数据库系统知识点总结

第一章

1、数据库系统相关概念

1、数据：描述事物的符号记录，包括数据的表现形式和数据解释两个部分。如数字、音频、图形、文本、图像、语言、视频等多种表现形式。经过数字化处理后存入计算机。

数据是信息的符号表示或载体。信息是数据的内涵是对数据的语义解释。

2、数据库（DB）：长期存储在计算机内、有组织、可共享的大量数据的集合。数据库中的数据按照一定的数据模型组织、描述和存储，具有娇小的冗余度、交稿的数据独立性和易扩展性，并可为各种用户共享。

3、数据库管理系统（DBMS）：位于用户和操作系统间的数据管理系统的一层数据管理软件。用途：科学地组织和存储数据，高效地获取和维护数据。包括数据定义功能，数据组织、存储和管理，数据库的事物管理和运行管理，数据库的建立和维护功能，其他功能。

4、数据库系统（DBS）：在计算机系统中引入数据库后的系统，一般由数据库。数据库管理系统（及其开发工具）、应用系统、数据库管理员构成。目的：存储信息并支持用户检索和更新所需的信息。

2、数据模型概念，作用及其 3 要素

模型：对现实世界中某个对象特征的模拟和抽象。

数据模型：是数据库中用来对现实世界数据特征的抽象的工具，是数据库中用于提供信息表示和操作手段的形式架构。

三要素：

（1） 数据结构：是所研究的对象类型的集合，是对系统静态特性的描述。

（2） 数据操作：对数据库中各种对象（型）的实例（值）所允许进行的操作的集合，包括操作及有关的操作规则，是对系统动态特性的描述。

（3） 数据的约束条件：是完整性规则的集合。完整性规则是给订的数据库模型中数据及其联系所具有的制约和依存规则，用一限定符合数据模型的数据库状态及其变化，以保证数据库的正确、有效、相容。

3、概念模型的作用？

概念模型实际上是显示世界到机器世界的一个中间层次。概念模型用于信息世界的建模，是现实世界到信息世界的第一层抽象，是数据库设计人进行数据库设计的有力工具，也是数据库设计人员和用户之间进行

交流所试用的语言。

4、概念模型中 ER 图的设计

E-R 图（实体—联系方法）提供了表示实体型、属性和联系的方法：

实体型：用矩形表示，矩形框内写明实体名。

属性：用椭圆形表示，并用无向边将其与相应的实体型连接起来。

5、数据库系统三级模式结构和二级映像功能，这种结构的优点？

三级模式结构：

（1） 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，是所有用户的公共数据视图。一个数据库只有一个模式。

模式的地位：是数据库系统模式结构的中间层，与数据的物理存储细节和硬件环境无关，与具体的应用程序、开发工具及高级程序设计语言无关。

模式定义的内容：数据的逻辑结构（数据项的名字、类型、取值范围等），数据之间的联系，数据有关的安全性、完整性要求

（2） 外模式：（子模式/用户模式）数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据库和逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的系统的逻辑表示。一个数据库可以有多个外模式。

外模式的地位：介于模式与应用之间

模式与外模式的关系：一对多。外模式通常是模式的子集。一个数据库可以有多个外模式。反映了不同的用户的应用需求、看待数据的方式、对数据保密的要求。对模式中同一数据，在外模式中的结构、类型、长度、保密级别等都可以不同。

外模式与应用的关系：一对多。同一外模式也可以为某一用户的多个应用系统所使用

但一个应用程序只能使用一个外模式

（3） 内模式：存储模式或内视图）是数据物理结构和存储方式的描述，是数据在数据库内部实际存储的表示方式：

记录的存储方式（顺序，B 树，hash 方法存储），索引的组织方式，数据是否压缩存储，数据是否加密。数据存储记录结构的规定，一个数据库只有一个内模式

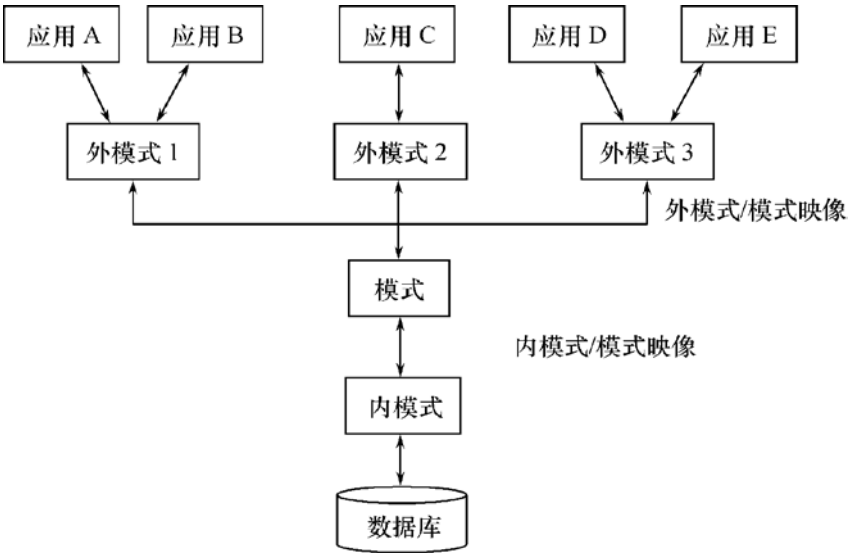
三级模式的优点：

（1）保证数据的独立性（内模式与模式分开物理独立；外模式与模式分开逻辑独立）

（2）简化用户窗口 （3）有利于数据共享 （4）利于数据的安全保密

（5）数据存储由 DBMS 管理（用户不用考虑存取路径等细节）

二级映像功能：



（1） 外模式/模式映像（应用可扩充性）

定义外模式(局部逻辑结构)与模式(全局逻辑结构)之间的对应关系，映象定义通常包含在各自外模式的描述中，每一个外模式，数据库系统都有一个外模式 / 模式映象。

用途：保证数据的逻辑独立性

当模式改变时，数据库管理员修改有关的外模式 / 模式映象，使外模式保持不变

应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

（2） 模式/内模式映像（空间利用率，存取效率）

模式/内模式映像是唯一的，它定义了数据全局逻辑结构与存储结构之间的对应关。数据库中模式 / 内模式映象是唯一的。该映象定义通常包含在模式描述中。

用途：保证数据的物理独立性

当数据库的存储结构改变了（例如选用了另一种存储结构），数据库管理员修改模式 / 内模式映象，使模式保持不变。应用程序不受影响。保证了数据与程序的物理独立性，简称数据的物理独立性。

优点：

（1）保证了数据库外模式的稳定性。

（2）从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般不需要修改。

（3）数据与程序之间的独立性，使得数据的定义和描述可以从应用程序中分离出去。

6、什么叫数据与程序的物理独立性？什么叫数据与程序的逻辑独立性？ 为什么数据库系统具有数据与程序的独立性？

1、数据与程序的逻辑独立性：当模式改变时，数据库管理员修改有关的外模式 / 模式映象，使外模式保持不变。从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

2、数据与程序的物理独立性：当数据库的存储结构改变了（例如选用了另一种存储结构），数据库管理员修改模式 / 内模式映象，使模式保持不变。应用程序不受影响。保证了数据与程序的物理独立性，简称数据的物理独立性。

数据库管理系统在三级模式之间错提供的二层影响保证了数据系统中的数据具有较高的逻辑独立性和物理独立性。

第二章：关系数据库

1、关系模型的 3 个组成部分及各部分所包括的主要内容。

1、关系数据结构：描述现实世界的实体以及实体间的各种联系。只包含单一的数据结构—关系。

2、关系操作

查询操作：选择、投影、连接、除、并、差、交、笛卡尔积等。

插入、删除、修改操作。

3、关系的完整性约束

实体完整性和参照完整性：关系模型必须满足的完整性约束条件称为关系的两个不变性，应该由关系系统自动支持。

用户定义的完整性：应用领域需要遵循的约束条件，体现了具体领域中的语义约束。

2、关系数据结构的形式化定义（各术语）

域：一组具有想通数据类型的值的集合。（用 D 表示）域中所包含的值的个数称为域的基数（用 m 表示）。

例:整数、实数等。

笛卡尔积：域上面的一个集合运算。给定一组域 D1，D2…Dn (可以是相同的域) D1，D2…Dn 的笛卡尔积为：D1×D2×…×Dn ＝｛(d1，d2…dn) | di ∈ Di，i＝1，2，…，n｝所有域的所有取值（n 元有序组）的一个组合。有序组的取值不能完全重复。

