Contents

[第一章：数据库基本概念 2](#_Toc212397006)

[数据库的发展阶段 2](#_Toc212397007)

[数据库DB 2](#_Toc212397008)

[数据库管理系统DBMS 2](#_Toc212397009)

[数据库系统DBS 2](#_Toc212397010)

[三级结构和二级映像（数据独立性） 5](#_Toc212397011)

[第二章：关系数据库基本理论 8](#_Toc212397012)

[关系模型 8](#_Toc212397013)

[关系系统 9](#_Toc212397014)

[关系代数 10](#_Toc212397015)

[第三章：数据库完整性 13](#_Toc212397016)

[RDBMS 完整性控制 13](#_Toc212397017)

[用户定义完整性 14](#_Toc212397018)

[第四章：SQL 15](#_Toc212397019)

[SQL 15](#_Toc212397020)

[语法 16](#_Toc212397021)

[第五章：数据库对象 20](#_Toc212397022)

[视图 20](#_Toc212397023)

[存储过程 21](#_Toc212397024)

[索引 22](#_Toc212397025)

[触发器 23](#_Toc212397026)

[第六章：数据库安全 25](#_Toc212397027)

[完整性 VS 安全性 25](#_Toc212397028)

[保证安全性的方法（从外到内） 25](#_Toc212397029)

[授权与角色 26](#_Toc212397030)

[第七章：（事务）并发控制 28](#_Toc212397031)

[事务 28](#_Toc212397032)

[并发 29](#_Toc212397033)

[第八章：备份和恢复 32](#_Toc212397034)

[回滚和重做 32](#_Toc212397035)

[故障类型及处理方法 32](#_Toc212397036)

[数据库恢复策略 32](#_Toc212397037)

[第九章：数据库应用系统设计 35](#_Toc212397038)

[函数依赖关系 35](#_Toc212397039)

[**范式** 35](#_Toc212397040)

[数据库设计 37](#_Toc212397041)

[错题 40](#_Toc212397042)

## 第一章：数据库基本概念

### 数据库的发展阶段

* 人工管理→文件系统阶段→数据库阶段

### 数据库DB

* 定义：永久存储、有组织、可共享的数据集合（属于**硬件**）
* 数据库特点
  + 结构化
  + 高数据对立性
  + 低冗余，高共享
  + 数据安全，正确，一致，可靠

### 数据库管理系统DBMS

* 定义：用户和操作系统之间的数据管理软件，总是基于某种数据模型。**是一个独立的系统**。
* 用于并发性控制的机制是：**封锁机制，时间戳机制**。

### 数据库系统DBS

* **DBS = DB + 硬件 + 软件（DBMS、操作系统） + 用户（人和应用程序）**；**需要操作系统的支持**
* 功能：安全性，完整性，并发控制。
* 数据库系统设计依赖的**数据模型**：
  + 转换关系：现实世界→信息世界（概念模型 ERD）→机器世界（逻辑+物理模型）
  + **概念模型 ERD**（信息世界的建模）
    - 独立于计算机，与具体DBMS无关
    - 常用：**E-R模型**（实体-联系模型）
      * **三要素：实体，属性，联系**
    - 联系的类型
      * 一对一
      * 一对多
      * 多对多
  + **逻辑模型**（计算机角度，DBMS结合）
    - 三要素（只有逻辑模型包括数据部分，因此**等同于数据模型的组成部分**）
      * **数据结构**
      * **数据操作**
      * **数据完整性约束**
    - **“层次模型”** 
      * 以“**树型结构**”组织数据。子节点只能有一个父节点。
      * IBM和IMS使用
    - **“网状模型” DBTG** 
      * **一个节点可以有多个父节点和多个子节点**
      * DBTG
    - **“关系模型”（最广泛）**
      * **对用户来说是一组二维表**，有严格的数学基础。
      * **模型**是**静态**的，稳定的；而**关系**是**动态**的。
      * Oracle
    - **非关系型 NoSQL**
      * 不保证 ACID 特性
      * 类型
        + 键值（Key-Value Model）

Redis， Oracle BDB

* + - * + 列存储

Cassandra， HBase

* + - * + 文档

MongoDB

* + - * + 图

Neo4j，InfoGrid

* + - **“面向对象模型”**
  + **物理模型**（最底层）
    - 面向存储结构和访问路径的实现方式
    - 涉及索引、哈希、B+树等数据存取方法

### 三级结构和二级映像（数据独立性）

**A diagram of a system

AI-generated content may be incorrect.**

* 当数据库的逻辑结构或物理存储结构发生变化时，**只需要修改二级映像**。
* **先概念设计（概念级），再逻辑设计（用户级/外模式），最后物理设计（物理级/内模式）**。
* **三级结构**
  + 好处：用户可以逻辑的，**抽象**的处理数据，不用关心具体的表示方式和物理存储方式。
  + **用户级（依赖 DBMS 提供的数据访问接口）**
    - 一个应用只能有一个**外模式/子模式/用户模式**；一个外模式可以有多个应用
      * **局部**逻辑结构，属于**逻辑设计**阶段任务。
      * **不是唯一**的
    - **“视图”** 是外模式的实现
  + **概念级（独立于DBMS）**
    - **模式/概念模式/逻辑模式** 
      * **全局**逻辑结构，属于**概念设计**阶段任务。
      * **唯一**的
    - **“表”** 是模式的实现
  + **物理级（依赖DBMS）**
    - **内模式/存储模式** 
      * **唯一**的
    - “存储结构”（文件、索引、B+树、哈希等）是内模式的实现
* **二级映像**
  + 定义：二级映像是三层模式之间的“映射关系”，用来把一层的数据结构对应到另一层。
  + 目的：实现应用程序和**数据的独立性**，即 应用程序无需随数据逻辑或物理结构的改变而修改。
  + **外模式/模式 映像关系** 
    - **保证逻辑独立性**
    - 逻辑独立性：模式的修改 只需要对 “外模式/模式”映像关系修改
  + **模式/内模式 映像关系** 
    - **保证物理独立性**
    - 物理独立性：内模式的修改 只需要对 “模式/内模式”映像关系修改

|  |  |  |  |
| --- | --- | --- | --- |
| 层级 | 名称 | 特点 | 实现示例 |
| 用户级 | 外模式 / 子模式 / 用户模式 | 局部逻辑结构；可有多个 | 视图（View） |
| 概念级 | 模式 / 概念模式 / 逻辑模式 | 全局逻辑结构；唯一 | 表（Table） |
| 物理级 | 内模式 / 存储模式 | 存储结构；唯一 | 文件、索引、B+树、哈希 |

|  |  |  |
| --- | --- | --- |
| 映像类型 | 作用 | 保证的数据独立性 |
| 外模式 ↔ 模式 | 当逻辑结构变化时，只修改映像即可 | 逻辑独立性 |
| 模式 ↔ 内模式 | 当物理存储方式变化时，只修改映像即可 | 物理独立性 |

