数据库 01

2025 年 3 月 18 日

考纲内容：

- **绪论**
  数据库的 4 个基本概念，数据管理技术的产生和发展，数据建模、概念模型和数据模型的三要素，数据库系统的三级模式结构，数据库的两级映像与数据独立性，数据库系统的组成。
- **关系模型**
  关系模型的数据结构及形式化定义，关系操作，关系完整性，关系代数（传统的集合运算、专门的关系运算）。
- **关系数据库标准语言 SQL**
  数据定义、数据查询、数据更新、空值处理、视图。
- **数据库安全性**
  数据库安全性概述，数据库安全性控制。

1 绪论

1.1 数据库的基本概念

数据库系统的四个基本概念：

1. 数据 (Data)：描述事物的符号记录，是数据库中存储的基本对象。
2. 数据库 (Database, DB)：长期存储在计算机内、有组织的、可共享的大量数据的集合。数据库中的数据具有如下特点：
   - 永久存储
   - 有组织
   - 可共享
3. 数据库管理系统 (Database Management System, DBMS)：位于用户与操作系统之间的一层数据管理软件，是基础软件，是一个大型复杂的软件系统。
4. 数据库系统 (Database System, DBS)：由数据库、数据库管理系统（及其应用开发工具）、应用程序和数据库管理员（DBA）组成的存储、管理、处理和维护数据的系统。

1.2 数据管理技术的产生和发展

数据管理技术的发展经历了三个阶段：

1. **人工管理阶段**（20 世纪 50 年代中期以前）

- 数据不保存
- 应用程序管理数据
- 缺点：数据不可共享、数据冗余度大、数据不一致性

2. **文件系统阶段**（20 世纪 50 年代中期至 60 年代中期）
   - 数据可长期保存
   - 文件系统实现对数据的管理
   - 缺点：数据共享性差、数据冗余大、数据独立性差
3. **数据库系统阶段**（20 世纪 60 年代末至今）
   - 数据结构化
   - 数据共享性高，冗余度低
   - 数据独立性高
   - 由 DBMS 统一管理和控制

1.3 数据建模与数据模型

1.3.1 数据建模

数据建模是抽象、表示和处理现实世界中数据的方法和过程。

1.3.2 概念模型

概念模型是按用户的观点来对数据和信息建模，主要用于数据库设计。
最常用的概念模型是实体-联系模型 (E-R 模型)，它由下列要素组成：

- 实体 (Entity)：客观存在并可相互区别的事物
- 属性 (Attribute)：实体所具有的某一特性
- 联系 (Relationship)：实体之间的关联

图 1: E-R 图示例：学生-课程关系

1.3.3 数据模型的三要素

数据模型是对现实世界数据特征的抽象，由三部分组成：

1. **数据结构**：描述数据库的组成对象及对象间的联系
2. **数据操作**：对数据库中各种对象实例允许执行的操作及操作规则
3. **数据约束**：保证数据库中数据满足特定语义规则的条件

按照抽象级别，数据模型可分为：

- **概念模型**：面向用户，如 E-R 模型
- **逻辑模型**：面向 DBMS，如层次模型、网状模型、关系模型、面向对象模型等
- **物理模型**：面向存储，描述数据在存储介质上的实际组织方式

1.4 数据库系统的三级模式结构

ANSI/SPARC 提出的三级模式结构包括：

1. 外模式 (External Schema)：也称为用户模式，是用户与数据库系统的接口，由若干外部视图组成。
2. 模式 (Schema)：也称为概念模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。
3. 内模式 (Internal Schema)：也称为存储模式，是数据物理结构和存储方式的描述，是数据在存储介质上的表示方式和存取方法。

注：此处三种模式，其实对应用户，软件开发者和数据库开发人员。

1.5 数据库的两级映像与数据独立性

1.5.1 两级映像

1. 外模式/模式映像：定义外模式与模式之间的对应关系，当模式改变时，对应的外模式/模式映像也需要改变。
2. 模式/内模式映像：定义模式与内模式之间的对应关系，当内模式改变时，对应的模式/内模式映像也需要改变。

1.5.2 数据独立性

1. 物理数据独立性：当数据库的内模式改变时，只需要修改模式/内模式映像，使模式保持不变，应用程序不受影响。
2. 逻辑数据独立性：当数据库的模式改变时，只需要修改外模式/模式映像，使外模式保持不变，应用程序不受影响。