元组（Tuple）：笛卡尔积中每一个元素（d1，d2，…，dn）叫作一个 n 元组（n-tuple）或简称元组(Tuple)(张清玫，计算机专业，李勇)、(张清玫，计算机专业，刘晨)等都是元组。

分量（Component）：笛卡尔积元素（d1，d2，…，dn）中的每一个值 di 叫作一个分量。张清玫、计算机专业、李勇、刘晨等都是分量。

基数（Cardinal number）：若 Di（i＝1，2，…，n）为有限集， Di 中的集合元素个数称为 Di 的基数，用 mi（i＝1，2，…，n）表示，则 D1×D2×…×Dn 的基数 M （即元素<d1,d2,……dn>的个数）为为所有域的基数的累乘之积。

例：A={a,b}　　B={1,2,0}

A 与 B 的笛卡尔积={<a,0>,<a,1>,<a,2>,<b,0>,<b,1>,<b,2>}

B 与 A 的笛卡尔积=｛<0,a>,<0,b>,<1,a>,<1,b>,<2,a>,<2,b>｝

期中<0,a>等都是元祖，a、b、0、1、2 都是分量，基数 M=2*3=6，一共有六个元组。

关系：是笛卡尔积的有限子集，无限关系在数据库系统中是无意义的。D1×D2×…×Dn 的子集叫作在域 D1，D2，…，Dn 上的 n 元关系，表示为 R（D1，D2，…，Dn）

R：关系名　　n：关系的目或度（Degree）

（1）元组：关系中的每个元素是关系中的元组，通常用 t 表示。

（2）单元关系与二元关系：当 n=1 时，称该关系为单元关系（Unary relation）或一元关系；当 n=2 时，称该关系为二元关系（Binary relation）

（3）关系的表示：关系也是二维表，表的每行对应一个元组，表的每列对应一个域。

（4）属性：由于笛卡尔积不满足交换律，即(d1，d2，…，dn)≠(d2 ，d1 ，…，dn)

但关系满足交换律，即 (d1，d2，…，di ，dj ，…，dn ）=（ d1，d2 ，…，dj，di ，…，dn ）（i，j = 1，2，…，n）

解决方法：为关系的每个列附加一个属性名以取消关系元组的有序性；关系中不同列可以对应相同的域；为了加以区分，必须对每列起一个名字，称为属性（Attribute）；n 目关系必有 n 个属性。

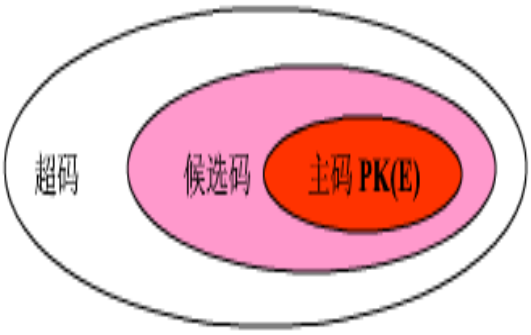
（5）码：

候选码：若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选码。

主属性：候选码的诸属性称为主属性，不包含在任何侯选码中的属性称为非主属性或非码属性。

超码：关系中能唯一标识元组的属性集，超码的任意超级仍是超码

候选码：任意真子集都不能成为超码的“最小”超



码，即在候选码中，若要再删除属性，就不是超码了。

3、外码，主码，候选码的概念

候选码：若关系中的某一属性组的职能唯一地标识一个元组，则称该属性组为候选码。

主码：若一个关系有多个候选码，则选定期中一个为主码。

外部码：设 F 是基本关系 R 的一个或一组属性。但不是关系 R 的码，如果 F 与基本关系 S 的主码 K 想对应，则称 F 是基本关系 R 的外部码，简称外码。

4、关系的 3 类完整性约束概念

实体完整性：若属性（指一个或一组属性）A 是基本关系 R 的主属性， A 不能取空值。

参照完整性：若属性（或属性组）F 是基本关系 R 的外码，它是基本关系 S 的主码 K 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为：或者取空值（F 的每个属性值均为空值）；或者等于 S 中某个元组的主码值。

用户定义的完整性：针对某一具体关系数据库的约束条件。反映某一具体应用所设计的数据必须满足的语义要求。

5、关系操作的特点，关系代数中的各种运算

关系操作的特点是集合操作方式，即操作的对象和结果是集合。

关系代数：

运算符		含义	运算符		含义
集合运算符	∪ - ∩ ×	并 差 交 笛卡尔积	比较运算符	> ≥ < ≤ = <>	大于 大于等于 小于 小于等于 等于 不等于
专门的关系运算符	σ π ⋈ ÷	选择 投影 连接 除	逻辑运算符	¬ ∧ ∨	非 与 或

1、并（R∪S）仍为 n 目关系，由属于 R 或属于 S 的元组组成。R∪S = { t|t ∈R ∨ t∈S }

2、差（R－S）仍为 n 目关系，由属于 R 而不属于 S 的所有元组组成。R -S = { t|t ∈R ∧ t∉S }

3、交（R∩S）仍为 n 目关系，由既属于 R 又属于 S 的元组组成。R∩S = { t|t ∈R ∧ t∈S } R∩S = R －(R-S）

4、笛卡尔积 R: n 目关系，k1 个元组；S: m 目关系，k2 个元组；R×S。

5、R，t∈R，t[Ai]

设关系模式为 R(A1，A2，…，An)，它的一个关系设为 R，t∈R 表示 t 是 R 的一个元组，t[Ai]则表示元组 t 中相应于属性 Ai 的一个分量。

6、A，t[A]， A

若 A={Ai1，Ai2，…，Aik}，其中 Ai1，Ai2，…，Aik 是 A1，A2，…，An 中的一部分，则 A 称为属性列或属性组；t[A]=(t[Ai1]，t[Ai2]，…，t[Aik])表示元组 t 在属性列 A 上诸分量的集合；A 则表示{A1，A2，…，An}中去掉{Ai1，Ai2，…，Aik}后剩余的属性组。

7、tr ts

R 为 n 目关系，S 为 m 目关系。tr ∈R，ts∈S， tr ts 称为元组的连接。tr ts 是一个 n + m 列的元组，前 n

个分量为 R 中的一个 n 元组，后 m 个分量为 S 中的一个 m 元组。

8、象集 Zx

给定一个关系 R（X，Z），X 和 Z 为属性组。当 t[X]=x 时，x 在 R 中的象集（Images Set）为：Zx={t[Z]|t □ R，t[X]=x}。

它表示 R 中属性组 X 上的分量值为 x 的诸元组在 Z 上分量的集合

9、选择：选择又称为限制（Restriction）σ:对元组按照条件进行筛选。在关系 R 中选择满足给定条件的诸元组 σ F(R) = {t|t∈R∧F(t)= '真'}。

10、投影：投影运算符 π 的含义：从 R 中选择出若干属性列组成新的关系 πA(R) = { t[A] | t □ R } A：R 中的属性列

投影操作主要是从列的角度进行运算。但投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组（避免重复行）。

11、连接：连接也称为 θ 连接 $\bowtie_{A\theta B}$ ：两张表中的元组有条件的串接。

从两个关系的笛卡尔积中选取属性间满足一定条件的元组 R $\bowtie_{A\theta B}$ S = { | tr □ ts R □ S tr[A]θts[B] }

A 和 B：分别为 R 和 S 上度数相等且可比的属性组 θ：比较运算符

连接运算从 R 和 S 的广义笛卡尔积 R×S 中选取（R 关系）在 A 属性组上的值与（S 关系）在 B 属性组上值满足比较关系 θ 的元组。

等值连接：θ 为“＝”的连接运算称为等值连接。

从关系 *R* 与 *S* 的广义笛卡尔积中选取 *A*、*B* 属性值相等的那些元组，即等值连接为：

$$R \bowtie_{A=B} S = \{ tr\ ts \mid tr \in R \wedge ts \in S \wedge tr[A] = ts[B] \}$$

自然连接是一种特殊的等值连接：两个关系中进行比较的分量必须是相同的属性组，在结果中把重复的属性列去掉。

自然连接的含义：R 和 S 具有相同的属性组 B。 R $\bowtie$ S = {tr ts| tr □ R ts □ S tr[B] = ts[B] }

外连接：如果把舍弃的元组也保存在结果关系中，而在其他属性上填空值(Null)，这种连接就叫做外连接。

左外连接：如果只把左边关系 R 中要舍弃的元组保留就叫做左外连接(LEFT OUTER JOIN 或 LEFT JOIN)。

右外连接：如果只把右边关系 S 中要舍弃的元组保留就叫做右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN)。

