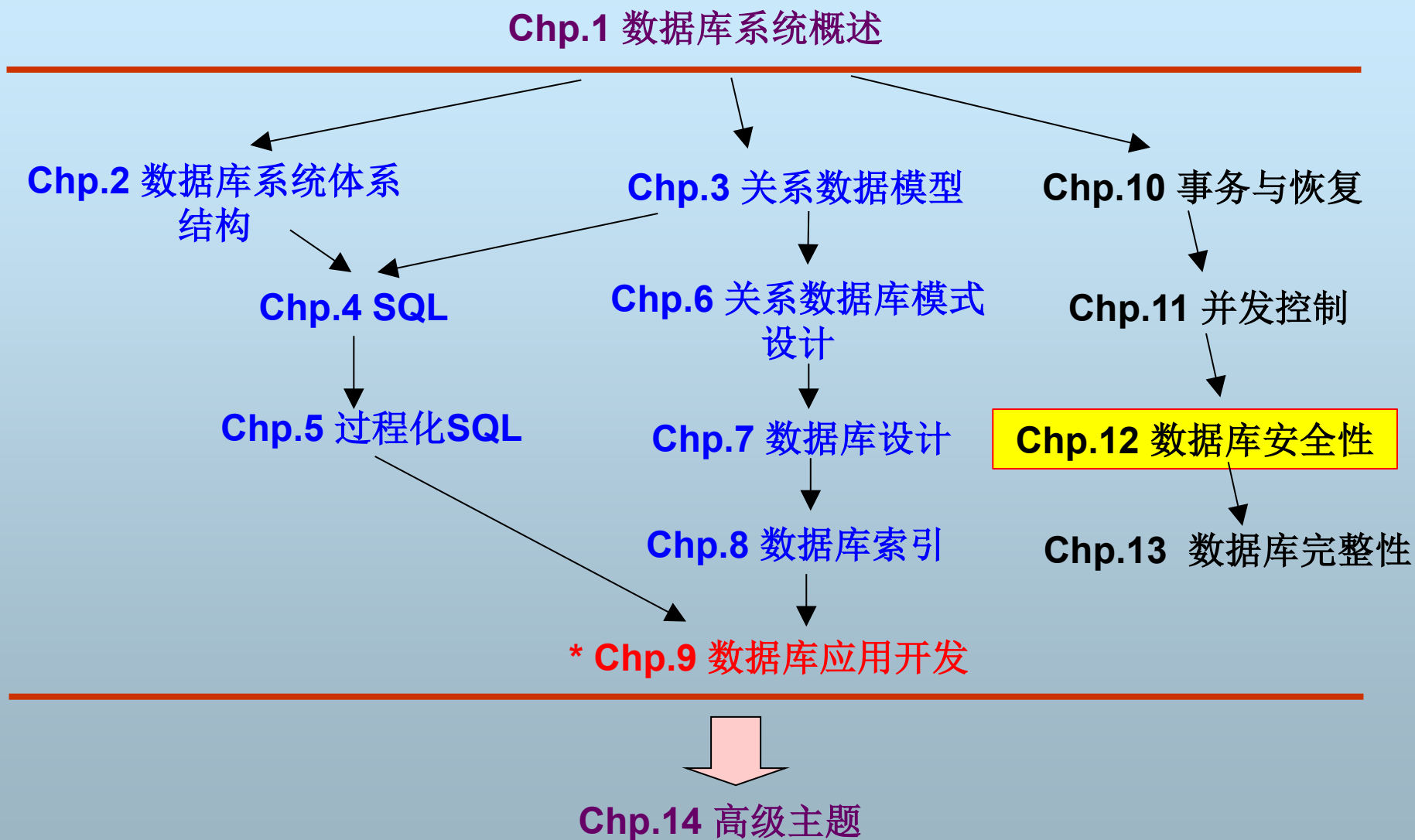


第12章 数据库安全性

课程知识结构



Databases Protection

- 数据库保护：排除和防止各种对数据库的干扰破坏，确保数据安全可靠，以及在数据库遭到破坏后尽快地恢复
- 数据库保护通过四个方面来实现
 - 完整性控制技术
 - ◆ Enable constraints
 - 安全性控制技术
 - ◆ Authorization and authentication
 - 数据库的恢复技术
 - ◆ Deal with failure
 - 并发控制技术
 - ◆ Deal with data sharing