## 第二章：关系数据库基本理论

### 关系模型

|  |  |  |  |
| --- | --- | --- | --- |
| 概念 | 含义 | 特点 | 类比（表） |
| 关系模型 | 一种数据库逻辑模型，用二维表表示 | 理论 | 整个数据库体系 |
| 关系模式 | 关系的结构定义 | 静态 | **表头（结构定义）** |
| 关系 | 关系模式的具体实例，一张二维表 | 动态 | **表中的数据内容** |

* 定义：一种数据库逻辑模型，使用二维表来表示实体及实体之间的联系。
  + **关系模型必须满足：实体完整性 + 参照完整性**（参考第三章）
* **性质**：
  + 列（属性）不可再分 —— 即为原子性（Atomicity）
  + 任意两个元组（行）不能完全相同 —— 唯一性
  + 每一行表示一个实体或联系，每一列表示一个属性（特征）
* **术语** 
  + 元组：表中的一行
  + **度（目）**：元数，即关系中属性的个数
  + **域**：某一列允许取值的“集合”或“类型范围”。
  + **分量**：元组中每个属性的值，也就是行（记录）中列的具体值。
    - 元组 (1001, “张三”, 20) 中的 1001、张三、20 都是分量
  + **基数**：关系中元组的数量，即行数
  + **关系模式**：二维表的结构，特指表头
  + 主码：primary key，唯一，**非空**
  + **候选码**：candidate key，最小，唯一，**非空**
    - 属于候选码的都叫做**主属性**，其它的叫做非主属性
      * 比如一个CK包含两个属性，那么只有合起来才叫做一个主属性，单独的话不能叫做主属性。
  + 超键：唯一，**非空**
* **操作** 
  + 定义：基于关系代数运算实现；操作对象是集合，操作结果也是集合
  + 查询，插入，删除，修改

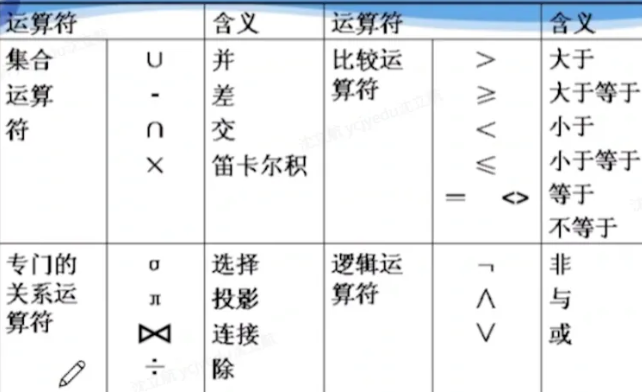
### 关系系统

* 定义：一个数据库系统在多大程度上符合**关系模型的要求**。
* 关系数据库系统的最低要求
  + **支持关系代数的三种基本操作：**
    - 选择（Selection）
    - 投影（Projection）
    - 连接（Join）
  + **用户不需要定义物理存取路径**
    - 即系统自动管理索引、文件存储与访问路径。
    - 用户只需逻辑地操作表（逻辑独立性）。
  + **系统中唯一的数据结构是“表”**
    - 不应混用文件、指针或层次结构。
* 根据其对“关系特性”的支持程度不同，可以分为以下几类：
  + 表式系统：只是“表格化存储” → 结构化但无操作
  + 最小关系系统：支持关系代数操作 → 可进行查询和组合
  + 关系完备系统：操作功能完备 → 满足关系代数的“关系完备性”
  + 全关系系统 RDBMS：在此基础上增加完整性约束 → 真正的关系数据库系统（如 Oracle、MySQL），依据**Codd 的关系模型** 构建的。
    - **Codd 的关系模型**提出了数据库应该以“关系”（二维表）为核心，数据间的逻辑关系由表中的行（元组）和列（属性）表示。

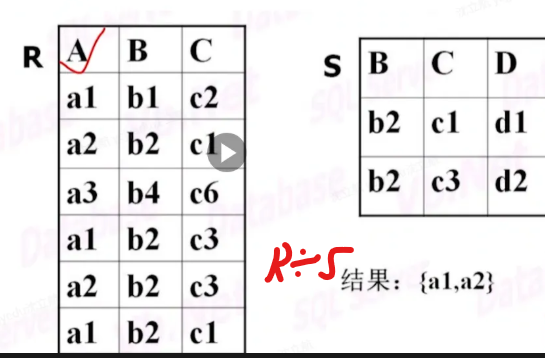
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | 数据结构 | 数据操作 | 完整性约束 | 特点 |
| 表式系统 | 表 | 无 | 无 | 仅仅把数据放到表里，但不具备完整的关系特性 |
| 最小关系系统 | 表 | 有关系操作（选择、投影、连接等） | 无 | 满足基本关系代数操作，但完整性可能缺失。 |
| 关系完备的系统 | 表 | 有关系操作（选择、投影、连接等） | 无 | 相对完善，但完整性机制可能有限。 |
| 全关系系统 | 表 | 有关系操作（选择、投影、连接等） | 有 | 真正意义上的关系数据库系统RDBMS。 |

### 关系代数

* **5种最基本的运算：投影，选择，笛卡尔积，并，差**



* **集合运算：并，差，交，广义笛卡尔积**
  + **并，差，交只能做同质运算** 
    - 属性个数（目）相同；
    - 对应属性的域（数据类型）相同。
  + **笛卡尔积**：对两个关系 和 ，笛卡尔积 是所有可能的**元组组合**集合。每个结果元组由 R 的一个元组和 S 的一个元组合并而成。
    - **广义笛卡尔积**的列数等于各个关系的列数之和，行数等于各个关系的行数乘积
  + 交：返回两个关系中共有的元组。
    - 或者
  + 差：返回在第一个关系中但不在第二个关系中的元组。
* **关系运算：投影 Projection Π，选择 Selection σ，连接，除**
  + 选择和投影是“单目运算”；连接和除是“双目运算”
    - **单目**：从一个关系中选出满足条件的元组
  + **投影会移除重复的行**
  + 连接
    - **等值连接**
      * 使用指定的属性列进行连接，并且连接后保留两个相同属性列。
    - **自然连接**
      * 一定是等值连接；它会**用 所有同名属性 作为连接条件**，而不是只选一个。
      * **自动去除重复属性列**；
      * **自然连接 = 选择+投影+笛卡尔积**
  + **除**：同时对列和行进行过滤，过滤后，只保留出现在R中的列，比如下面例子中只保留A列
    - **结果会自动去除重复值**。