12、除÷ ：给定关系 R (X，Y) 和 S (Y，Z)，其中 X，Y，Z 为属性组；R 中的 Y 与 S 中的 Y 可以有不同的属性名，但必须出自相同的域集；R 与 S 的除运算得到一个新的关系 P(X)，P 是 R 中满足下列条件的元组在 X 属性列上的投影：

(若)元组在 X 上分量值 x 的象集 Yx 包含 S 在 Y 上投影的集合，记作：

R÷S = {tr [X] | tr □ R Y (S) □ Yx } Yx：x 在 R 中的象集，x =tr[X]

第三章：关系数据库标准语言 SQL

1、SQL 语言的特点

1、综合统一。
2、高度非过程化。
3、面向集合的操作方式。
4、以同一种语法结构提供多种使用方式
5、语言简洁、易学易用。

2、SQL 语言的数据查询，数据定义，数据操纵功能，这些分别有哪些 SQL 语句？

1、数据定义：定义数据库中的基本对象、模式(架构)定义、表定义、视图和索引。

操作对象	操作方式		
	创建	删除	修改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	DROP+CREATE
索引	CREATE INDEX	DROP INDEX	

注意：SQL(Oracle 除外)一般不提供修改视图定义和索引定义的操作，需要先删除再重建

（1）模式：

定义模式：**CREATE SCHEMA <模式名> AUTHORIZATION <用户名>[<表定义子句>|<视图定义子句>|<授权定义子句>]**

例：CREATE **SCHEMA** TEST AUTHORIZATION ZHANG

CREATE **TABLE** TAB1(COL1 SMALLINT， COL2 INT，COL3 CHAR(20)，COL4 NUMERIC(10，3)，COL5 DECIMAL(5，2))；

为用户 ZHANG 创建了一个模式 TEST，并在其中定义了一个表 TAB1。

删除模式：**DROP SCHEMA <模式名> <CASCADE|RESTRICT>**

CASCADE(级联)：删除模式的同时把该模式中所有的数据库对象全部删除

RESTRICT(限制)； 没有任何下属的对象时 才能执行。

（2）表：

定义基本表：**CREATE TABLE <表名>**

（<列名> <数据类型>[<列级完整性约束条件>]

[，<列名> <数据类型>[<列级完整性约束条件>]] …

[，<表级完整性约束条件>])；

列级完整性约束--涉及到该表的一个属性

- NOT NULL ： 非空值约束
- UNIQUE：唯一性（单值约束）约束
- PRIMARY KEY:主码约束
- DEFAULT <默认值>:默认（缺省）约束
- Check < (逻辑表达式) >:核查约束,定义校验条件
- NOT NULL ： 非空值约束
- UNIQUE：唯一性（单值约束）约束
- PRIMARY KEY:主码约束
- DEFAULT <默认值>:默认（缺省）约束
- Check < (逻辑表达式) >:核查约束,定义校验条件

表级完整性约束--涉及到该表的一个或多个属性。

- UNIQUE(属性列列表) ： 限定各列取值唯一
- PRIMARY KEY (属性列列表) ： 指定主码
- FOREIGN KEY (属性列列表) REFERENCES <表名> [(属性列列表)]
- Check(<逻辑表达式>) ： 检查约束
- PRIMARY KEY 与 UNIQUE 的区别？

例：建立“学生”表 Student，学号是主码，姓名取值唯一

```
CREATE TABLE Student
  (Sno    CHAR(9) PRIMARY KEY，/*主码*/
        Sname   CHAR(20) UNIQUE，        /* Sname 取唯一值*/
        Ssex    CHAR(2)，Sage    SMALLINT，Sdept   CHAR(20))；
```

数据类型

数据类型	含义
CH数据类型 AR(n)	长度为n的定长字符串
VARCHAR(n)	最大长度为n的变长字符串
INT	长整数（也可以写作INTEGER）
SMALLINT	短整数
NUMERIC(p，d)	定点数，由p位数字（不包括符号、小数点）组成，小数后面有d位数字
REAL	取决于机器精度的浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为n位数字
DATE	日期，包含年、月、日，格式为YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为HH:MM:SS

修改基本表：**ALTER TABLE <表名>**

[ADD <新列名> <数据类型> [完整性约束]]

[DROP <列名> |<完整性约束名>]

[ALTER COLUMN<列名> <数据类型>];

例：向 Student 表**增加**“入学时间”列，其数据类型为日期型

ALTER TABLE Student **ADD** S_entrance DATE；

不论基本表中原来是否已有数据，新增加的列一律为空值

将年龄的数据类型由字符型（假设原来的数据类型是字符型）**改为**整数

ALTER TABLE Student **ALTER COLUMN** Sage INT；

注：修改原有的列定义有可能会破坏已有数据

增加课程名称必须取唯一值的约束条件。

ALTER TABLE Course ADD UNIQUE(Cname);

直接**删除属性列**:(新标准) 例： ALTER TABLE Student **Drop Sage**；

删除基本表 ：**DROP TABLE <表名>［RESTRICT| CASCADE］;**

RESTRICT：(受限) 欲删除的基本表不能被其他表的约束所引用，如果存在依赖该表的对象（触发器，视图等），则此表不能被删除。

CASCADE：（级联）在删除基本表的同时，相关的依赖对象一起删除。

例：删除 Student 表 DROP TABLE Student CASCADE ;

基本表定义被删除，数据被删除；表上建立的索引、视图、触发器等一般也将被删除 。

DROP TABLE时，SQL99 与 3个RDBMS的处理策略比较

序号	标准及主流数据库的处理方式 / 依赖基本表的对象	SQL99		Kingbase ES		ORACLE 9i		MS SQL SERVER 2005
		R	C	R	C		C	
1.	索引	无规定		√	√	√	√	√
2.	视图	×	√	×	√	√ 保留	√ 保留	√ 保留
3.	DEFAULT，PRIMARY KEY，CHECK（只含该表的列）NOT NULL等约束	√	√	√	√	√	√	√
4.	Foreign Key	×	√	×	√	×	√	×
5.	TRIGGER	×	√	×	√	√	√	√
6.	函数或存储过程	×	√	√ 保留	√ 保留	√ 保留	√ 保留	√ 保留

R表示RESTRICT，C表示CASCADE

'×'表示不能删除基本表，'√'表示能删除基本表，'保留'表示删除基本表后，还保留依赖对象

（3）索引：

建立索引的目的：加快查询速度。

DBA 或 表的属主（即建立表的人）（显式）；DBMS 一般会自动建立以下约束列上的索引（隐式）PRIMARY KEY UNIQUE 建立索引。

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表名> (<列名>[<次序>][,<列名>[<次序>]]…);

用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。默认：ASC。

UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录

CLUSTER 表示要建立的索引是聚簇索引。索引项顺序与表中记录的物理顺序一致。

聚簇索引 CLUSTER

例：CREATE CLUSTER INDEX Stusname ON Student(Sname)；

在 Student 表的 Sname（姓名）列上建立一个聚簇索引，而 且 Student 表中的记录将按照 Sname 值的升序存放。

一个基本表上最多只能建立一个聚簇索引；在最经常查询的列上建立聚簇索引以提高查询效率；经常更新的列不宜建立聚簇索引。

唯一值索引 UNIQUE

例：为学生-课程数据库中的 Student，Course，SC 三个表建 立索引

CREATE UNIQUE INDEX SCno ON SC(Sno ASC，Cno DESC)；

对于已含重复值的属性列不能建 UNIQUE 索引 。对某个列建立 UNIQUE 索引后，插入新记录时 DBMS 会自动检查新记录在该列上是否取了重复值。这相当于增加了一个 UNIQUE 约束。

删除索引 ：**DROP INDEX <索引名>；**

删除索引时，系统会从数据字典中删去有关该索引的描述。

例： 删除 Student 表的 Stusname 索引：DROP INDEX Stusname

2、数据查询：基本格式

Select A1，A2，…，An

From R1,R2,.., Rm

Where F

（1）单表查询

●选择表中的若干列 （投影）

查询指定列 （相当于 π A(R)，A= A1，A2，…，An ）

例：查询全体学生的学号与姓名

SELECT Sno，Sname

FROM Student；

选出所有属性列：在 SELECT 关键字后面列出所有列名按用户指定顺序显示。

将<目标列表达式>指定为 *按关系模式中的属性顺序显示。

例：查询全体学生的详细记录

SELECT Sno，Sname，Ssex，Sdept , Sage

FROM Student；

或 SELECT *

FROM Student；

P.S: SELECT 子句的<目标列表达式>可以为：

算术表达式：

例：查全体学生的姓名及其出生年份

SELECT Sname，2011-Sage /*假定当年的年份为 2011 年*/

FROM Student；

输出结果：

Sname	2011-Sage
李勇	1991
刘晨	1992

字符串常量、函数

例：查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名

SELECT Sname，‘Year of Birth: ’，2004-Sage，ISLOWER(Sdept)