主要内容

- 数据库安全性控制概述
- 自主存取控制
- 强制存取控制
- 视图机制

一、数据库安全性控制概述

■ 非法使用数据库的情况

- 用户编写一段合法的程序绕过**DBMS**及其授权机制，通过操作系统直接存取、修改或备份数据库中的数据；
- 直接或编写应用程序执行非授权操作；
- 通过多次合法查询从数据库中推导出一些保密数据

一、数据库安全性控制概述

- **例：[推理分析问题]** 某数据库应用系统禁止查询单个人的工资，但允许查任意一组人的平均工资。用户甲想了解张三的工资，于是他：
 - 首先查询包括张三在内的一组人的平均工资
 - 然后查用自己替换张三后这组人的平均工资
 - 从而推导出张三的工资

统计数据库：仅提供记录的统计信息查询，例如 **count/sum/average** 等，但不允许查询单独的记录。

一、数据库安全性控制概述

- **例: [SQL注入问题]** 如某系统需要你输入用户名称和口令, 现在假设有一用户admin, 我们不知道他的口令, 却想以他的身份登陆
- 正常情况下, 我们在第一个文本框输入admin, 第二个文本框输入1234之类的密码。程序中的查询语句可能是:
 - `sql="SELECT * FROM user WHERE username = ' " & text1.value & " ' AND passwd= ' " & text2.value & " ' "`
- 执行时候就是
 - `SELECT * FROM user WHERE username='admin' AND passwd='1234'`
- 如果我们在text2里输入的不是1234, 而是1234 ' OR 1=1 , 在执行时SQL语句就成了
 - `SELECT * FROM user WHERE username='adam' AND passwd='1234' OR 1=1`
 - 绕过了登录验证

一、数据库安全性控制概述

■ 数据库安全性控制的常用方法

- 用户标识和鉴定
- 加密存储

◆ MySQL对称加密函数 AES_ENCRYPT

```
1 INSERT INTO encrypted_data (data) VALUES  
2 (AES_ENCRYPT('sensitive data', 'secret key'));
```

```
1 SELECT id, AES_DECRYPT(data, 'secret key') AS  
2 decrypted_data FROM encrypted_data;
```

◆ MySQL非对称加密函数 RSA_ENCRYPT

```
1 SET @private_key = '';  
2 SET @public_key = '';  
3 SELECT RSA_NEWKEY(2048, @private_key, @public_key);  
4 SELECT @private_key, @public_key;
```

```
1 INSERT INTO encrypted_data (data) VALUES  
2 (RSA_ENCRYPT('sensitive data', @public_key));
```

```
1 SELECT id, RSA_DECRYPT(data, @private_key) AS  
2 decrypted_data FROM encrypted_data;
```

- 存取控制
- 视图
- 审计

二、存取控制

■ 存取控制（Access Control）机制的功能

● 授权（Authorization）

- ◆ 对每个用户定义存取权限

● 验证（Authentication）

- ◆ 对于通过鉴定获得上机权的用户（即合法用户），系统根据他的存取权限定义对他的各种操作请求进行控制，确保他只执行合法操作

● 授权和验证机制一起组成了DBMS的安全子系统

二、存取控制

■ 常用存取控制方法

- 自主存取控制（**Discretionary Access Control**，简称**DAC**）
 - ◆ C1级
 - ◆ 灵活
- 强制存取控制（**Mandatory Access Control**，简称 **MAC**）
 - ◆ B1级
 - ◆ 严格

1、数据安全的级别

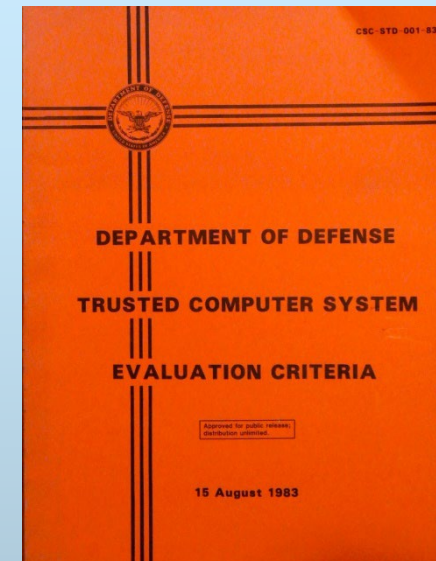
- 1983年，美国颁布可信计算机系统安全评测标准TCSEC (Trusted Computer System Evaluation Criteria)，把数据安全级别划分为四类七级：