数据独立性是数据库系统的重要特征，它保证了应用程序和数据库结构的相对独立，从而提高了数据库系统的可维护性和扩展性。

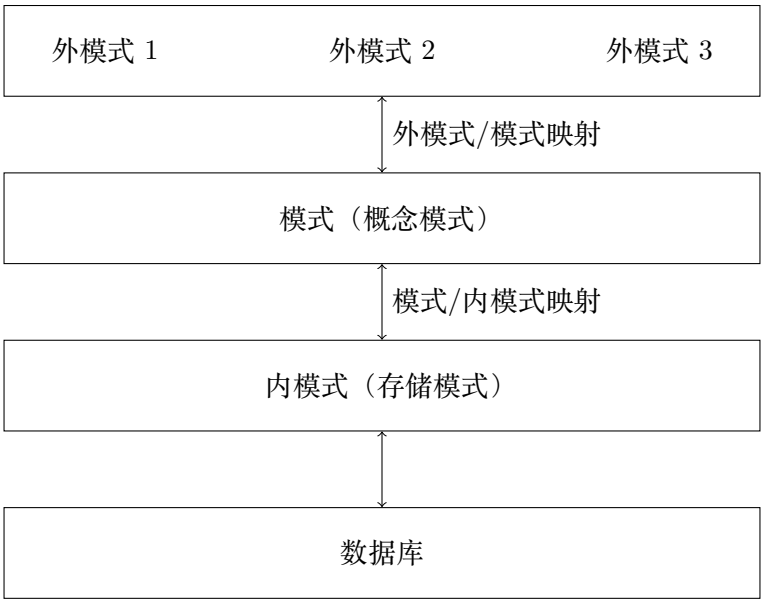


图 2: 数据库系统的三级模式结构

1.6 数据库系统的组成

数据库系统由以下几部分组成：

1. **硬件平台**：计算机、存储设备和网络设备等
2. **数据库**：存储在计算机中的数据集合
3. **DBMS**：管理数据库的软件
4. **应用程序**：为用户提供操作界面的程序
5. **数据库管理员 (DBA)**：负责数据库的规划、设计、维护和管理
6. **用户**：使用数据库的人，包括最终用户、应用程序员和 DBA

1.7 小结

- 数据库系统的四个基本概念：数据、数据库、数据库管理系统和数据库系统
- 数据管理技术的发展经历了人工管理、文件系统和数据库系统三个阶段
- 数据模型由数据结构、数据操作和数据约束三要素组成
- 数据库系统采用三级模式结构：外模式、模式和内模式
- 数据库的两级映像支持物理数据独立性和逻辑数据独立性
- 数据库系统由硬件、软件、数据、人员等组成部分构成

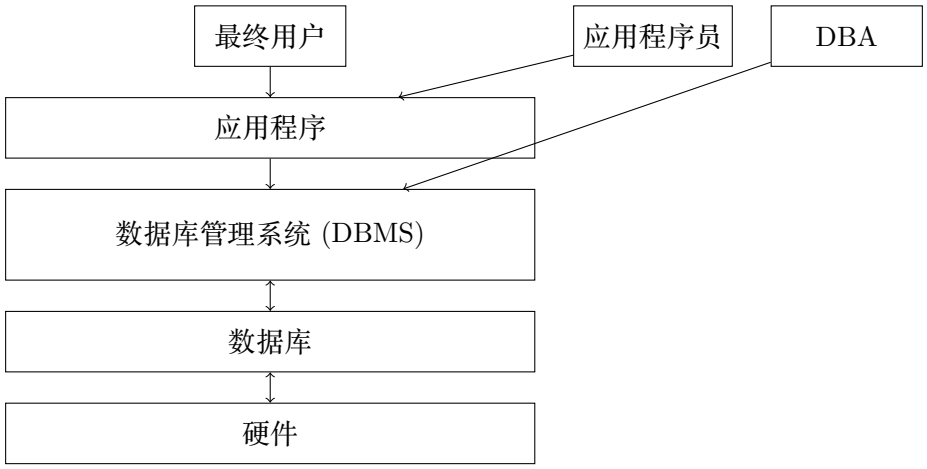


图 3: 数据库系统的组成

2 关系模型

2.1 关系模型的数据结构

关系模型是目前最重要的数据库模型，由 E.F.Codd 于 1970 年首先提出。关系模型的基本数据结构非常简单，就是关系，即二维表格结构。

2.1.1 关系的形式化定义

设有 n 个域 $D_1, D_2, \ldots, D_n$，它们的笛卡尔积为：

$$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) | d_i \in D_i, i = 1, 2, \ldots, n\} \tag{1}$$