FROM Student；

输出结果：

Sname	'Year of Birth:'	2004-Sage	ISLOWER(Sdept)
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	is

列别名

SELECT Sname *as* NAME,'Year of Birth: ' *as* BIRTH，
2011-Sage *as* BIRTHDAY,LOWER(Sdept) *as* DEPARTMENT
FROM Student;
输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1991	cs
刘晨	Year of Birth:	1992	is

●选择表中的若干元组（选择）
消除重复性：
指定 DISTINCT 关键词，去掉表中重复的行
SELECT **DISTINCT** Sno FROM SC；
注意 DISTINCT 短语的作用范围是所有目标列
错误的写法 SELECT DISTINCT Cno，DISTINCT Grade FROM SC;
正确的写法 SELECT DISTINCT Cno，Grade FROM SC;
SELECT 子句缺省情况是保留重复元组(ALL),
例：查询选修了课程的学生学号。**SELECT Sno FROM SC；**等价于：
SELECT ALL Sno FROM SC；
WHERE 子句常用的查询条件（相当于σF）

查询条件	谓词
比较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空值	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

优先级

使用比较运算符 或逻辑运算符 NOT + 比较运算符
例：查询计算机科学系全体学生的名单
SELECT Sname
FROM Student
WHERE Sdept=‘CS’；
例：查询所有年龄在 20 岁以下的学生姓名及其年龄
SELECT Sname，Sage
FROM Student
WHERE Sage < 20； //NOT Sage>=20
谓词: **BETWEEN … AND … NOT BETWEEN … AND …**
例；查询年龄在 20~23 岁（包括 20 岁和 23 岁）之间的学生的姓名、系别和年龄
SELECT Sname，Sdept，Sage
FROM Student
WHERE Sage BETWEEN 20 AND 23；//Sage>=20 and Sage<=23
例：查询年龄不在 20~23 岁之间的学生姓名、系别和年龄
SELECT Sname，Sdept，Sage
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23；//Sage<20 or Sage>23

IN <值表>， NOT IN <值表>

例：查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别

```
SELECT Sname，Ssex
FROM    Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' );
```

例：查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别

```
SELECT Sname，Ssex
FROM Student
 WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' );
```

[NOT] LIKE ‘<匹配串>’ [ESCAPE ‘ <换码字符>’]

匹配串为固定字符串

例：查询学号为 200215121 的学生的详细情况。

```
SELECT *                                    SELECT   *
FROM   Student          等价于：            FROM   Student
WHERE   Sno LIKE ‘200215121';               WHERE Sno = ' 200215121 ';
```

匹配串为含通配符的字符串

%：代表任意长度（可以是 0）的字符串

_：代表任意单个字符

字符串本身就含有 % 或 _ 时，在% 或 _ 之前加上转义符“\”要使用 ESCAPE ‘<换码字符>’将通配符转义为普通字符。如果‘\’ 要作为一个普通字符，用连续两个‘\’ 表示一个真正的‘\’。

例：查询以"DB_"开头，且倒数第 3 个字符为 i 的课程的详细情况。

```
SELECT   *
FROM     Course
WHERE   Cname LIKE   'DB\_%i_ _' ESCAPE '\  ‘;
```

IS NULL 或 IS NOT NULL “IS” 不能用 “=” 代替

例：查所有有成绩的学生学号和课程号

```
SELECT Sno，Cno
FROM   SC
WHERE   Grade IS NOT NULL;
```

逻辑运算符：AND 和 OR 来联结多个查询条件，AND 的优先级高于 OR，可以用括号改变优先级，可用来实现多种其他谓词。

[NOT] IN

[NOT] BETWEEN … AND …

改写 例：查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' )
```

可改写为：

```
SELECT Sname，Ssex
FROM     Student
WHERE   Sdept= ' IS ' OR Sdept= ' MA' OR Sdept= ' CS '
```

● ORDER BY 子句：对查询结果排序

可以按一个或多个属性列排序：升序：ASC；降序：DESC；缺省值为升序。

当排序列含空值时：空值最大

ASC：排序列为空值的元组最后显示；DESC：排序列为空值的元组最先显示。

例：查询选修了 3 号课程的学生的学号及其成绩，查询结果按分数降序排列

SELECT Sno，Grade
FROM　SC
WHERE　Cno= ' 3 '
ORDER BY Grade DESC；

● 聚集函数：对查询结果集中的某列进行计算或统计。

计数

COUNT（[DISTINCT|ALL] *）

COUNT（[DISTINCT|ALL] <列名>）

计算总和 **SUM（[DISTINCT|ALL] <列名>）**

计算平均值　**AVG（[DISTINCT|ALL] <列名>）**

最大最小值

MAX（[DISTINCT|ALL] <列名>）

MIN（[DISTINCT|ALL] <列名>）

例：查询学生 200215012 选修课程的总学分数

SELECT SUM(Ccredit)
FROM　SC，　Course
WHER Sno='200215012' AND SC.Cno=Course.Cno;

注：除 Count（*）,都要跳过空值；Where 子句不能使用聚集函数。

● GROUP BY 子句：对查询结果分组。

用途细化聚集函数的作用对象

未对查询结果分组，聚集函数将作用于整个查询结果

对查询结果分组后，聚集函数将分别作用于每个组

使用 **GROUP BY** 后：其 SELECT 子句的列名列表中只能出现分组属性和集函数。

如果分组后还要按照条件对这些组进行筛选，可使用 **having** 短语指定筛选条件

例：　查询选修了 3 门以上课程的学生学号

SELECT Sno
FROM　SC
GROUP BY Sno
HAVING　COUNT(*) >3；

例：查询有 3 门以上课程是 90 分以上学生的学号及（90 分以上的）课程数

SELECT Sno, COUNT(*)
FROM　SC
Where　Grade >90
GROUP BY Sno
HAVING　COUNT(*) >3；

HAVING 短语与 WHERE 子句的区别：作用对象不同

WHERE 子句作用于基表或视图，从中选择满足条件的元组

HAVING 短语作用于组，从中选择满足条件的组。

3、数据更新

（1）插入数据

插入元组

INSERT

[INTO] <表名> [(<属性列 1>[，<属性列 2 >…)]

VALUES (<常量 1> [，<常量 2>]　…　)

功能：将新元组插入指定表中；新元组的属性列 1 的值为常量 1，属性列 2 的值为常量 2，…。

INTO 子句：属性列的顺序可与表定义中的顺序不一致，但须指定列名；没有指定属性列，表示要插入的

是一条完整的元组；指定部分属性列，未指定的属性列取空值，具有 NOT NULL 的属性列除外。

VALUES 子句：提供的值必须与 INTO 子句匹配，值的个数，值的类型。

例：将一个新学生元组（学号：200215128；姓名：陈冬；性别：男；所在系：IS；年龄：18 岁）插入到 Student 表中

```
INSERT
INTO   Student (Sno，Sname，Ssex，Sdept，Sage)
VALUES ('200215128'，'陈冬'，'男'，'IS'，18)；
```

例： 将学生张成民的信息插入到 Student 表中

```
INSERT
INTO   Student
VALUES (‘200215126’，  ‘张成民’，  ‘男’，18，'CS');
```

例：插入一条选课记录('200215128'，'1 ')。

```
INSERT
INTO SC(Sno，Cno)
VALUES (‘ 200215128 ’，‘ 1 ’ )；
```

RDBMS 将在新插入记录的 Grade 列上自动地赋空值。或者：

```
INSERT
INTO SC
VALUES (‘ 200215128 ’，‘ 1 ’，NULL)；
```

因为没有指出 SC 的属性名，在 GRADE 列上要明确给出空值

（2）修改数据

UPDATE <表名>

SET <列名>=<表达式>[，<列名>=<表达式>]…

[WHERE <条件>]；

SET 子句：指定修改方式，要修改的列，修改后取值：<表达式>。

WHERE 子句：指定要修改的元组，，缺省表示要修改表中的所有元组。

功能：修改指定表中满足 WHERE 子句条件的元组。

修改某一个元组的值

例：将学生 200215121 的年龄改为 22 岁

```
UPDATE   Student
SET Sage=22
WHERE   Sno=' 200215121 '；
```

修改多个元组的值

例：将所有学生的年龄增加 1 岁