## 第三章：数据库完整性

|  |  |  |
| --- | --- | --- |
| 类型 | 定义 | 示例 |
| 实体完整性约束（Entity Integrity） | 每个实体必须有唯一标识; PRIMARY KEY | 主键（Primary Key） |
| 参照完整性约束（Referential Integrity） | 外键的值必须能在主表的主键中找到，或者为空; FOREIGN KEY | 外键（Foreign Key） |
| 用户定义完整性约束（User-defined Integrity） | 根据业务需求自定义的约束; CHECK、NOT NULL、UNIQUE、DEFAULT | 性别只能是“男”或“女”；年龄>0 |
| 关键字完整性约束（Key Integrity Constraint） | 候选键或主键必须唯一，且非空。UNIQUE 或 PRIMARY KEY |  |

* **完整性 = 正确性 + 相容性**
  + 用于保证数据库中数据的准确性、一致性和可靠性。
* 关系模型必须满足：实体完整性 + 参照完整性。
* CONSTRAINT <约束名> PRIMARY KEY (...)
* CONSTRAINT <约束名> FOREIGN KEY (...) REFERENCES ...
* CONSTRAINT <约束名> UNIQUE (...)
* CONSTRAINT <约束名> CHECK (...)

### RDBMS 完整性控制

* 控制方式
  + 定义机制：通过 DDL 语句（如 CREATE TABLE）定义约束
  + 完整性检查：在 INSERT、UPDATE、DELETE 操作后进行约束验证
  + 违约处理：处理违反完整性约束的情况
* 3种实现方式
  + **拒绝执行（默认）**：直接阻止非法操作
  + **级连操作（CASCADE）**：自动修改或删除所有相关不一致元组
  + **设置为空值或默认值**：自动将外键或列值设置为 NULL 或默认值

### 用户定义完整性

* 常见用户定义约束关键字：
  + NOT NULL：列不能为空
  + UNIQUE：列值唯一
  + CHECK：满足特定条件
  + DEFAULT：默认值
* 用户违背该完整性，则RDBMS应该 拒绝执行

## 第四章：SQL

### SQL

* **SQL = 关系代数语言 + 关系演算语言** 
  + 关系代数：用对关系的运算来表达查询要求
  + 关系演算：用谓语来表达查询要求
* 特点
  + 关系型数据库语言
  + 综合统一：集合 DDL、DML、DCL
  + 高度非过程化：只需说明“做什么”，无需说明“怎么做”
  + 面向集合操作
  + 多种使用方式：既可独立使用，也可嵌入程序语言
  + 简单易学
* 子查询（Subquery）
  + **不相关子查询**
    - 子查询独立执行，不依赖外层查询
    - 执行顺序：子查询 → 得到结果 → 外层查询使用结果

SELECT name, salary

FROM Employee

WHERE salary > (SELECT AVG(salary) FROM Employee);

* + **相关子查询**
    - 子查询依赖外层查询的某一行
    - 外层取一行 → 子查询执行一次 → 判断是否满足条件

SELECT name, salary, dept\_id

FROM Employee e1

WHERE salary > (

SELECT AVG(salary)

FROM Employee e2

WHERE e1.dept\_id = e2.dept\_id

);

### 语法

|  |  |  |
| --- | --- | --- |
| 类型 | 包含语句 | 说明 |
| DDL（Data Definition Language） | CREATE, DROP, ALTER | 定义或删除数据库对象（表、视图、索引、模式） |
| DML（Data Manipulation Language） | SELECT, INSERT, UPDATE, DELETE | 数据操作 |
| DCL（Data Control Language） | GRANT, REVOKE | 权限控制 |

* **语法执行顺序：FROM→WHERE→GROUP BY→HAVING→SELECT→ORDER BY**
* CREATE
  + SCHEMA
  + TABLE
  + VIEW
  + INDEX：数据库为表中的一个或多个列建立的 **快速查询数据结构**
    - **CREATE INDEX** idx\_student\_name **ON** Student **(**Name**);**
* ALTER 只允许修改：TABLE, INDEX
  + **添加列**: ALTER TABLE Flight ADD test1 INTEGER, ADD test2 INTEGER; 不能同时添加constraint
  + **删除列**: ALTER TABLE Flight DROP test1;
  + **添加Constraint:** 只能先删除旧的外键约束，再添加一个新的外键约束
    - ALTER TABLE Orders ADD FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id);
    - ALTER TABLE Orders ADD UNIQUE (column\_name);
  + **删除Constraint**: 除了NOT NULL和DEFAULT以外，别的都需要对应的Constraint名字来删除
    - ALTER TABLE table\_name DROP CONSTRAINT constraint\_name;
    - ALTER TABLE table\_name ALTER COLUMN column\_name DROP NOT NULL;
  + **重命名列**: ALTER TABLE table\_name RENAME COLUMN old\_column\_name TO new\_column\_name;
  + **重命名表**: ALTER TABLE old\_table\_name RENAME TO new\_table\_name;
  + **修改Default Value 或者 data type** 
    - ALTER TABLE table\_name ALTER COLUMN column\_name SET DEFAULT 7.77;
    - ALTER TABLE table\_name ALTER COLUMN column\_name TYPE new\_data\_type;
* DROP
  + **RESTRICT**：若对象被引用，则删除失败。
    - DROP TABLE Department RESTRICT;
  + **CASCADE**：删除对象的同时，删除引用它的对象
    - DROP TABLE Department CASCADE;
* **TRUNCATE 和 DELETE都可以删除表中所有数据，但保留表结构；只有DROP可以用于删除表结构**。
  + DELETE FROM 表名; -- 不加 WHERE 删除所有行
  + DELETE FROM 表名 WHERE 删除指定行
  + TRUNCATE TABLE 表名;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 操作 | 类型 | 删除对象 | 是否保留表结构 | 备注 |
| DELETE | DML | 表中的数据（可加 WHERE 条件） | 是 | 可以有条件删除，可回滚（事务可控） |
| TRUNCATE | DDL | 表中所有数据 | 是 | 删除所有数据，速度快，不可条件删除，通常不可回滚 |
| DROP | DDL | 表及表结构 | 否 | 完全删除表，包括结构和数据 |

* 笛卡尔乘积: 把表 A 的每一行和表 B 的每一行组合，结果行数 = |A| × |B|

SELECT \*

FROM A, B;

或者

SELECT \*

FROM A CROSS JOIN B;