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。



“橘皮书”

1、数据安全的级别

■ **1999年我国颁布了信息安全评估级别，共分为五级与美国标准的对应关系如下：**

- **第一级：用户自主保护级C1级**
- **第二级：系统审计保护级C2级**
- **第三级：安全标记保护级B1级**
- **第四级：结构化保护级B2级**
- **第五级：访问验证保护级B3级**

2、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

(1) 关系数据库系统中的存取权限

■ SQL中的存取权限定义方法

● GRANT/REVOKE

(2) 授予权限

■ 例子：授予语句权限

- 下面的示例给用户 **Mary** 和 **John** 授予多个语句权限。
 - ◆ **GRANT** *CREATE DATABASE, CREATE TABLE* **TO** *Mary, John*
- 授予全部语句权限给用户**Rose**
 - ◆ **GRANT** *ALL* **to** *Rose*

(3) 授予权限

■ 例子：授予对象权限

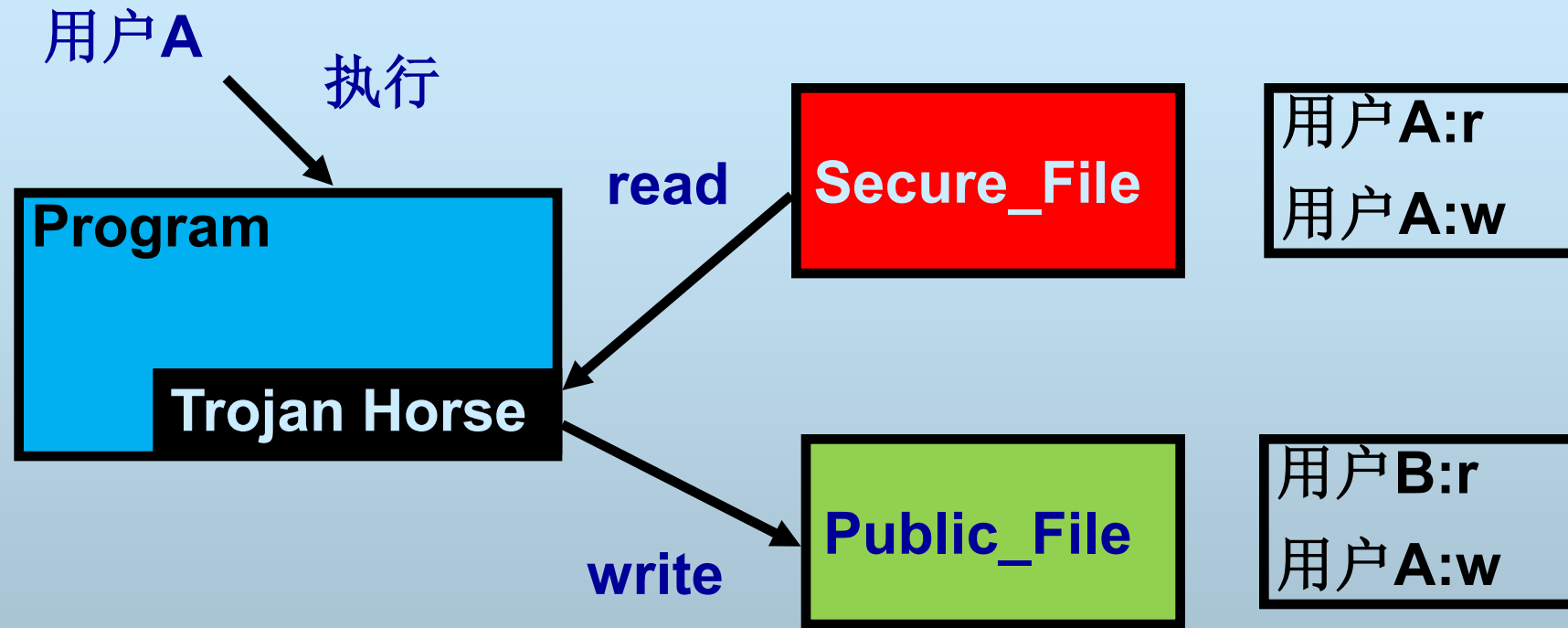
- **GRANT** *All ON authors* **TO** *userA*