关系 (Relation)是笛卡尔积 $D_1 \times D_2 \times \ldots \times D_n$ 的子集，表示为 $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$，其中：

- R 是关系名
- A_i 是属性名
- D_i 是域（属性的取值范围）
- n 是关系的目或度（Degree），表示关系的属性个数
- 关系中的每个元组（Tuple）对应表中的一行
- 关系模式（Relation Schema）：$R(A_1, A_2, \ldots, A_n)$

2.1.2 关系的性质

1. 列是同质的：每一列中的数据来自同一个域，是同一类型的数据

2. 不同的列可来自同一个域：不同属性可对应相同的域
3. 列的顺序无关紧要：列的次序可以任意交换
4. 行的顺序无关紧要：行的次序可以任意交换
5. 行列确定唯一的值：给定行号和列名后，表中的值唯一确定
6. 不允许表中有重复的行（元组）：任意两个元组在至少一个属性上取值不同
7. 每个分量必须是不可分的数据项：不允许表中的表（非规范化）

Example 1. 某大学学生关系 *Student* 的一个实例：

Sno	***Sname***	***Ssex***	***Sage***
201901	李勇	男	*20*
201902	刘晨	女	*19*
201903	王敏	女	*18*

关系模式：*Student(Sno, Sname, Ssex, Sage)*

2.1.3 关系模型中的基本概念

1. 候选键（Candidate Key）：能唯一标识关系中元组的最小属性集合。
2. 主键（Primary Key）：从候选键中选定的一个，用于唯一标识关系中的元组。
3. 外键（Foreign Key）：关系 R 的一个属性（或属性集），它不是 R 的主键，但是它在另一个关系 S 中是主键。
4. 主属性（Prime Attribute）：包含在任何一个候选键中的属性。
5. 非主属性（Non-prime Attribute）：不包含在任何候选键中的属性。

2.2 关系操作

关系模型的操作主要分为查询和更新两类。

2.2.1 查询操作

查询操作是关系数据库中最基本的操作，主要包括：

- 选择（Selection）：从关系中选取满足条件的元组
- 投影（Projection）：从关系中选取指定的列
- 连接（Join）：将两个关系按照共同属性组合成一个关系
- 除法（Division）：A ÷ B，求 A 中满足 B 中所有条件的元组
- 并（Union）：两个关系的并集

- 差（Difference）：两个关系的差集
- 交（Intersection）：两个关系的交集
- 笛卡尔积（Cartesian Product）：两个关系的所有可能组合

2.2.2 更新操作

更新操作包括：

- 插入（Insert）：向关系中添加元组
- 删除（Delete）：从关系中删除元组
- 修改（Update）：修改关系中的元组

2.3 关系完整性

关系模型中的完整性约束是保证数据库中数据正确性、有效性和相容性的规则，主要包括：

2.3.1 实体完整性（Entity Integrity）

实体完整性规则：关系的主键属性值不能为空（NULL）。
这保证了每个实体（即关系中的每一行）都能被唯一标识。

2.3.2 参照完整性（Referential Integrity）

参照完整性规则：如果关系 R 的外键 F 是关系 S 的主键，则关系 R 中每个元组在 F 上的取值必须是：

- 要么等于关系 S 中某个元组的主键值
- 要么为空值（如果允许外键取空值）

2.3.3 用户定义的完整性（User-defined Integrity）

用户定义的完整性是针对具体应用的约束条件，例如：

- 属性值的范围约束（例如年龄必须大于 0 且小于 120）
- 属性间的相互约束（例如入职日期必须晚于出生日期）

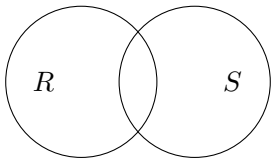
2.4 关系代数

关系代数是一种抽象的查询语言，它用对关系的运算来表达查询。

2.4.1 传统的集合运算

1. 并（Union）：$R \cup S = \{t | t \in R \vee t \in S\}$
2. 差（Difference）：$R - S = \{t | t \in R \wedge t \notin S\}$
3. 交（Intersection）：$R \cap S = \{t | t \in R \wedge t \in S\}$
4. 笛卡尔积（Cartesian Product）：$R \times S = \{(r, s) | r \in R \wedge s \in S\}$