* FULL JOIN: 保留左右两个表的所有行，如果某一边没有满足ON匹配，就用NULL填充。

SELECT \*

FROM A

FULL JOIN B

ON A.id = B.id;

## 第五章：数据库对象

|  |  |  |
| --- | --- | --- |
| 对象 | 对应模式 | 说明 |
| 存储过程（Stored Procedure） | **外模式** / 用户模式（External） | 封装用户操作逻辑，供应用程序或用户调用 |
| 视图（View） | **外模式** / 用户模式（External） | 用户看到的逻辑表示，可以隐藏部分数据，实现局部逻辑结构 |
| 表（Table） | **概念模式** / 模式（Conceptual） | 数据库核心结构，定义列和数据类型，是逻辑数据模型的实现 |
| 索引（Index） | **内模式** / 存储模式（Internal） | 用于加速数据访问，属于物理实现细节 |
| 触发器（Trigger） | **内模式** / 存储模式（Internal） | 数据操作自动触发，属于数据库内部机制 |

### 视图

CREATE VIEW view\_name AS

SELECT ...

FROM ...

WHERE ...

WITH CHECK OPTION;

* 定义
  + 视图是一个虚拟表，是对基本表的查询结果命名后的**逻辑表示**。
  + 数据库中只存放视图定义（定义语句），**不存放数据**。
  + 所有对视图的操作最终都会转换为对原始表的操作。
* 特点
  + 在有WITH CHECK OPTION后，插入、更新、删除时会检查是否仍符合视图定义条件，不符合则拒绝执行。
  + 不允许出现 ORDER BY / DISTINCT，因为视图定义只描述集合，不描述排序或唯一性。
  + 使用DROP VIEW view\_name; 删除
* 作用
  + 主要作用：提供用户视角的数据
  + **简化用户操作**（屏蔽复杂的连接与计算）。
  + **逻辑独立性**：数据库重构时，不影响依赖视图的应用。
  + **安全性**：只暴露用户需要的数据，保护机密信息。
  + **多角度访问**：不同用户可通过不同视图看同一数据。
  + **增强可读性**：复杂查询能更清晰表达。
* 类型

|  |  |  |
| --- | --- | --- |
| 类型 | 说明 | 可更新性 |
| 行列子集视图 | 从基本表中选取部分行和列构成，保留主键 | ✅ 可更新 |
| 汇总（聚合）视图 | 使用 SUM、AVG、COUNT、GROUP BY 等 | ❌ 只读 |
| 连接视图 | 基于多个表的连接结果 | ❌ 一般只读 |
| 基于视图的视图 | 由其他视图生成的视图 | 通常只读 |

### 存储过程

* 定义：存储过程是一组为完成特定功能而编写的 SQL 语句集合，**存储在数据库中**，可以由应用程序或用户通过调用来执行。
* 作用
  + 主要是：提高数据操作效率
  + 减少网络数据流量
  + 增强代码的重用性和共享性
  + 加快系统运行速度
    - **首次执行会被编译，之后会被缓存**
  + 提高数据的安全性：只给用户使用procedure的权限，不授予对于表的权限
* 用**CREATE PROCEDURE**或**CREATE PROC**创建存储过程；用**EXCUTE**调用存储过程
* 可以**返回结果集**和**单一结果**

### 索引

* 定义：索引是数据库中用于**加速数据检索**的数据结构，相当于书本的“目录”。它不改变数据逻辑结构，但可以极大提高查询性能。
* 特点
  + 可以对一个或多个目创建索引来查询性能。
  + 由 **DBA（数据库管理员）** 或 **建表人** 根据需要创建；
  + 由 **DBMS 自动维护**；由 **DBMS 自动决定** 是否使用索引（用户无需干预）；
  + DROP INDEX index\_name; 删除索引
  + 索引加速查询，但会降低插入、删除、更新的性能（因为要维护索引结构）。适用于频繁查询、少量更新的表。
* 索引类型
  + 聚簇索引
    - 与**索引顺序一致**，数据表中的数据行按照索引键（key）的排序顺序在磁盘上物理连续地存储，**聚簇索引本身就是表的物理存储**。**仅能有一个（通常是主键）**。
    - **叶节点直接存放数据行**
    - **查询单行快**
      * 查一次索引页即可，因为叶子节点直接有数据。
    - **查询多行快**
      * **顺序 I/O**，连续读取，速度快。
  + 非聚簇索引
    - 与**索引顺序无关，非聚簇索引是对数据表的一层“目录”，不是数据本身。**可有**多个**。
    - **叶节点存放指针**（指向数据行）
    - **查询单行时慢**
      * 查一次索引页 + 查一次数据页 → 两次 I/O
    - **查询多行时慢**
      * 每找到一条记录，都要随机跳转一次磁盘页。

|  |  |  |
| --- | --- | --- |
| 类型 | 特点 | 示例 |
| 普通索引（INDEX） | 加速查询，无唯一性要求 | CREATE INDEX idx\_name ON table(col); |
| 唯一索引（UNIQUE INDEX） | 索引列值必须唯一（NULL可重复）。三种语法创建，主键，唯一键，和CREATE UNIQUE INDEX | CREATE UNIQUE INDEX idx\_unique ON table(col); |
| 主键索引（PRIMARY KEY） | 自动创建唯一且非空的索引 | 由 PRIMARY KEY 约束自动生成 |
| 聚簇索引（CLUSTER INDEX） | 数据行**按索引顺序**实际存储（物理有序），**查询速度快。插入，删除开销大**。 | 一个表**只能有一个（通常是主键）** |
| 非聚簇索引（NON-CLUSTERED INDEX） | 仅存储指向数据行的指针（逻辑有序），**查询速度慢。插入，删除开销小**。 | 一个表可有多个 |

### 触发器

* 定义：一个**用户自定义**的 SQL **事务命令集合**。
  + 在大部分数据库中，触发器只能针对“表”定义。（单选就选表，多选把视图也选上）
* **作用**：
  + 实现复杂的约束，尤其是 **主键/外键无法保证的完整性**
  + 执行业务规则，如日志记录、数据自动更新等
* **特点**：
  + 总是**绑定到表**（或视图）
  + **自动触发**，不能被直接调用
  + 触发器本身是一个**事务**
* CREATE TRIGGER用来创建；DROP TRIGGER用来删除；用ALTER TRIGGER来更改
* INSERTED 表和 DELETED 表
  + **INSERTED 表**：存储新增或更新后的数据行的数据行
  + **DELETED 表**：存储被删除或更新前的数据行
  + **触发器表**：指触发器绑定的表（真实表）
  + 它们都是 虚拟表，只在触发器内部存在，触发器结束后自动删除

|  |  |  |  |
| --- | --- | --- | --- |
| 操作类型 | INSERTED 表内容 | DELETED 表内容 | 触发器表（原表）变化 |
| INSERT | 新插入的行 | 空 | 新行被加入表中 |
| UPDATE | 更新后的行 | 更新前的行 | 原表中对应行被修改为新值 |
| DELETE | 空 | 被删除的行 | 原表中对应行被删除 |