- GRANT** *INSERT, UPDATE, DELETE ON authors* **TO** *Mary, John, Tom*

自主存取控制小结

- 定义存取权限
 - 用户
- 检查存取权限
 - DBMS
- 授权粒度
 - 数据对象粒度：数据库、表、属性列、行
- 优点
 - 能够通过授权机制有效地控制其他用户对敏感数据的存取
- 缺点
 - 可能存在数据的“无意泄露”：低级别用户访问到保密数据
 - ◆ 原因：这种机制仅仅通过对数据的存取权限来进行安全控制，而数据本身并无安全性标记
 - 无法防止木马信息窃取

自主存取控制不能防止木马



用户B通过Trojan Horse将Secure_File拷贝到Public_File中完成信息窃取

➡ **解决方法：** 对系统控制下的所有主客体实施**强制存取控制**策略

3、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

(1) 主体和客体

- 在MAC中，DBMS所管理的全部实体被分为**主体**和**客体**两大类
- 主体是系统中的活动实体
 - DBMS所管理的实际用户
 - 代表用户的各进程
- 客体是系统中的被动实体，是受主体操纵的
 - 文件
 - 基本表
 - 索引
 -

(1) 敏感度标记

- 对于主体和客体，DBMS为它们每个实例（值）指派一个敏感度标记（Label）
 - 主体的敏感度标记称为**存取级(Clearance Level)**
 - 客体的敏感度标记称为**密级(Classification Level)**
- 敏感度标记分成若干级别
 - 例如：绝密（**Top Secret**） / 机密（**Secret**） / 可信（**Confidential**） / 公开（**Public**）
- **MAC机制就是通过对比主体的Label和客体的Label，最终确定主体是否能够存取客体**

MAC常被称为“标签安全”或“标记安全”

(2) 强制存取控制规则

- 当某一用户（或某一主体）以标记label登录系统时，系统要求他对任何客体的存取必须遵循下面两条规则：——“下读上写”

- (1) 仅当主体的存取级别大于或等于客体的密级时，该主体才能读取相应的客体；
- (2) 仅当主体的存取级别等于（或小于）客体的密级时，该主体才能写相应的客体。