需要注意的是，进行并、差、交运算的两个关系必须是同元（union-compatible）的，即它们必须具有相同的目（属性数）且对应的属性来自相同的域。



$R \cup S$: 两个圆的全部区域
$R \cap S$: 两个圆的交叉区域
$R - S$: 仅在 R 中的区域

图 4: 集合运算示意图

2.4.2 专门的关系运算

1. 选择（Selection）：$\sigma_F(R) = \{t | t \in R \wedge F(t) = \text{true}\}$

   选择操作是从关系 R 中选取满足给定条件 F 的元组。

2. 投影（Projection）：$\Pi_A(R) = \{t[A] | t \in R\}$

   投影操作是从关系 R 中选取指定的属性 A 组成新的关系。

3. 连接（Join）：

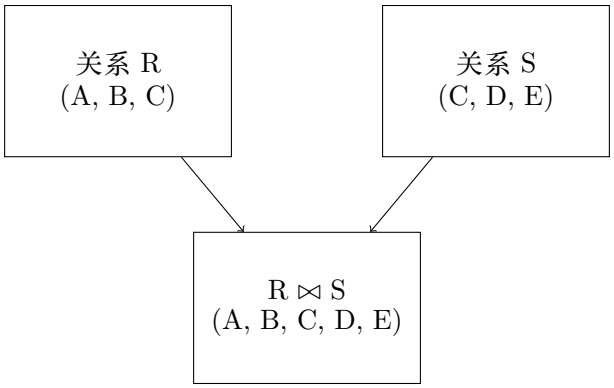
   - 自然连接（Natural Join）：$R \bowtie S = \{rs | r \in R \wedge s \in S \wedge r[A] = s[A]\}$，其中 A 是 R 和 S 共有的属性集合。
   - θ-连接（Theta Join）：$R \bowtie_\theta S = \{rs | r \in R \wedge s \in S \wedge \theta(r, s)\}$
   - 外连接（Outer Join）：保留在连接中无匹配的元组，缺少的属性值用 NULL 填充。

4. 除法（Division）：$R \div S = \{t[X] | t \in R \wedge \forall s \in S, ts \in R\}$

   除法操作用于回答” 对于 S 中的所有值，找出 R 中与之相关的所有值” 类型的查询。

Example 2. 考虑以下两个关系：

学生关系 *Student(Sno, Sname, Sage, Sdept)*
选课关系 *SC(Sno, Cno, Grade)*



自然连接：基于共同属性 C

图 5: 自然连接示例

以下关系代数表达式表示”查询所有选修了*'C1'* 课程的学生姓名”:
$\Pi_{Sname}(\sigma_{Cno='C1'}(Student \bowtie SC))$
执行过程：

1. 首先，*Student* 和 *SC* 进行自然连接（按 *Sno* 属性）
2. 然后，选择 *Cno='C1'* 的元组
3. 最后，投影出 *Sname* 属性

2.5 扩展的关系代数操作

除了基本的关系代数操作外，还有一些扩展操作：

1. 广义投影（Generalized Projection）：允许在投影列表中包含计算表达式
2. 聚集（Aggregation）：包括 COUNT、SUM、AVG、MAX、MIN 等聚合函数
3. 外连接（Outer Join）：保留在连接中无匹配的元组
4. 半连接（Semi-join）：$R \ltimes S = \Pi_R(R \bowtie S)$

2.6 小结

- 关系模型以简单的二维表格形式表示数据，具有形式化的数学基础
- 关系的基本特性包括列的同质性、行列无序性、元组的唯一性等
- 关系完整性约束包括实体完整性、参照完整性和用户定义的完整性
- 关系代数提供了一套形式化的操作，分为传统的集合运算和专门的关系运算
- 关系代数是关系数据库查询语言的理论基础，SQL 语言实现了关系代数的大部分功能