## 第六章：数据库安全

### 完整性 VS 安全性

* 相同：都是对数据库进行保护
* 不同

|  |  |  |
| --- | --- | --- |
| 对比 | 完整性 | 安全性 |
| 保护对象 | 数据合法性 | 数据不被非法访问或破坏 |
| 防护重点 | 防止合法用户输入不合规数据 | 防止非法用户窃取、更改或破坏数据 |

### 保证安全性的方法（从外到内）

* **用户标识和鉴别**
  + 通过口令（密码）确认用户身份
* **存取控制**
  + 具体包含
    - **定义用户权限**：分配操作数据库对象的权限，如 SELECT、INSERT、UPDATE、DELETE
    - **合法权限检查**：确保用户操作在权限范围内
  + 存取控制类型：
    - **强制存取控制（MAC, Mandatory Access Control）**
      * 系统为每位用户和数据对象分配安全级别
      * 用户不能随意修改权限，严格执行访问策略
    - **自主存取控制（DAC, Discretionary Access Control）**
      * 数据/资源所有者可以自主决定谁可以访问自己的数据
* **视图**
  + 通过限制用户可见的数据列或行，实现数据访问控制
* **审计**
  + 建立审计日志，记录用户操作，便于追踪和分析异常行为
* **数据加密**
  + 对存储或传输的数据进行加密，防止非法访问

### 授权与角色

* **授权（Grant）**：定义用户对数据库对象的操作权限
* **角色（Role）**：权限的集合
  + 将一组具有相同权限的用户组织在一起，形成角色
  + 一个用户可以拥有多个角色
  + 简化权限管理

|  |  |  |
| --- | --- | --- |
| 操作 | 语法 | 说明 |
| 创建角色 | CREATE ROLE role\_name; | 创建一个新角色 |
| 授予权限 | **GRANT** privilege ON object **TO** user/role; | 由 DBA、OWNER 或拥有权限的用户授予 |
| 收回权限 | **REVOKE** privilege ON object **FROM** user/role; | 撤销指定权限 |

* **GRANT — 授权**
  + 用于将权限授予用户或角色。

可授权 **整个表/视图** 或 **具体列**（列级授权需明确列名）。比如下面的SELECT就是授权读取整个表，UPDATE就是授权更改Sno列。

GRANT SELECT, UPDATE(Sno)

ON TABLE Student

TO U4;

* + 可以最后加上 **WITH GRANT OPTION**：允许被授权用户再授予权限给其他用户。
* **REVOKE — 收回权限**
  + 用于收回用户或角色的权限。**可以收回数据库级权限，表级权限，列级权限**。

REVOKE SELECT

ON TABLE SC

FROM PUBLIC;

* + 可选 **CASCADE**：同时收回被该用户授权给其他用户的权限。
  + 可选 **RESTRICT**：如果用户有下属授权，则阻止收回。
  + 使用 **FROM PUBLIC** 可以一次收回或授予所有用户的权限。

## 第七章：（事务）并发控制

### 事务

* 定义：用户定义的数据库操作序列，要么全部执行成功，要么全部不执行。是保证数据库 一致性 的核心机制。
  + 不可分割：保证原子性（Atomicity）
  + 可以是一条 SQL，也可以是多条 SQL
  + 一个程序可以包含多个事务
* 事务处理技术主要包括**数据库恢复技术**和**并发控制技术**。
* **BEGIN** **TRANSACTION** 开始；**COMMIT**结束；**ROLLBACK**处理事务异常
* 特性 ACID
  + **原子性 Atomicity**：事务中的操作要么全做，要么全不做
  + **一致性 Consistency**：事务执行前后，数据库保持完整性约束
  + **隔离性 Isolation** 
    - 锁机制
    - 事务隔离级别
    - 并发控制算法
  + **持续性 Durability**：事务一旦提交，修改永久保存，即使系统故障也不丢失
* 破坏 ACID 的因素
  + 事务并行/并发执行
  + 事务运行时被强制停止

### 并发

* **并发方式**
  + **交叉**：单处理机，通过时间片轮转实现并发
  + **同时**：多处理机，多事务真正同时执行
* 操作带来的数据不一致
  + **丢失修改/更新**：写+写
    - 两个事务同时修改同一数据，后提交的修改覆盖了先提交的修改，导致先前的更新丢失。
  + **不可重复读**：读+写
    - 同一事务中，两次读取同一数据却得到不同的结果。
  + **脏读**：写+读
    - 一个事务读到了另一个事务尚未提交的数据。
* **并发控制技术**
  + 时间戳
  + 乐观控制法
  + 多版本并发控制（MVCC）
  + 封锁
    - 锁类型

|  |  |  |
| --- | --- | --- |
| 锁 | 别名 | 作用 |
| 排他锁 X | 写锁 | 一个事务加 X 锁后，其他事务不能加 S 或 X 锁（不可读不可写） |
| 共享锁 S | 读锁 | 一个事务加 S 锁后，其他事务可加 S 锁（可读），不能加 X 锁（不可写） |

* + - 锁协议

|  |  |  |  |
| --- | --- | --- | --- |
| 协议 | 读操作 | 写操作 | 能防止的问题 |
| 一级 | 不加锁（裸读） | 必须加X锁直到事务结束 | 防止 丢失修改 |
| 二级 | 读时加 S 锁，**读完释放** | 必须加X锁直到事务结束 | 防止 丢失修改、脏读 |
| 三级 | 读时加 S 锁**直到事务结束** | 必须加X锁直到事务结束 | 防止 丢失修改、脏读、不可重复读 |