```
UPDATE Student
SET Sage= Sage+1；
```

带子查询的修改语句　　子查询须放在比较运算符之后

例：将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET   Grade=0
WHERE   'CS'=
                (SELETE Sdept
                 FROM   Student
                 WHERE   Student.Sno = SC.Sno)；
```

（3）删除数据

DELETE

FROM　　<表名>

[WHERE <条件>]；

功能：删除指定表中满足 WHERE 子句条件的元组。

WHERE 子句：指定要删除的元组；缺省表示要删除表中的全部元组，表的定义仍在数据字典中。

删除某一个元组的值

例：删除学号为 200215128 的学生记录

```
DELETE
 FROM Student
 WHERE Sno= 200215128 ';
```

删除多个元组的值

例：删除所有的学生选课记录

```
DELETE
FROM SC;
```

带子查询的删除语句

例：删除计算机科学系所有学生的选课记录

```
DELETE
FROM SC
WHERE   'CS'=
          (SELETE Sdept
          FROM Student
         WHERE Student.Sno=SC.Sno);
```

4、视　图

特点：虚表，是从一个或几个基本表（或视图）导出的表；只存放视图的定义，不存放视图对应的数据；基表中的数据发生变化，从视图中查询出的数据也随之改变。

基于视图的操作： 查询、删除、受限更新、定义基于该视图的新视图。

（1）定义视图

建立视图

CREATE　VIEW <视图名>　[(<列名>　[，<列名>]…)]

AS　<子查询>

[WITH　CHECK　OPTION]；

子查询：不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION：表示对视图进行 UPDATE，INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

组成视图的属性列名：全部省略或全部指定，但在下列三种情况下必须明确指定组成视图的所有列名：

某个目标列不是单纯的属性名，而是聚集函数或列表达式；

多表连接时选出了几个同名列作为视图的字段；

需要在视图中为某个列启用新的名字。

RDBMS 执行 CREATE VIEW 语句时只是把视图定义存入数据字典，并不执行其中的 SELECT 语句。在对视图查询时，按视图的定义从基本表中将数据查出。

行列子集视图：从单个基本表导出 ，只是去掉了基本表的某些行和某些列保留了主码

例：建立信息系学生的视图

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

WITH CHECK OPTION 例：建立信息系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有信息系的学生

```
CREATE VIEW IS_Student
AS
SELECT Sno，Sname，Sage
FROM    Student
WHERE    Sdept= 'IS'
WITH CHECK OPTION；
```

加上了 WITH CHECK OPTION 子句：RDBMS 对 IS_Student 视图的更新操作：修改操作：自动加上 Sdept= 'IS'的条件；删除操作：自动加上 Sdept= 'IS'的条件；插入操作：自动检查 Sdept 属性值是否为'IS' 。如果不是，则拒绝该插入操作。如果没有提供 Sdept 属性值，则自动定义 Sdept 为'IS'。

基于多个基表的视图

例：建立信息系选修了 1 号课程的学生视图

```
CREATE VIEW IS_S1(Sno，Sname，Grade)
AS
SELECT Student.Sno，Sname，Grade
FROM    Student，SC
WHERE    Sdept= 'IS' AND
                Student.Sno=SC.Sno AND
                SC.Cno= '1'；
```

基于视图的视图

例：建立信息系选修了 1 号课程且成绩在 90 分以上的学生的视图

```
CREATE VIEW IS_S2
AS
SELECT Sno，Sname，Grade
FROM    IS_S1
WHERE    Grade>=90；
```

由于视图中的数据不会实际存储，所以定义视图时可根据应用的需要，设置一些派生属性列或虚拟列，以便于查询和统计。

以 SELECT * 方式创建的视图可扩充性差，应尽可能避免 。

缺点：修改基表 Student 的结构后，Student 表与 F_Student 视图的映象关系被破坏，导致该视图不能正确工作。

删除视图 **DROP　VIEW　<视图名>；**

该语句从数据字典中删除指定的视图定义。如果该视图上还导出了其他视图，使用 CASCADE 级联删除语句，把该视图和由它导出的所有视图一起删除 。删除基表时，由该基表导出的所有视图定义都必须显式地使用 DROP VIEW 语句删除。

例：删除视图 BT_S：

```
DROP VIEW BT_S;
```

删除视图 IS_S1：拒绝执行

级联删除：DROP VIEW IS_S1 CASCADE;

（2） 查询视图：查询视图与查询基本表相同,视图定义后，就可以像对待基本表一样对视图进行查询（SELECT）操作。

视图消解法（View Resolution）进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义 ；转换成等价的对基本表的查询，把视图定义中的子查询与用户的查询结合起来；执行修正后的查询。

例：在信息系学生的视图中找出年龄小于 20 岁的学生

SELECT Sno，Sage
FROM IS_Student
WHERE Sage<20；

视图消解转换后的查询语句为：

SELECT Sno，Sage
FROM Student
WHERE Sdept= 'IS' AND Sage<20；

例：在 S_G 视图中查询平均成绩在 90 分以上的学生学号和平均成绩

SELECT *
FROM S_G
WHERE Gavg>=90；

S_G 视图的子查询定义：

CREATE VIEW S_G (Sno，Gavg)
AS
SELECT Sno，AVG(Grade)
FROM SC
GROUP BY Sno；

（3） 更新视图

DBMS 实现视图更新的方法：转换为对基本表的更新。视图消解法（View Resolution）。

只有对成为“可更新”视图才能进行更新操作。

SQL2 对“可更新”视图给出正式定义：从关系 R 选出某些属性（用 select 而不是 select distinct）定义的视图，R 本身可以是可更新的视图；Where 中不能嵌套涉及 R 的子查询；Select 必须包括足够多的属性，所有 not null 的属性必须包括。

对于视图元组的更新操作（INSERT、DELETE、UPDATA），有以下三条规则：

如果一个视图是从多个基本表使用联接操作导出的，那么不允许对这个视图执行更新操作。

如果在导出视图的过程中，使用了分组和聚集函数操作，也不允许对这个视图执行更新操作。

行列子集视图可以执行更新操作。

在 SQL2 中，允许更新的视图在定义时，必须加上“WITH CHECK OPTION”短语。DBMS 在更新视图时会进行检查，防止用户通过视图对不属于视图范围内的基本表数据进行更新。

例：将信息系学生视图 IS_Student 中学号 200215122 的学生姓名改为“刘辰”。

UPDATE IS_Student
SET Sname= '刘辰'
WHERE Sno= ' 200215122 ';

转换后的语句：

UPDATE Student
SET Sname= '刘辰'
WHERE Sno= ' 200215122 ' AND Sdept= 'IS';

例：向信息系学生视图 IS_S 中插入一个新的学生记录：200215129，赵新，20 岁

INSERT
INTO IS_Student
VALUES(‘95029’，‘赵新’，20)；

转换为对基本表的更新：

INSERT
INTO Student(Sno，Sname，Sage，Sdept)
VALUES(‘200215129 '，'赵新'，20，'IS')；

例：删除信息系学生视图 IS_Student 中学号为 200215129 的记录

DELETE
FROM IS_Student
WHERE Sno= ' 200215129 ';
转换为对基本表的更新：
DELETE
FROM Student
WHERE Sno= ' 200215129 ' AND Sdept= 'IS';

更新视图的限制：一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新。对其他类型视图的更新不同系统有不同限制：

(1) 若视图是由两个以上基本表导出的，则此视图不允许更新

(2) 若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

(3) 若视图的字段来自集函数，则此视图不允许更新。

(4) 若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

(5) 若视图定义中含有 DISTINCT 短语，则此视图不允许更新。

(6) 若视图定义中有嵌套查询，并且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。

(7) 一个不允许更新的视图上定义的视图也不允许更新

（4） 视图的作用

1. 视图能够简化用户的操作，聚焦于所关心的数据上。
2. 视图使用户能以多种角度看待同一数据，增加灵活性 。
3. 视图对重构数据库提供了一定程度的逻辑独立性。
4. 视图能够对机密数据提供安全保护。
5. 适当的利用视图可以更清晰的表达查询。

3、用 SQL 实现复杂的查询

略

4、什么是基本表？什么是视图？两者的区别和联系是什么？为何要引入视图？

基本表：是独立存在的表。在 SQL 中，一个关系对应于一个表。

视图：是从一个或多个基本表所导出的表。视图本身并不独立存储在数据库中，是一个虚表，即数据库中之存放视图的定义而不存放其所对应的数据，这些数据仍然存放在导出的视图的基本表中。视图在概念上与基本表等同，用户可以像使用基本表那样使用视图，可以在视图上再定义视图。