3 关系数据库标准语言 SQL

3.1 简介

SQL (Structured Query Language) 是关系数据库的标准语言，由 IBM 研究所开发，最初称为 SEQUEL (Structured English Query Language)。SQL 已成为关系数据库领域的国际标准语言，几乎所有主流数据库管理系统都支持 SQL。

SQL 语言具有以下特点：

- 综合统一：集数据定义、数据查询、数据操纵和数据控制功能于一体
- 高度非过程化：用户只需指定”做什么”，而非”怎么做”
- 面向集合操作：操作对象和结果都是集合
- 以同一种语法结构提供多种使用方式
- 语言简洁，易学易用

3.2 数据定义 (DDL)

数据定义语言 (Data Definition Language) 用于定义数据库结构，包括创建、修改和删除数据库对象。

3.2.1 创建表

Listing 1: 创建表的基本语法

```
CREATE TABLE 表名 (
    列名1 数据类型1 [列级约束条件],
    列名2 数据类型2 [列级约束条件],
    ...
    [,表级约束条件]
);
```

示例：创建学生表

```
CREATE TABLE Student (
    Sno CHAR(9) PRIMARY KEY,
    Sname VARCHAR(20) NOT NULL,
    Ssex CHAR(2),
    Sage SMALLINT,
    Sdept VARCHAR(20)
);
```

3.2.2 修改表

```
-- 添加列
ALTER TABLE 表名 ADD 列名 数据类型 [约束];

-- 修改列定义
ALTER TABLE 表名 MODIFY 列名 数据类型 [约束];

-- 删除列
ALTER TABLE 表名 DROP COLUMN 列名;
```

3.2.3 删除表

```
DROP TABLE 表名;
```

3.3 数据查询 (DQL)

数据查询语言 (Data Query Language) 是 SQL 的核心功能，用于从数据库中检索数据。

3.3.1 基本查询结构

```
SELECT [DISTINCT] <目标列表达式> [别名]
FROM <表名或视图名> [别名]
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]];
```

3.3.2 单表查询

示例：查询所有学生的学号和姓名

```
SELECT Sno, Sname
FROM Student;
```

3.3.3 多表连接查询

示例：查询学生及其选修课程成绩

```
SELECT Student.Sno, Sname, Cname, Grade
FROM Student, Course, SC
WHERE Student.Sno = SC.Sno
  AND SC.Cno = Course.Cno;
```

3.3.4 嵌套查询

```sql
SELECT Sname
FROM Student
WHERE Sno IN (
    SELECT Sno
    FROM SC
    WHERE Cno = 'C001'
);
```

3.3.5 集合查询

```sql
-- 并操作
SELECT Sno FROM Student WHERE Sdept = 'CS'
UNION
SELECT Sno FROM SC WHERE Grade > 90;

-- 交操作
SELECT Sno FROM Student WHERE Sdept = 'CS'
INTERSECT
SELECT Sno FROM SC WHERE Grade > 90;

-- 差操作
SELECT Sno FROM Student WHERE Sdept = 'CS'
EXCEPT
SELECT Sno FROM SC WHERE Grade > 90;
```

3.4 数据更新 (DML)

数据操纵语言 (Data Manipulation Language) 用于添加、修改和删除数据库中的数据。

3.4.1 数据插入

```sql
-- 插入单条数据
INSERT INTO 表名 [(列名1, 列名2, ...)]
VALUES (值1, 值2, ...);

-- 插入查询结果
INSERT INTO 表名 [(列名1, 列名2, ...)]
SELECT 列1, 列2, ... FROM 表名 WHERE 条件;
```

3.4.2 数据修改

```
UPDATE 表名
SET 列名1 = 表达式1, 列名2 = 表达式2, ...
[WHERE 条件];
```

3.4.3 数据删除

```
DELETE FROM 表名
[WHERE 条件];
```

3.5 空值处理

空值 (NULL) 表示”不知道”或”不存在”或”无意义”的值，需要特殊处理。

3.5.1 空值判断

```
-- 判断是否为空
SELECT Sname
FROM Student
WHERE Sage IS NULL;

-- 判断是否非空
SELECT Sname
FROM Student
WHERE Sage IS NOT NULL;
```