* + - **活锁**：事务（或进程）虽然在不断改变自己的状态以避免冲突，但始终无法完成操作，也无法前进。
      * 比如优先级高的事务持续占用资源，低优先级的事务始终无法执行完成。
      * 如何避免？**先来先服务**
    - **死锁**：两个或多个事务（或进程） 相互等待对方持有的资源，导致都无法继续执行。
      * 如何避免？
        + **一次封锁法**：事务在开始操作前，一次性将所有需要的数据加锁。
        + **顺序封锁法**：给所有资源定义统一的顺序编号。事务必须按编号顺序申请锁。
      * 诊断
        + **超时法**：诊断
        + **等待图法**：节点标识事务，边表示T1→T2
      * 解除
        + 选择一个处理代价最小的事务，将其撤销
* **并发调度的可串行化**
  + **可串行化调度**：一个调度如果执行结果与某种事务 **串行执行** 的结果相同，则称该调度 **可串行化**。
    - 保证并发执行的事务结果正确，不产生冲突。
  + **两段锁协议 2PL**
    - 定义：一个事务执行过程中，先 **申请锁**（上锁阶段），后 **释放锁**（解锁阶段），且两个阶段不交错。
    - 作用：
      * 事务遵守两段锁是可串行化的**充分**条件。
      * **遵守2PL仍然可能产生死锁**。
    - 阶段划分：
      * **增长阶段**：事务只申请锁，不释放锁
      * **收缩阶段**：事务只释放锁，不再申请锁

## 第八章：备份和恢复

### 回滚和重做

* + **回滚（Rollback）**：恢复未提交事务
  + **重做（Redo）**：恢复已提交事务

### 故障类型及处理方法

|  |  |  |
| --- | --- | --- |
| 故障类型 | 原因 | 处理方式 |
| 事务故障 | 用户操作错误、应用程序错误，运算溢出、死锁 | 回滚 未完成事务 |
| 系统故障 | CPU故障、断电、内存丢失 | 撤销未完成事务 + 重做已提交事务 |
| 介质故障 | 磁盘损坏、磁头碰撞 | 备份 + 日志恢复 |
| 外部原因 | 自然灾害、病毒入侵 | 备份 + 日志恢复 |

### 数据库恢复策略

* **建立冗余数据**
  + **数据转储（Backup）**
    - **静态转储**：系统无运行事务时进行，保证一致副本
    - **动态转储**：记录转储期间事务修改，结合日志恢复到某一时刻（日志+副本=把数据库恢复到某一时刻）
    - **海量转储**：每次备份整个数据库
    - **增量转储**：每次只备份上次转储后修改的数据
  + **登记日志文件（Log）**
    - **用途**：**事务故障恢复 & 系统故障恢复**
    - **内容**：
      * 事务标识（开始、更新、提交）
      * 操作对象
      * 操作类型（插入、删除、修改）
      * 更新前后的值
    - **原则**：
      * 严格遵循并发事务执行顺序。
      * **先写日志文件，再写数据库**。
* 数据库恢复策略
  + 标准顺序：**先备份 → 再日志 → 先重做（REDO） → 后回滚（ROLLBACK）**
    - 读取的第一个备份是数据库的全备份
  + **事务故障恢复**
    - **方式**：**反向扫描日志**，撤销未完成事务
    - **影响**：**只影响当前事务**，不影响其他事务，数据库整体保持一致
    - 例子：**运算溢出，除0**。
  + **系统故障恢复**
    - **方式**：系统重启，**正向扫描日志**
    - **影响**：
      * **内存中数据丢失**。
      * 磁盘已提交数据不丢失
      * 未完成事务回滚，已提交事务重做
  + **介质故障恢复**
    - **方式**：使用备份 + 日志，**需DBA介入**
    - **影响**：
      * **磁盘数据库部分或全部丢失/损坏**。
      * 依赖 **备份 + 日志** 恢复完整数据
  + **病毒入侵**
    - 不仅可能破坏数据库本身的数据，还可能影响**日志、备份**以及正在运行的事务，因此恢复通常需要 **备份 + 日志**，并可能涉及**全量检查和数据校验**。

### 备份种类

|  |  |  |
| --- | --- | --- |
| 类型 | 说明 | 适用场景 |
| 全备份 | 备份数据库中的**全部数据和结构** | 周期性完整保存数据库 |
| 差异备份 | 备份自上次**全备份以来**发生变化的数据 | 介于全备份和增量备份之间 |
| 增量备份 | 只备份自**上次任何类型备份**以来发生变化的数据 | 节省空间和时间，但恢复复杂 |
| 日志备份 | 备份事务日志文件内容，用于**时间点恢复** | 配合全备份或差异备份使用 |
| 镜像备份 | 实时复制数据库到另一存储设备 | 高可用与灾难恢复 |
| 冷备份 | 数据库关闭状态下复制数据库文件 | 简单、安全，但不可用期间长 |
| 热备份 | 数据库运行状态下进行备份 | 常用于高可用系统 |

## 第九章：数据库应用系统设计

### 函数依赖关系

* 函数依赖：若属性组 **Y** 函数依赖于属性组 **X**，记作 **X → Y**，则称 **X** 为 **决定因子（Determinant）**。
  + **平凡函数依赖**：当 **Y ⊆ X** 时，称 **X → Y** 为平凡依赖。  
    例：{A, B} → {A}。
  + **非平凡函数依赖**：当 **Y ⊄ X** 时，称 **X → Y** 为非平凡依赖。  
    例：{A, B} → {C}。
* **完全函数依赖**
  + **X → Y** 成立，
  + 且对于 **X** 的任何真子集 **X′**，依赖 **X′ → Y** 均不成立，
* **传递函数依赖**
  + **X → Y**
  + **Y → Z**
  + 且 **Y ⊄ X** 且 **Z ⊄ Y**

### **范式**

**1NF(第一范式)**

* 要求:属性值必须是原子性(不可再分)。
* 与函数依赖关系不大,主要约束数据结构。

**2NF(第二范式)**

* 基于:完全函数依赖
* 要求:消除 部分函数依赖，即非主属性必须完全依赖于主键，不能非主属性只函数依赖于主键中的一部分。
* 解决问题:减少因部分依赖导致的数据冗余。

**3NF(第三范式)**

* 基于:传递函数依赖
* 要求:消除 传递函数依赖，即非主属性不能传递依赖于主键。但是箭头可以从非主属性指向Candidate Key的一部分。
* 解决问题:避免更新异常。

**BCNF(Boyce-Codd 范式)**

* 基于:候选键与函数依赖的更严格约束
* 要求:关系中的每一个函数依赖X→Y,X必须是候选键。即所有箭头都从超键出发。
* 解决问题:进一步消除冗余,避免异常。

**4NF**

* 消除非平凡的多值依赖。
* 若存在非平凡多值依赖 X ↠ Y（X 多值决定 Y），则 X 必须是关系 R 的超键（superkey）。

**关系规范化/规范化过程**

* 定义：指通过一系列范式（如第一范式1NF、第二范式2NF、第三范式3NF）来消除**数据冗余和避免逻辑结构异常**。
* 原则：**保持无损连接 和 原本的依赖关系**。
* **插入异常**：应该插入的数据无法插入。
* **删除异常**：不该删除的数据被删除了。
* **更新异常**：更新某条数据时，如果同一信息在表的多个地方重复存储，需要在所有位置同时修改，否则会导致数据不一致。