例如：市长（上级）可以查看各个厅局（下级）的文件，但各个厅局文件的修改必须由相应的厅局人员来完成

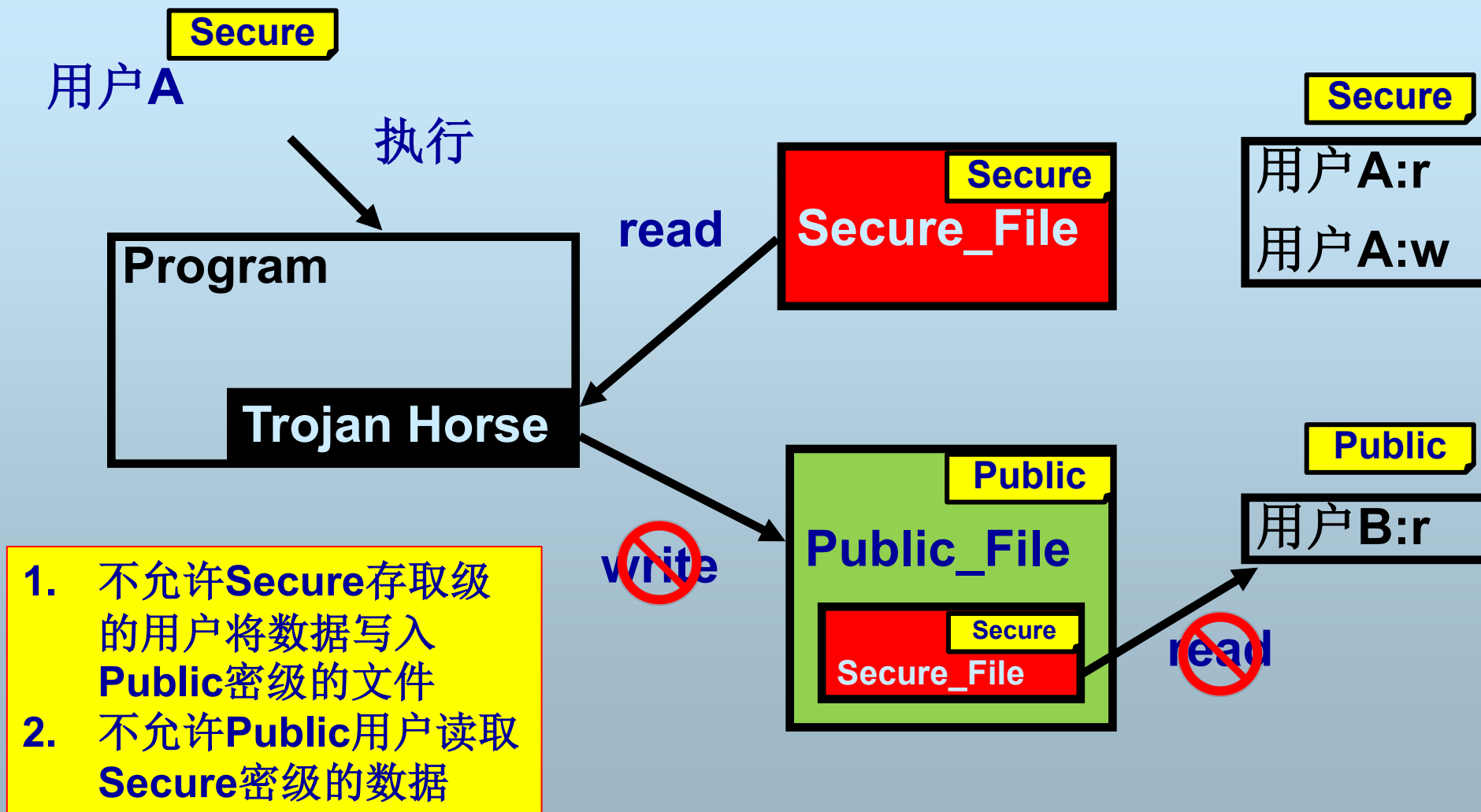
(3) 强制存取控制方法特点

- **MAC**是对数据本身进行密级标记
- 无论数据如何复制，标记与数据是不可分的整体
- 只有符合密级标记要求的用户才可以操纵数据
- 从而提供了更高级别的安全性



MAC可以防止木马攻击

(4) MAC 可以防止木马攻击



(5) Trusted Oracle中的MAC

■ Trusted Oracle: Oracle 7的MAC版本 (1991)

Example of a table in DBMS MAC mode:

ROWLABEL	EMPNO	ENAME	JOB	SAL	DEPTNO
-----	-----	-----	-----	-----	-----
UNCLASSIFIED	7369	SMITH	CLERK	800	20
SENSITIVE	7499	ALLEN	SALESMAN	1600	30
SENSITIVE	7521	WARD	SALESMAN	1250	30
TRULY_SENSITIVE	7566	JONES	MANAGER	2975	20
SENSITIVE	7654	MARTIN	SALESMAN	1250	30
TRULY_SENSITIVE	7698	BLAKE	MANAGER	2850	30
TRULY_SENSITIVE	7782	CLARK	MANAGER	2450	10
SENSITIVE	7788	SCOTT	ANALYST	88	20
TRULY_SENSITIVE:ALPHA	7839	KING	PRESIDENT	5000	10
SENSITIVE	7844	TURNER	SALESMAN	1500	30
UNCLASSIFIED	7876	ADAMS	CLERK	1100	20
UNCLASSIFIED	7900	JAMES	CLERK	950	30
SENSITIVE	7902	FORD	ANALYST	3000	20
UNCLASSIFIED	7934	MILLER	CLERK	1300	10

三、视图机制

- 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

视图举例

- 例：王平只能检索计算机系学生的信息
- 先建立计算机系学生的视图**CS_Student**

```
CREATE VIEW CS_Student  
AS  
SELECT  
FROM Student  
WHERE dept='CS';
```