3.5.2 空值函数

许多数据库系统提供处理空值的特殊函数：

- `COALESCE(expr1, expr2, ...)`：返回第一个非空表达式的值
- `NULLIF(expr1, expr2)`：如果 expr1=expr2 则返回 NULL，否则返回 expr1
- `ISNULL(expr, replacement)`：如果 expr 为 NULL 则返回 replacement，否则返回 expr

```
-- 示例：如果年龄为NULL则显示为'未知'
SELECT Sname, COALESCE(CAST(Sage AS VARCHAR), '未知') AS Age
FROM Student;
```

3.6 视图

视图 (View) 是从一个或几个基本表（或视图）导出的表，是一种虚拟表。

3.6.1 创建视图

```
CREATE VIEW 视图名 [(列名1, 列名2, ...)]
AS 查询语句
[WITH CHECK OPTION];
```

示例：创建计算机系学生的视图

```
CREATE VIEW CS_Student
AS SELECT Sno, Sname, Sage
FROM Student
WHERE Sdept = 'CS';
```

3.6.2 查询视图

```
SELECT * FROM CS_Student;
```

3.6.3 修改视图

```
ALTER VIEW 视图名
AS 新查询语句;
```

3.6.4 删除视图

```
DROP VIEW 视图名;
```

3.6.5 视图更新的限制

并非所有视图都是可更新的。一般而言，包含以下内容的视图不可更新：

- 包含 GROUP BY 子句或聚合函数的视图
- 包含 DISTINCT 关键字的视图
- 包含操作符如 UNION, INTERSECT, EXCEPT 的视图
- 包含子查询中引用 FROM 子句的同一个表的视图

4 数据库安全性

4.1 数据库安全性概述

4.1.1 安全性的概念与重要性

数据库安全性 (Database Security)是指保护数据库以防止未授权的访问、恶意攻击或意外泄露，同时确保数据的完整性、可用性和保密性。在信息时代，数据库安全已成为组织信息系统安全的核心部分。

数据库安全性涉及多个层面：

- **物理安全**：保护数据库服务器和存储介质
- **系统安全**：操作系统和网络层面的安全措施
- **DBMS 安全机制**：数据库管理系统提供的安全功能
- **数据安全**：保护存储在数据库中的具体数据

4.1.2 安全威胁类型

数据库面临的主要安全威胁包括：

1. 机密性威胁：未授权用户获取敏感数据
2. 完整性威胁：数据被非法修改
3. 可用性威胁：合法用户无法访问所需数据
4. 责任认定威胁：无法追踪用户操作

安全漏洞可能来自内部威胁（如员工）或外部威胁（如黑客），且内部威胁通常更为严重。

4.1.3 数据库安全需求

一个安全的数据库系统应满足以下需求：

- **身份认证**：确认用户身份
- **访问控制**：限制用户只能访问授权资源
- **审计跟踪**：记录用户活动以供事后检查
- **加密保护**：防止数据被窃取时泄露信息
- **完整性控制**：确保数据不被非法修改
- **可用性保障**：确保系统正常运行并防止拒绝服务攻击

4.2 数据库安全性控制

4.2.1 身份认证

身份认证 (Authentication)是数据库安全的第一道防线，用于验证用户身份。
常见的认证机制包括：

- 基于密码的认证（最常见）
- 多因素认证（结合密码、生物识别、令牌等）

- 基于证书的认证
- 集成的认证（如 LDAP、Active Directory）

在 SQL 中实现用户认证：

Listing 2: 创建用户并设置密码

```
-- MySQL中创建用户
CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';

-- Oracle中创建用户
CREATE USER username IDENTIFIED BY password;

-- SQL Server中创建用户
CREATE LOGIN username WITH PASSWORD = 'password';
CREATE USER username FOR LOGIN username;
```

4.2.2 访问控制机制

访问控制 (Access Control)用于确保用户只能执行被授权的操作。DBMS 通常实现以下访问控制模型：

自主访问控制 (DAC) 自主访问控制允许数据对象的所有者决定谁可以访问对象以及允许的操作。在关系数据库中主要通过授权机制实现。

Listing 3: SQL 中的权限授予与回收

```
-- 授予权限
GRANT privilege_list
ON object
TO user_list
[WITH GRANT OPTION];

-- 回收权限
REVOKE privilege_list
ON object
FROM user_list
[CASCADE | RESTRICT];
```

示例：授予查询权限

```
-- 授予用户查询学生表权限
GRANT SELECT ON Student TO user1;

-- 授予包含授权权的权限
GRANT SELECT, UPDATE(Sage, Sdept) ON Student
TO user2 WITH GRANT OPTION;
```

```
7
8 -- 回收权限
9 REVOKE SELECT ON Student FROM user1;
```