### 数据库设计

* 定义：**优化数据库逻辑模式与物理模式**的过程。
* 主要内容
  + 信息管理要求：需要存储、管理的数据类型及其约束。
  + 数据操作要求：需要支持的操作类型（查询、插入、删除、修改等）。
* **新奥尔良法** — 六步法
  + 1. **需求分析**
    - 目标：了解并分析用户的实际需求，明确系统功能和数据需求。
    - 主要任务：确定用户接口、输入输出、操作流程。
    - 常用工具：
      * IPO（输入-处理-输出）图
      * UML（统一建模语言）
      * 数据流程图（DFD）
      * 数据字典（Data Dictionary）
      * 功能结构图（Function Structure Chart）
  + 2. **概念结构设计**
    - 目标：用抽象的概念模型表示数据与其相互关系。
    - 常用工具：ER 图（E-R Diagram）
    - 核心概念：
      * 关系模式（Schema）：表的结构定义（表头）。
      * 关系（Relation）：表中的实际数据（表体）。

|  |  |  |
| --- | --- | --- |
| 联系类型 | 是否需将“联系”单独转换为关系（表） | 说明 |
| 1 : N（或部分 1 : 1） | 否 | 通常在 N 方放外键 |
| M : N（多对多） | 必须 | 需创建中间表 |
| 特殊 1 : 1 | 可合并 | 当参与度或语义允许时，可合并为一个实体 |

* + - 数据抽象方法：
      * **分类**：把现实世界中具有相同性质或特征的对象归为一类，形成一个实体类型
      * **聚类**：把多个相关的实体类型或联系类型结合起来，形成一个更高层次的抽象实体。是一种**横向的“合并”**。
      * **概括**：从多个实体类型中，抽取出它们的共同属性和共同联系，形成一个上层（父类）实体类型。是一种**纵向的“向上抽象”**。
  + 3. **逻辑结构设计**
    - 目标：将概念模型转换为具体的逻辑结构（关系模式）。
    - 内容：
      * 确定各表的主键、外键和约束条件。
      * 进行规范化设计（如1NF、2NF、3NF、BCNF等）。
      * 确定各用户的子模式（视图）。
  + 4. **物理结构设计**
    - 目标：确定数据在存储设备上的组织方式。
    - 主要内容：
      * 索引结构（B+树索引、哈希索引、聚簇索引等）。
      * 存储路径与访问方法设计。
      * 数据存储分布与访问优化。
    - 主要设计工具：索引
  + 5. **数据库实施**
    - 建立数据库模式与表结构。
    - 导入初始数据并进行测试。
    - 编写应用程序与接口。
  + 6. **数据库运行与维护**
    - 监控数据库性能与安全。
    - 进行备份、恢复、优化与权限管理。
    - 定期调整索引与存储结构。

## 错题

【例1-1-2】DBMS是位于()的一层软件。

A.数据库和用户

B.数据库和操作系统

C.计算机硬件和操作系统

D.用户和操作系统

D，看第一章的图

3.下列关于数据库管理系统的说法,正确的是(多选)

A.数据库管理系统与操作系统有关,操作系统的类型决定了能够运行的数据库管理系统的类型

B数据库管理系统对数据库文件的访问必须经过操作系统实现才能实现

C.数据库应用程序可以不经过数据库管理系统而直接读取数据库文件

D.数据库管理系统对用户隐藏了数据库文件的存放位置和文件名

ABD；用户通过 SQL 操作数据库对象（表、视图、索引等），不需要关心这些对象在磁盘上对应的具体文件名或路径。

4.下列叙述中,正确的有(多选)。

A.数据库技术的根本目标是要解决数据库共享的问题

B.数据库系统中,数据库的物理结构必须与逻辑结构一致

C数据库系统是一个独立的系统,但需要操作系统的支持

D.数据库系统减少了数据冗余

AD；数据库系统DBS不是独立的，数据库管理系统DBMS是独立的

1.下列关于关系中主属性的描述,正确的是(多选)。

A生码所包含的属性一定是主属性

B外码所引用的属性一定是主属性

C候选码所包含的属性都是主属性

D.任何个主属性都可以唯一地标识表中的一行数据

ABC

6.对关系模型叙述正确的是(多选)。

A.建并在严格的数学理论、集合论和谓词演算公式的基础之上

B.微机DBMS绝大部分采取关系数据模型

C.用二维表表示关系模型是其一大特点

D.不具有连接操作的DBMS也可以是关系数据库系统

ABC

2.关系模型是最早出现的数据模型。

错误，最早的是数据模型

4.数据完整性在列级上的约束,包括(多选)。

A.数据类型约束

B.数据格式约束

C.取值范围约束

D空值约束

ABCD; 列级约束主要是针对单个字段（列）限制其取值的合法性; 完整性（Integrity）**指的是**保证数据库中存储的数据是正确、有效且一致的; 限定int长度属于数据格式约束

3.关系型数据库的完整性约束有两类,实体完整性和参照完整性。()

错误，不止2个

2.表一旦创建成功可以随意修改,不会对数据产生影响。

错误，表的结构修改（如增加/删除列、修改列类型、删除列约束）可能会影响已有数据。

4. 学生关系模式S(S#,Sname,Sex,Age),S的属性分别表示学生的学号、姓名、性别、

年龄。要在表S中删除一个属性“年龄”,可选用的SQL语句是()。

A. DEDETE Age from S

B. ALTER TABLE S DROP Age

C. UPDATE S Age

D. ALTER TABLE S DELETE 'Age'

B

【例5-2-2】可以在创建表时用()来创建唯一索引,也可以用()来创建唯一索引。

A.Create table Create index

B.设置主键约束,设置唯一约束

C.设置主键约束,Create index

D.以上都可以

B

- 创建唯一索引的方法有三种：

- 主键约束

- 唯一约束

- CREATE UNIQUE INDEX

2.下面关于视图的描述,正确的是(

A.如果视图引用多个表时,可以用delete命令删除数据

B.通过修改视图可以影响基表中的数据

C.修改基表中的数据完全不会影响视图中的数据

D.可以修改那些通过计算得到的字段,例如年龄

B

7.关于存储过程,以下哪些说法是正确的? (多选)。

A.存储过程是一组完成特定功能的SQL语句集

B.存储过程在首次执行时需要编译,之后会被缓存,从而提高执行效率

C.存储过程不能带有参数

D.存储过程不能返回结果集

AB

4.关于安全性控制,下面选项正确的是(多选).