视图举例

- 在视图上进一步定义存取权限

```
GRANT SELECT  
ON CS_Student  
TO 王平 ;
```

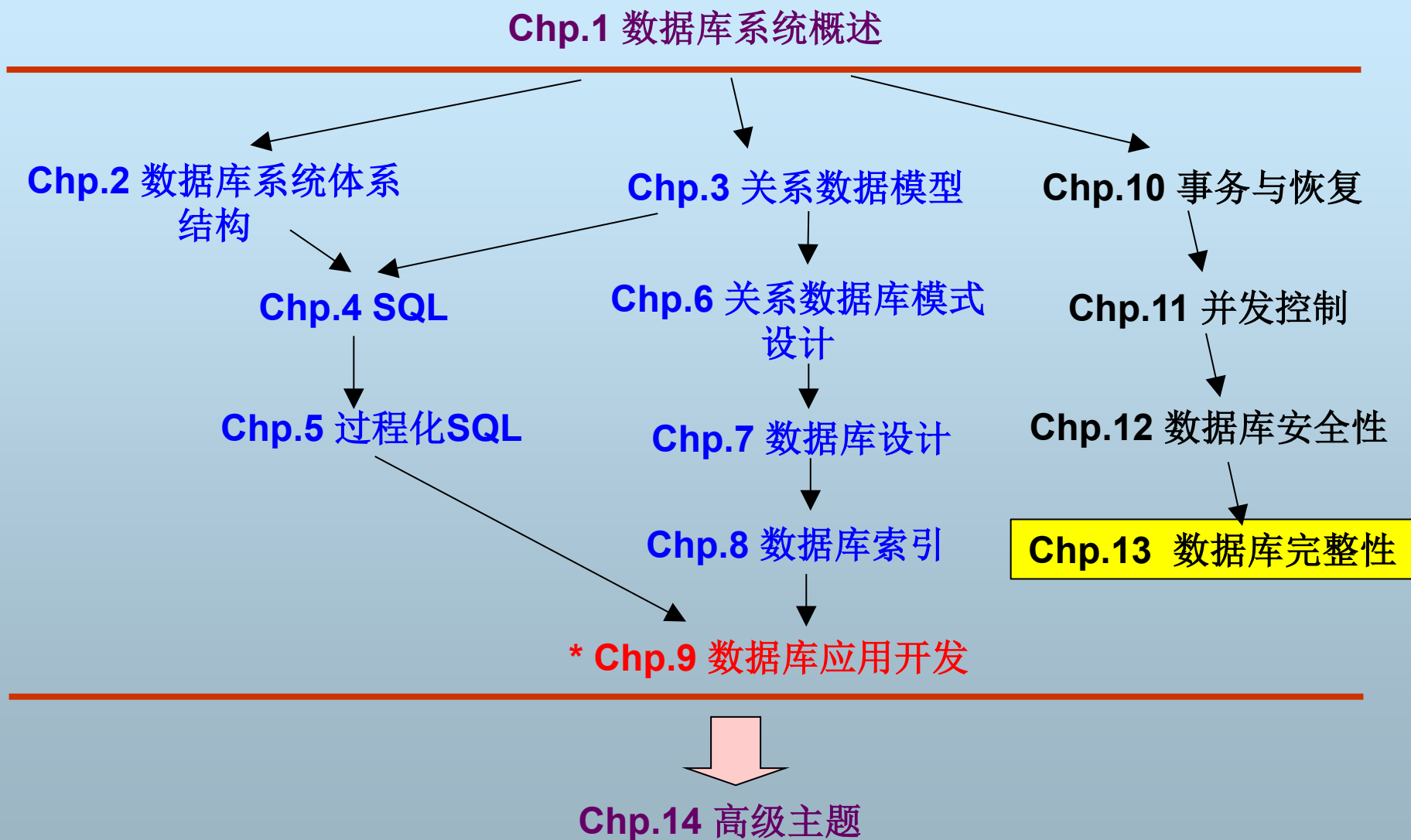
- 同时禁止在基本表的直接存取权限

本章小结

- 数据库安全性控制概述
- 自主存取控制方法
- 强制存取控制方法
- 视图机制

第13章 数据库完整性

课程知识结构



Databases Protection

- 数据库保护：排除和防止各种对数据库的干扰破坏，确保数据安全可靠，以及在数据库遭到破坏后尽快地恢复
- 数据库保护通过四个方面来实现
 - 完整性控制技术
 - ◆ Enable constraints
 - 安全性控制技术
 - ◆ Authorization and authentication
 - 数据库的恢复技术
 - ◆ Deal with failure
 - 并发控制技术
 - ◆ Deal with data sharing

主要内容

- 数据库完整性概念
- 完整性约束类型
- 完整性实施途径

一、数据库完整性概念

- 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

1、完整性控制功能

■ 完整性控制机制应具有的三个功能

- 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）
- 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。