为何要引入视图：

1. 视图能够简化用户的操作，聚焦于所关心的数据上。
2. 视图使用户能以多种角度看待同一数据，增加灵活性 。
3. 视图对重构数据库提供了一定程度的逻辑独立性。
4. 视图能够对机密数据提供安全保护。
5. 适当的利用视图可以更清晰的表达查询。

第四章：数据库的安全性

1、什么数据库安全性

数据库的安全性是指保护数据库以防止非法使用所造成的数据泄漏、更改或破坏。

2、数据库安全控制的常用方法和技术

1、用户标识和鉴别：由系统提供一定的方式，让用户表示自己的名字或身份。每次用户要求进入系统时，由系统进行核对，通过鉴定后系统才能提供使用权。

2、存取控制：通过用户权限定义和合法权限检查确保只有拥有合法权限的用户才能访问数据库，所有未授权人员均无法存取数据。

3、视图机制：为不同的用户定义视图，通过视图机制把要保密的数据对无权限用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

4、审计：建立审计日志，把用户对数据库的所有操作自动记录下来并放入审计日志中。DBA 可以利用审计跟踪信息来重现导致数据库现状的一系列事件，找出非法存取数据的人、时间和内容等。

5、数据加密：对所存储和传输的数据进行加密处理，从而使得不掌握解密算法的人无法获知数据。

3、什么是数据库中的自主存取控制方法和强制存取控制方法？为什么强制存取控制提供了更高级别的安全性？

自主存取控制方法：定义各个用户对不同数据对象的存取权限。当用户要访问数据库时，首先要检查其存取权限，以防止非法用户对数据库进行存取。

“自主存取控制”中“自主”的含义：用户可以将自己所拥有的存取权限“自主”地授予他人，即用户具有一定的“自主”权。

强制存取控制方法：每一个数据对象被（强制地）标以一定的加密级别，每位用户也被（强制地）授予某一级别的许可证。系统规定只有具有某一许可证级别的用户才能存取加密级别的数据对象。

强制存取控制（MAC）是对数据本身进行密级标记，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提高了更高级别的安全性。

4、自主存取控制语句：授权与收回语句。

GRANT 语句和 REVOKE 语句实现。

关系数据库系统中存取控制权限：

对象类型	对象	操作类型
数据库模式	模式	CREATE SCHEMA
	基本表	CREATE TABLE，ALTER TABLE
	视图	CREATE VIEW
	索引	CREATE INDEX
数据	基本表和视图	SELECT，INSERT，UPDATE，DELETEREFERENCES，ALL PRIVILEGES
数据	属性列	SELECT，INSERT，UPDATE，REFERENCES ALL PRIVILEGES

1、GRANT（授权）

GRANT 语句的一般格式：

GRANT <权限>[,<权限>]...

ON <对象类型> <对象名> ，[<对象类型> <对象名>]

TO <用户>[,<用户>]...

[WITH GRANT OPTION];

将对指定操作对象的指定操作权限授予指定的用户

发出 GRANT：DBA，数据库对象创建者（即属主 Owner），拥有该权限的用户

接受权限的用户 ：一个或多个具体用户；PUBLIC（全体用户）。

例：把查询 Student 表权限授给用户 U1

GRANT SELECT
ON TABLE Student
TO U1;

例：把查询 Student 表和修改学生学号的权限授给用户 U4

GRANT UPDATE(Sno), SELECT
ON *TABLE* Student
TO U4;

对属性列的授权时必须明确指出相应属性列名

2、REVOKE：授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回

REVOKE 语句的一般格式为：

REVOKE <权限>[,<权限>]...
ON <对象类型> <对象名> >[,<对象类型> <对象名>] …
FROM <用户>[,<用户>]... >[CASCADE|RESTRICT];

把指定对象的指定操作权限从指定用户处收回。

例：把用户 U4 修改学生学号的权限收回

REVOKE *UPDATE(Sno)*
ON *TABLE* Student
FROM U4;

例：收回所有用户对表 SC 的查询权限

REVOKE SELECT
ON *TABLE* SC
FROM **PUBLIC;**

例：把用户 U5 对 SC 表的 INSERT 权限收回

REVOKE INSERT
ON TABLE SC
FROM U5 CASCADE ;（缺省是 RESTRICT）

将用户 U5 的 INSERT 权限收回的时候必须级联（CASCADE）收回，因为 U5 将 SC 的 INSERT 权限授予 U6，U6 又将其授予 U7，CASCADE 系统只收回直接或间接从 U5 处获得的权限。

Grant 和 Revoke 向用户授予或收回对数据的操作权限

3、创建数据库模式的权限 ：DBA 在创建用户时实现

CREATE USER 语句格式

CREATE USER <username>
［WITH］[DBA | RESOURCE | CONNECT］

拥有 DBA 权限的用户是系统中的超级用户；只有系统的超级用户才有权创建新的数据库用户；如果没有指定创建的新用户的权限，默认该用户拥有 CONNECT 权限,只能登录数据库.。

拥有的权限	可否执行的操作			
	CREATE USER	CREATE SCHEMA	CREATE TABLE	登录数据库 执行数据查询和操纵
DBA	可以	可以	可以	可以
RESOURCE	不可以	不可以	可以	可以
CONNECT	不可以	不可以	不可以	可以，但必须拥有相应权限

权限与可执行的操作对照表

5、统计数据库存在何种特殊安全性

第五章：数据库的完整性

1、什么数据库完整性与数据库安全性的区别和联系

数据完整性：数据的正确性和相容性。

数据库的完整性和完全性是两个不同的概念，但他们有一定联系。

前者是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出，即所谓垃圾进垃圾出所造成的无效操作和错误结果。

后者是保护数据库，防止被恶意破坏和非法存储。

也就是说，安全性措施的防范对象是非法用户和非法操作，完整性措施的防范对象是不符合语义的数据。

2、数据模型中完整性约束条件的概念，RDBMS 的完整性控制机制应具有的功能

完整性约束条件：数据库中的数据所应满足的语义约束条件。

RDBMS 的完整性控制机制应具有 3 个方面的功能：

1、 定义功能，即提供定义完整性约束条件的机制。

2、 检查功能，即检查用户所发出的操作请求是否违背完整性约束条件。

3、 违约反映，如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的措施来保证数据的完整性。

3、用 SQL 实现完整性控制的方法

4、RDBMS 如何实现参照完整性

RDBMS 在现实参照完整性时需要考虑可能破坏参照完整性的各种情况，以及用户违约后的处理策略。

可能破坏参照完整性的 4 种情况：在参照关系中插入元祖、修改外码值时可能破坏参照完整性；在删除被参照表的元组、修改主码值时可能破坏参照完整性。

5、触发器的概念

触发器（Trigger）是用户定义在关系表上的一类由事件驱动的特殊过程，有时也叫 (Event-Condition-Action Rule)或 ECA 规则。

一旦定义，任何用户对表的增删改，均由服务器自动激活触发器，进行集中的完整性控制；可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力。

目的：实现由主键和外键所不能保证的参照完整性和数据一致性，定义更复杂的约束和业务规则，可采用触发器。

1、定义触发器

CREATE TRIGGER 语法格式

CREATE TRIGGER <触发器名>

{BEFORE | AFTER} <触发事件（INSERT、UPDATE、DELETE

）> ON <表名>

FOR EACH {ROW | STATEMENT}

［WHEN <触发条件>］

<触发动作体>

CREATE TRIGGER 必须是批处理中的第一条语句，并且只能应用于一个表。

触发器只能在当前的数据库中创建，但是可以引用当前数据库的外部对象。

定义触发器的语法说明:

1. 创建者：表的拥有者

2. 触发器名：表名和触发器必须在同一架构下

3. 表名：触发器的目标表

4. 触发事件：INSERT、DELETE、UPDATE

触发动作体与触发事件之间的关系：AFTER 、BEFORE 、INSTEAD OF，默认是 AFTER

5.触发器类型：依照触发动作的间隔尺寸，行级触发器（FOR EACH ROW）：触发动作体的执行次数根据目标表的行数决定；语句级触发器（FOR EACH STATEMENT）：触发动作体只执行一次。

6. 触发条件：触发器激活(触发事件发生)后，如触发条件为真，触发动作体才会执行，省略 WHEN 触发条件，触发动作体在触发器激活后立即执行。

7. 触发动作体：触发动作体可以是一个匿名 PL/SQL(Transact-SQL)过程块，也可以是对已创建存储过程的调用。如果触发动作体执行失败，激活触发器的事件会终止，目标表不发生变化。

8、存储过程（Stored Procedure）：是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它 。