A.数据库的授权机制可以控制到元组级别

B.数据库管理员的权限最高

C.数据库的权限包括查询、插入、更新和删除:

D.除数据库管理员外,其他人都没有向别人授予数据库操作的权力;

BC;A选项中最高只能到列组级别

6.安全性控制防范的主要对象不包括(多选).

A.合法用户

B不合语义的数据

C.非法操作

D.不正确的数据

ABD; 安全性针对非法用户

8.角色(多选)。

A.是一个拥有各种操作权限的用户

B.是一组拥有相同操作权限的用户

C数据库中为方便对用户和权限进行管理而建立的

D.一个用户可以拥有不同的角色

BCD

3.在数据库的安全性控制中,授权对象的约束范围越大,授权子系统就越灵活。

错误，越小越灵活

只有允许并发操作时,才有可能出现死锁。

正确

【例7-4-1】(多选)并发访问冲突指的是(多选)。

A.两个或多个事务同时读取同一个数据项

B.两个或多个事务同时修改同一个数据项

C.对同一数据同时访问时,某些事务读,其他事务改写

D.两个或多个事务同时删除数据项

BCD

7.事务是保证数据一致性的基本手段,事务处理技术主要包括(多选)

A.数据库备份技术

B.数据库恢复技术

C.数据库安全技术

D.并发控制技术

BD; 这就是一个知识点，记住就好

9.数据库管理系统用于进行并发性控制的机制是(多选)

A.封锁机制

B.检测机制

C.时间戳机制

D.完整性机制

AC

5.若数据库中只包含成功事务提交的结果,则此数据库就称为处于()状态。

A.安全

B一致

C.不安全

D.不一致

B

4.下列关于恢复数据库的说法,不正确的是(多选)。

A.在恢复数据库时不允许有用户访问数据库

B.恢复数据库时必须按照备份的顺序还原全部的备份

C.恢复数据库时,对是否有用户在使用数据库没有要求

D首先进行恢复的备份可以是差异备份和日志备份

AD

4.运算溢出属于系统故障。

错误，属于事务故障

【例9-2-4】数据库物理设计使用的主要工具是(多选)。

A.E-R 图

B.数据库查询语言

C数据库索引

D.数据库管理系统

C

1.下列属于数据库设计方法的有(多选)

A新奥尔良方法

B基于E-R模型的设计方法

C.第三范式的设计方法

D.面向对象的设计方法

E统一建模语言方法

ABCDE

4.下列陈述中,正确的是(多选)

A 2NF 必然属于 1NF

B 3NF 必然属于 2NF

C.3NF必然属于BCNF

D BCNF 必然属于3NF

ABD

8.关系规范化中的插入/删除操作异常是指(多选)。

A.不该删除的数据被删除

B.不该插入的数据被插入

C.应该删除的数据未被删除

D.应该插入的数据未被插入

AD

9.规范化过程主要是为克服下列(多选)的缺陷。

A.数据不一致性

B.结构不合理

C.冗余度大

D.数据库逻辑结构中的异常

CD；我觉得AB也是正确的？老实回答说A是C和D导致的结果；B描述不精确，（结构包括逻辑结构、存储结构，而规范化只能解决逻辑结构不一致问题）

5.在关系数据库的规范化理论中,在执行“分解”时,必须遵守规范化原则:保持原有的依赖关系和关系的相互独立性。

错误，应该是保持依赖关系和无损连接

2.设关系R和S的元数分别为r和s,则RxS的目数和元组数分别是为

A.r\*s, r+s

B.r-s, r\*s

C.r+s, r\*s

D.MAX(r,s), r+s

C

3.数据库中,数据的独立性是指()

A.数据库与数据库管理系统的相互独立

B.用户程序与DBMS的相互独立

C用户的应用程序与数据库中的数据是根互独立的

D.应用程序与数据库中数据的逻辑结构相互独立

C

12.关系R和关系S只有一个公共属性,T1是R与S等值连接的结果,T2是R与S自然连接的结果,下列叙述中正确的是()。

A.T1的属性个数等于T2的属性个数

B.T1的属性个数小于T2的属性个数

C.T1的属性个数大于或等于T2的属性个数

DT1的属性个数大于T2的属性个数

D，因为自然连接会**自动消除重复属性**，即公共属性只保留一次

19.在一个E-R模型中,有7个不同的实体集和5个不同的二元联系,其中有1个一对一联系、2个一对多联系和2个多对多联系,根据ER模型转换成关系模型的规则,转换成关系的数目最少是()。

A.12

B.11

C.7

D.9

D，多对多需要，每个实体需要

27.在SQL中CREATE VIEW 语用于建立视图,如果要求对视图更新时必须满足查询中表达式,应当在该语句中使用()短语。

A. WITH UPDATE

B.WITH INSERT

C.WITH DELETE

D.WITH CHECK OPTION

D

36.通常情况下,数据模型由()三部分组成。

A.数据结构、数据操作和完整性约束

B.层次、数据操作和完整性约束

C.关系、数据操作和完整性约束

D.层次、关系和完整性约束

A；数据模型本身更抽象，它包括概念模型、逻辑模型和物理模型。概念模型（ER）没有“数据操作”的概念；物理模型关注存储和访问路径，也不强调操作和完整性约束。所以在教材题目里，一般认为 **数据模型的三部分 = 逻辑模型的三要素**，因此选 A。

59.关于存储过程,说法正确的是()

A.存储过程是定义在客户端的程序

B.存储过程可以接收输入参数并以输出参数的形式将多个值返回给调用者

c.存储过程是自动执行的,无需调用

D.创建存储过程主要是为了减少服务器端的代码编写

B；D不对因为存储过程本身就是运行在服务器端的，并不能减少服务器端代码量。应该说是减少用户的代码编写量？

60.备份数据库的主要目的是为了防止数据丢失,下列可能造成数据丢失的是()

A存储数据的磁盘出现故

B.数据库管理系统出现故

C.因用户的不正常操作而更改了数据

D.服务器操作系统崩溃

A；单选选择A，多选选择全部。BCD还会导致数据不一致的问题。

4.下面关于数据库设计的说法中,正确的有()

A数据库设计中,概念设计要先于逻辑设计和物理设计

B数据库概念模式独立于任何数据库管理系统,不能直接用于数据库实现

C.物理设计阶段需要考虑表查询的效率

D.E-R 模型设计是逻辑没计阶段的任务

ABC

15.数据库管理员的职责包括(

A.负责整个数据库系统的建立

B负责整个数据库系统的管理

C负责整个数据库系统的维护和监控

D.负责数据库的重构

ABCD