. 交换选择运算的顺序
iii. 投影运算的串联与只执行最后一个投影运算等价
iv. 选择运算和投影运算的交换律（谓词相同的情况下）
v. θ连接/笛卡尔积运算的交换律、结合律
vi. 选择运算和 θ连接/笛卡尔积运算 的分配律
vii. 投影运算和 θ连接/笛卡尔积运算 的分配律
viii. 交运算和并运算的交换律、结合律
ix. 选择运算和 交/并/差运算 的分配律
x. 投影运算和 并运算 的分配律

启发式规则（关系代数转换规则）的策略总结:

- 尽早执行选择运算和投影运算
- 合并笛卡尔积运算和其后的选择运算为连接运算
- 利用二元运算的结合律对叶子节点进行重排序，条件越严格的选择运算，越先执行
- 如果有公共表达式在计算中多次出现，只执行一次其值的计算

关系代数运算的代价估算

关系代数运算的代价估算

前提：DBMS知道数据库的统计信息，即其中数据的规模

数据库的统计信息包含三类：（DBMS应周期性更新这些数据）

参数：R-关系，A-属性，I-索引

基本关系统计信息：

- nTuples(R)：关系R的元组数目（R的基数）
- bFactor(R)：关系R的块因子blocking factor，即一块可以存储R中元组的个数
- nBlocks(R)：存储关系R需要的块数，即以上两个参数之商

关系中的属性的统计信息：

- nDistinct(A,R)：属性A在关系R中不同取值的个数
- min(A,R)/max(A,R)：属性A在R中的可能最小值/最大值
- SC(A,R)：属性A在关系R中的选择基数 selection cardinality，即满足属性A的某个等值条件的平均元组个数，这需要DBMS根据实际情况估算

属性上多级索引的统计信息:

- nLevels(A,I)：I的索引级数
- nLfBlocks(A,I)：I中叶子的块数

通过这些数据，以及DBMS在各种关系运算操作中使用的算法，即可估算不同算法的运算代价的相对值

第十二章 数据库的安全性

2021年6月17日 8:00

数据库安全性的定义：保护数据库免受威胁的机制

威胁 Threat：任何有意或无意的，对系统或组织造成负面影响的行为或事件

安全性防御措施

1 授权 Authorization

授予一个主体权利或权限，使其能实现对系统或系统对象的合法访问。

认证 Authentication：确认用户身份是否属实的机制

权限 Privileges：包括访问特定数据库对象（如 关系、视图和索引）以及运行不同的DBMS系统实体

2 访问控制 Access contorl

自主访问控制（DAC）：拥有权限的用户通过 GRANT和 REVOKE 命令来控制用户权限

问题：未授权的用户可能利用授权用户，令其泄露机密数据

强制访问控制（MAC）：对数据库对象赋予安全级别（security class），并对每一个用户赋予访问许可级别（clearance）。系统根据用户的级别和访问对象的安全级别，通过特定的规则判断是否允许用户的读写操作。

SQL标准中仅包含对DAC的支持，MAC的支持一般为DBMS自行建立

3 视图

通过向某些用户隐藏数据库的一部分信息来限制用户

4 备份和恢复

- 备份 Backup：周期性将数据库、日志文件、DBMS程序（可选）复制到脱机存储介质上
- 日志 Log：日志文件包含数据库的所有变化记录。保存并维护日志文件，使得数据库出现故障时能够有效地进行恢复

5 完整性约束

防止非法数据的生成

6 加密 Encryption

使用特定算法对数据进行编码

- 对称加密：加密和解密密钥相同，因此密钥的交换也要保证安全性。

  对称加密要实现真正的安全，密钥理应和消息一样长

- 非对称加密：使用公钥和私钥，并且加密算法可以公开

7 RAID方法

对运行DBMS的硬件，尤其是磁盘驱动器，构建独立磁盘冗余阵列，保障硬件系统的安全性

（RAID 的具体内容参考操作系统-_-）