例：定义一个 BEFORE 行级触发器，为教师表 Teacher 定义完整性规则“教授的工资不得低于 4000 元，如果低于 4000 元，自动改为 4000 元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
    BEFORE INSERT OR UPDATE ON Teacher
   /*触发事件是插入或更新操作*/
    FOR EACH ROW                              /*行级触发器*/
   AS BEGIN                          /*定义触发动作体，是 PL/SQL 过程块*/
        IF (new.Job='教授') AND (new.Sal < 4000) THEN
        new.Sal :=4000;
        END IF;
   END;
```

例：定义 AFTER 行级触发器，当教师表 Teacher 的工资发生变化后就自动在工资变化表 Sal_log 中增加一条相应记录

首先建立工资变化表 Sal_log

```
CREATE TABLE Sal_log
    (Eno      NUMERIC(4)   references teacher(eno)，
     Sal       NUMERIC(7，2)，
     Username   char(10)，
     Date     TIMESTAMP
    );
```

2、激活触发器

一个数据表上可能定义了多个触发器，同一个表上的多个触发器激活时遵循如下的执行顺序：（1） 执行该表上的 BEFORE 触发器。（2） 激活触发器的 SQL 语句。（3） 执行该表上的 AFTER 触发器。

例：当执行修改某个教师工资的 SQL 语句，激活上述定义的触发器 UPDATE Teacher SET Sal=800 WHERE Ename='陈平'; 执行顺序是：

执行触发器 Insert_Or_Update_Sal

执行 SQL 语句“UPDATE Teacher SET Sal=800 WHERE Ename='陈平';”

执行触发器 Insert_Sal；

执行触发器 Update_Sal

3、删除触发器 DROP TRIGGER <触发器名> ON <表名>;

触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除

例：删除教师表 Teacher 上的触发器 Insert_Sal