强制访问控制 (MAC) 强制访问控制基于安全标签，不由用户决定，而是由系统强制执行。每个数据对象被分配一个安全级别，每个用户被授予一个安全许可，只有当用户的安全许可满足对象的安全级别要求时才允许访问。

基于角色的访问控制 (RBAC) RBAC 通过定义角色并将权限分配给角色，再将用户分配到角色，从而简化权限管理。

Listing 4: 使用角色进行权限管理

```
1 -- 创建角色
2 CREATE ROLE student_admin;
3
4 -- 给角色授权
5 GRANT SELECT, INSERT, UPDATE ON Student TO student_admin;
6
7 -- 将用户分配到角色
8 GRANT student_admin TO user1, user2;
```

4.2.3 视图机制与安全

视图是实现数据库安全性的重要方法，可以：

- 隐藏表中的某些列
- 隐藏表中的某些行
- 将多个表的信息组合在一起，但仅显示授权部分

Listing 5: 使用视图限制数据访问

```
1 -- 创建只包含非敏感信息的视图
2 CREATE VIEW PublicStudentInfo AS
3 SELECT Sno, Sname, Sdept
4 FROM Student;
5
6 -- 将视图权限授予用户
7 GRANT SELECT ON PublicStudentInfo TO public_users;
```

4.2.4 审计

审计 (Auditing)是监控并记录用户对数据库的访问和操作，帮助检测安全违规并提供责任追踪。

Listing 6: 启用数据库审计

```
-- Oracle启用审计
AUDIT SELECT, UPDATE, DELETE ON Student;

-- SQL Server设置审计
CREATE SERVER AUDIT StudentDataAudit
TO FILE (FILEPATH = 'C:\auditlog');

CREATE DATABASE AUDIT SPECIFICATION StudentTableAudit
FOR SERVER AUDIT StudentDataAudit
ADD (SELECT, UPDATE, DELETE ON Student BY public);
```

审计信息通常包含:

- 操作用户
- 操作类型
- 操作时间
- 操作对象
- 操作结果
- 可能的前后值

4.2.5 数据加密

加密 (Encryption)是将数据转换为只有授权方能理解的形式，保护数据即使在被窃取的情况下也不被理解。

数据库加密有几个层次:

1. **连接加密**：客户端与服务器之间的通信加密（如 SSL/TLS）
2. **透明数据加密 (TDE)**：整个数据库文件加密
3. **列级加密**：只加密敏感列
4. **应用层加密**：在应用程序中对数据进行加密/解密

Listing 7: SQL Server 中的列加密示例

```
-- 创建主密钥
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'StrongPassword123!';

-- 创建证书
CREATE CERTIFICATE StudentCert WITH SUBJECT = 'Student Data Protection';

-- 创建对称密钥
```

```sql
CREATE SYMMETRIC KEY StudentKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE StudentCert;

-- 加密数据
OPEN SYMMETRIC KEY StudentKey DECRYPTION BY CERTIFICATE StudentCert;

UPDATE Student
SET SSN = EncryptByKey(Key_GUID('StudentKey'), SSN);

CLOSE SYMMETRIC KEY StudentKey;
```

4.2.6 推断控制

推断控制 (Inference Control)用于防止用户通过合法查询推断出敏感信息。推断威胁常见于统计数据库和数据仓库。

常见的推断控制方法包括：

- 限制查询集大小（防止针对特定个体的查询）
- 添加随机噪声（在结果中添加细微的随机变化）
- 数据扰动（修改原始数据，但保持统计特性）
- 限制查询重叠（防止通过多个查询交叉获取信息）

4.2.7 数据库安全的综合防御策略

有效的数据库安全需要多层次防御策略：

1. **定期安全评估**：识别和修补安全漏洞
2. **最小权限原则**：只授予必要的权限
3. **安全更新和补丁**：保持 DBMS 为最新版本
4. **数据备份与恢复**：建立定期备份和恢复程序
5. **安全配置**：禁用不必要的功能和默认账户
6. **安全培训**：培训管理员和用户安全意识

数据库安全是一个持续过程，需要定期评估和更新安全措施以应对新的威胁和漏洞。