2、完整性规则定义

- **DBA向DBMS提出的一组完整性规则，来检查数据库中的数据是否满足语义约束，主要包括三部分：**
 - 触发条件：系统什么时候使用规则来检查数据
 - 约束条件：系统检查用户发出的错误操作违背了什么完整性约束条件
 - 违约响应：违约时要做的事情
- 完整性约束规则是由**DBMS**提供的语句来描述，存储在数据字典，但违约响应由系统来处理

2、完整性规则定义

■ 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

	学号不能为空	教授工资不得低于 1000 元
D	约束的对象为 SNO 属性	约束的对象为 Sal 属性
O	插入和修改 STUDENT 元组时	插入和修改职工元组时
A	SNO 不能为空	Sal 不能小于 1000
C	无（作用于所有记录的 SNO 属性）	职称='教授'（ 作用于职称='教授'的记录）
P	拒绝执行	拒绝执行

二、完整性约束分类

- 按约束的粒度
 - 表级约束、列级约束、元组级约束
- 按约束对象的状态
 - 静态约束、动态约束
- 按约束作用类型分
 - 域完整性、实体完整性、参照完整性

1、按约束粒度分类

■ 按约束的粒度分

- **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；

1、按约束粒度分类

[例 1] 建立部门表DEPT，要求部门名称Dname列为字符串型且不可为空，部门编号Deptno列为主码

```
CREATE TABLE DEPT
(
  Deptno  NUMBER,
  Dname   CHAR(9) NOT NULL,
  Location CHAR(10),
  PRIMARY KEY (Deptno)
);
```


列级约束

表级约束

1、按约束粒度分类

[例 2] 当学生的性别是男时，其名字不能以Ms. 打头。

```
CREATE TABLE Student
(Sno      CHAR(9),
Sname     CHAR(8) NOT NULL,
sex       CHAR(2),
age       SMALLINT,
PRIMARY KEY (Sno),
CHECK (sex='女' OR Sname NOT LIKE 'Ms. %')
);
```



- 定义了元组中Sname和 sex两个属性值之间的约束条件
- 性别是‘女’的元组都能通过该项检查，因为sex=‘女’成立；当性别是男性时，要检查Sname不能以Ms. 开头

2、按约束对象的状态分类

■ 按约束对象的状态分

- **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；
 - ◆ 例如：学生关系中年龄不能大于**100**
- **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
 - ◆ 例如调整工资时须满足：

现有工资 > 原有工资+工龄*100

2、按约束作用类型分类

■ 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

域完整性

- 域是一组具有相同类型的值的集合
- SQL3支持域的概念，并可以用**CREATE DOMAIN***语句建立一个域以及该域应满足的完整性约束条件

[例 3] 建立一个性别域GenderDomain，并对其中的限制命名

```
CREATE DOMAIN GenderDomain CHAR(2)
    CONSTRAINT chkGD CHECK(VALUE IN ('男', '女'));
```

```
CREATE TABLE student
( Sno CHAR(10),
  sex GenderDomain,
  Sname CHAR(20)
)
```

* Oracle和MySQL不支持Create Domain语句，但Oracle可用Create Type语句实现类似功能（MySQL不支持）

三、完整性实施途径

- 约束 (Constraint)
- 触发器 (Trigger)
- 规则 (Rule)
- 断言 (Assertion)

1、约束

- 约束的用途是限制输入到表中的值的范围。约束是实施数据完整性的首选方法。
- SQL中的约束
 - 主键约束 (**Primary Key**)
 - 唯一键约束 (**Unique**)
 - 外键约束 (**Foreign Key**)
 - 检查约束 (**Check**)

2、触发器

- 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。

3、规则

- 规则是一组用过程化SQL(如T-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

- T-SQL是Microsoft SQL Server的过程化SQL语言
- Oracle和MySQL不支持Create Rule

3、规则

- 例：在数据库中创建一个Email的规则对象，其值为包含@的字符串。

- 创建规则(T-SQL)

```
CREATE RULE rl_email  
AS  
@val LIKE '%@%'
```

- 绑定规则(绑定到学生表的Email字段) **sp_bindrule**是系统存储过程

```
sp_bindrule 'rl_email', 'student.Email'
```

4、断言

- **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
- 例：限制每一门课程选修人数不超过**60**人。

● T-SQL

```
CREATE ASSERTION asser1 CHECK(60>=ALL (SELECT count(*) FROM  
SC GROUP BY c#));
```

* *Oracle和MySQL不支持Create Assertion语句*

本章小结

- 数据库完整性概念
- 完整性约束类型
- 完整性实施途径