```
DROP TRIGGER Insert_Sal ON Teacher;
```

第六章：关系数据理论

1、为何要提出关系数据库规范化，即规范化理论是为了解决关系数据库中什么问题而引入的?

用来改造关系模式。通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、删除异常、更新异常和数据冗余问题。

2、函数依赖的基本概念，码(包括超码，主码，候选码，外码)的基本概念

1、函数依赖：设 R(U)是一个属性集 U 上的关系模式，X 和 Y 是 U 的子集， 若对于 R(U)的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等， 而在 Y 上的属性值不等， 则称 “X 函数确定 Y”或 “Y 函数依赖于 X”，记作 X→Y。

所谓函数依赖是指关系中属性或属性组的值可以决定其它属性的值，设 R(U)是属性集 U 上的关系模式，X、Y 是 U 的子集：

如果 X 和 Y 之间是 **1：1** 关系（一对一关系），如学校和校长之间就是 1:1 关系，则存在函数依赖 **X → Y 和 Y →X。**

如果 X 和 Y 之间是 **1：n** 关系（一对多关系），如年龄和姓名之间就是 1:n 关系，则存在函数依赖 **Y → X。**

如果 X 和 Y 之间是 **m：n** 关系（多对多关系），如学生和课程之间就是 m:n 关系，则 X 和 Y 之间**不存在函数依赖。**

在关系模式 R(U)中，对于 U 的子集 X 和 Y，如果 X→Y，但 Y □ 则称 X→Y 是非平凡的函数依赖；若 X→Y，但 Y 则称 X→Y 是平凡的函数依赖。

例：在关系 SC(Sno, Cno, Grade)中，非平凡函数依赖： (Sno, Cno) → Grade；

平凡函数依赖： (Sno, Cno) → Sno (Sno, Cno) → Cno

若 X→Y，则 X 称为这个函数依赖的决定属性组，也称为**决定因素**（Determinant）。

2、 完全函数依赖： 在 R(U)中，如果 X→Y，并且对于 X 的任何一个真子集 X’，都有 X’ Y，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{F} Y$。

3、部分函数依赖：若 X→Y，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作 $X \xrightarrow{P} Y$

例：中 (Sno,Cno)→Grade 是完全函数依赖，(Sno,Cno)→Sdept 是部分函数依赖

∵Sno →Sdept 成立，且 Sno 是（Sno，Cno）的真子集。

当存在部分依赖时，就会产生数据冗余。

4、传递函数依赖：在 R(U)中，如果 X→Y，(Y $\subseteq$ X) ,Y $\not\rightarrow$ X ，Y→Z，Z $\subseteq$ Y，则称 Z 对 X 传递函数依赖,记为：$X \xrightarrow{传递} Z$

注: 如果 Y→X， 即 X←→Y，则 Z 直接依赖于 X。

例: 在关系 Std(Sno, Sname，Sdept, Mname)中，有：

Sno → Sdept，Sdept → Mname

Mname 传递函数依赖于 Sno

5、侯选码：设 K 为 R<U,F>中的属性或属性组，若 $K \xrightarrow{F} U$（每个属性）， 则 K 称为 R 的侯选码（Candidate Key）

注 K 满足两个条件：

1．K 完全函数决定该关系的所有其它属性。

2.K 的任何真子集都不能完全函数决定 R 的所有其它属性，K 必须是最小的。

若候选码多于一个，则选定其中的一个做为**主码（Primary Key）**，通常称之为码。

主属性（Prime attribute）：包含在任何一个候选码中的属性。

非主属性或非码属性:不包含在任何码中的属性。

例：关系模式 S(Sno,Sdept,Sage)，单个属性 Sno 是码，

SC（Sno，Cno，Grade）中，（Sno，Cno）是码

由于码能唯一确定一个元组，所以关系的码函数决定该关系的所有属；一个关系中的所有属性都函数依赖于该关系的码。

例：关系模式 R（P，W，A）P：演奏者　　W：作品　　A：听众：一个演奏者可以演奏多个作品，某一作品可被多个演奏者演奏，听众可以欣赏不同演奏者的不同作品。

码为(P，W，A)，即 All-Key

6、外部码： 关系模式 R 中属性或属性组 X 并非 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的外部码（Foreign key）也称外码

如在 SC（Sno，Cno，Grade）中，Sno 不是码，

但 Sno 是关系模式 S（Sno，Sdept，Sage）的码，则 Sno 是关系模式 SC 的外部码

主码与外部码一起提供了表示关系之间联系的手段

3、1NF,2NF,3NF,BCNF 的定义及应用，能判断某一个关系模式处于第几范式，各级别范式存在的问题（插入，删除和更新异常）和解决方法、

各种范式之间存在联系：

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

1、1NF ： 如果一个关系模式 R 的所有属性都是不可分的基本数据项，则 R∈1NF

第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库；简而言之，第一范式就是无重复的列,关系数据库研究的关系都是规范化的关系。但是满足第一范式的关系模式并不一定是一个好的关系模式。

例：关系模式 S-L-C(Sno, Sdept, Sloc, Cno, Grade)　Sloc 为学生住处，假设每个系的学生住在同一个地方。

函数依赖包括：　(Sno, Cno) F　Grade

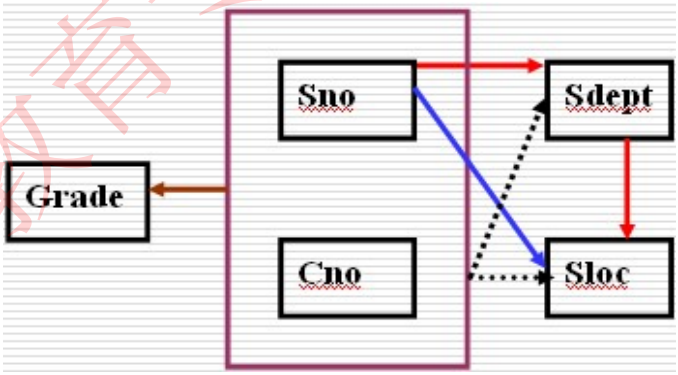
(Sno, Cno) P Sdept　∵ Sno → Sdept

(Sno, Cno) P Sloc　∵ Sdept 传递 Sloc

S-L-C 的码为(Sno, Cno)。

S-L-C 满足第一范式。

非主属性 Sdept 和 Sloc 部分函数依赖于码(Sno, Cno)。



插入异常：如未选课的学生不能插入。

S-L-C

Sno	Sdept	Sloc	Cno	Grade
S1	CS	BLD1	C1	95
S1	CS	BLD1	C2	90
S1	CS	BLD1	C4	88
S3	IS	BLD2	C2	70
S4	IS	BLD2	C3	78
:	:	:	:	:
S7	PHY	BLD2	NULL	NULL

X

如：学生S7已入学,且Sdept=PHY,Sloc=BLD2,但还未选课

由于主码Cno为空值,所以学生已有信息无法插入

删除异常：如放弃修一门课，只选修这-门课的学生被删除

S-L-C

Sno	Sdept	Sloc	Cno	Grade
S1	CS	BLD1	C1	95
S1	CS	BLD1	C2	90
S1	CS	BLD1	C4	88
S3	IS	BLD2	C2	70
S4	IS	BLD2	C3	78
:	:	:	:	:

"S4"的其他信息一并被删除

如：S4不再选择课程C3,需删除C3数据项

由于C3为主属性,删除C3必须删除整个元组,导致S4其他信息丢失

数据冗余度大　　修改复杂

S-L-C

Sno	Sdept	Sloc	Cno	Grade
S1	CS → IS	BLD1 → BLD2	C1	95
S1	CS → IS	BLD1 → BLD2	C2	90
S1	CS → IS	BLD1 → BLD2	C4	88
S3	IS	BLD2	C2	70
S4	IS	BLD2	C3	78
:	:	:	:	:

如：学生S1转系,由CS系转至IS系

Sdept和Sloc被重复存储,重复次数为S1选课数,修改时需无遗漏地考察每个相关元组,比较复杂

原因：存在对码的冗余依赖。Sdept、 Sloc 部分函数依赖于码。

解决方法：S-L-C 分解为两个关系模式，以消除这些部分函数依赖

SC（Sno， Cno， Grade）　　S-L（Sno， Sdept， Sloc）

2、2NF；若 R∈1NF，且每一个非主属性完全函数依赖于码，则 R∈2NF

简而言之，第二范式就是每一行被码唯一标识

例：S-L-C(Sno, Sdept, Sloc, Cno, Grade) ∈1NF

S-L-C(Sno, Sdept, Sloc, Cno, Grade) ∈2NF

SC（Sno， Cno， Grade） ∈ 2NF

S-L（Sno， Sdept， Sloc） ∈ 2NF

采用投影分解法将一个 1NF 的关系分解为多个 2NF 的关系，可以在一定程度上减轻原 1NF 关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。但将一个 1NF 关系分解为多个 2NF 的关系，并不能完全消除关系模式中的各种异常情况和数据冗余。

3、3NF：关系模式 R<U，F> 中若不存在这样的码 X、属性组 Y 及非主属性 Z（Z　　⊃,Y使得 X→Y，Y → X， Y→Z 成立，即**每个非主属性都不传递依赖于 R 的码**，则称 R<U，F> ∈ 3NF。若 R∈3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码。

简而言之，第三范式（3NF）要求一个数据库表中不能包含其它表中已包含的非码信息。

例：S-L(Sno, Sdept, Sloc) ∈ 2NF　　S-L(Sno, Sdept, Sloc) ∈ 3NF

S-D(Sno，Sdept) ∈ 3NF　　D-L(Sdept， Sloc)∈ 3NF

如果 R∈3NF，则 R 也是 2NF。

局部依赖和传递依赖是模式产生数据冗余和操作异常的两个重要原因。

由于 3NF 模式中不存在非主属性对候选码的局部依赖和传递依赖，因此一定程度上上解决原 2NF 关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题具有较好的性能。将一个 2NF 关系分解为多个 3NF 的关系后，仍然不能完全消除关系模式中的各种异常情况和数据冗余。

4、BC 范式（BCNF）：关系模式 R<U，F>∈1NF，若 X→Y 且 Y　　时 X必含有码，即每个属性都不传递依赖于 R 的码，则 R<U，F> ∈BCNF。

等价于：每一个决定因素都包含码，即消除任何属性对码的部分和传递函数依赖

若 R∈BCNF 所有非主属性对每一个码都是完全函数依赖；所有的主属性对每一个不包含它的码，也是完全函数依赖；没有任何属性完全函数依赖于非码属性。

R ∈BCNF ⇄（充分 / 不必要）R ∈3NF 。如果 R∈3NF，且 R 只有一个候选码

R ∈BCNF ⇄（充分 / 必要）R ∈3NF

例：关系模式 S（Sno，Sname，Sdept，Sage）假定 S 有两个码 Sno，Sname

S∈3NF　　S ∈ BCNF

例：系模式 SJP（S，J，P）　s 学生，J 课程， P 名次

函数依赖：（S，J）→P；(J，P）→S　（S，J）与（J，P）都可以作为候选码,属性相交

SJP∈3NF，(不存在非主属性对码的部分和传递依赖)

SJP∈BCNF,(每个决定因素都包含码)

第七章：数据库设计

1、数据库设计的基本步骤？每一个阶段的具体内容和方法？P205

1、需求分析

（1）内容：通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。

调查的重点是“数据”和“处理”，通过调查、收集与分析获得用户对数据库如下要求：

信息要求：用户需要从数据库中获得信息的内容与性质。由信息要求可以到处数据要求，即在数据库中需要存储哪些数据。

处理要求：用户要完成什么处理功能，对处理的相应时间有什么要求，处理方式是批处理还是联机处理。

安全性与完整性要求。

（2）方法

1、 调查组织机构情况。包括了解该组织部门组成情况、各部门的职责等，为分析信息流程准备。

2、 调查各部门的业务活动情况。包括了解各个部门输入和使用什么数据库，如何加工处理这些数据，输出什么信息，输出到什么部门，输出结果的格式是什么，这是调查的重点。

3、 在熟悉了业务活动的基础上，协助用户明确对信息系统的各种要求，包括信息要求、处理要求、安全性与完整性要求（重点）。

4、 确定新系统的边界。对前面调查的结果进行初步分析，确定哪些功能由计算机完成或将来准备让那个计算机完成，哪些活动由人工完成。

（3）数据字典

数据流程图：表达了数据和处理的关系。

数据字典：系统中各类数据描述的集合，是进行详细的数据手机和数据分析所获得的主要成果。

1、 数据项：

数据项描述=｛数据项名、数据项含义说明、别名、数据类型、长度、取值范围、取值含义、与其他数据项的逻辑关系、数据项之间的联系｝

2、　　数据结构：数据之间的组合关系。

数据结构描述=｛数据结构名、含义说明、组成：｛数据项或数据结构｝｝

3、　　数据流：数据结构在系统内传输的路径。

数据流描述=｛数据流名、说明、数据流来源、数据流去向、组成：｛数据结构｝、平均流量、高峰期流量｝

4、　　数据存储：数据结构停留或保存的地方，也是数据流的来源和去向之一。

数据存储描述=｛数据存储名、说明、编号、输入的数据流、输出的数据流、组成：｛数据结构｝、数据量、存取额度、存取方式｝

5、　　处理过程：具体处理逻辑一般用判定表或判定树来描述。

处理过程描述=｛处理过程名、说明、输入：｛数据流｝、输出：｛数据流｝、处理：｛需要说明｝｝

2、概念结结构设计

（1）方法

1、自顶向下：即首先定义全局概念结构的框架茫然和逐步细化。

2、自底向上。即首先定义各局部应用的概念结构，然后将他们集合起来，得到全局概念结构。

3、逐步扩张。首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构。

4、混合策略。即将自顶向下和自底向上结合，用自顶向下策略设计一个全局概念结构的框架，以它为顾家记成由自底向上策略中设计的各局部概念结构。

3、逻辑结构设计

4、物理结构设计

5、数据库实施

6、数据库运行和维护

2、什么是数据库的概念结构设计及其设计步骤，E-R 图的设计

E-R 图 P213

3、什么是数据库的逻辑结构设计？试述其设计步骤？E-R 图向关系模型的转换

逻辑结构设计 P224

4、数据库的再组织和重构造

第八章：数据库编程

1、嵌入式 SQL 与主语言之间的通信

2、游标的概念，用法，作用

第十章：数据库恢复技术

1、事务概念和四个特性，恢复技术能保证事务的哪几个特性

事务是用户所定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。事务具有四个特性：

1、原子性：事务是数据库的逻辑单位，事务中所包括的读项操作要么都做，要么都不做。

2、一致性：事务执行的结果必须是使数据库从某个一致性状态转变到另一个一致性状态。

3、隔离性：一个事务的执行不能被其他事务干扰，即一个事务内部的操作及所试用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

4、持续性：（永久性）事务一旦提交，它对数据库中主句的改变就应该是永久的。接下来的其他操作或昂不应对其执行结果产生任何影响。

为了保证事务的原子性、一致性与持续性。DBMS 必须对事务故障、系统故障和介质故障进行恢复；为了保证事务的隔离性和一致性，DBMS 需要对并发操作进行控制。

2、数据库中为什么要有恢复子系统？它的功能是什么？

因为计算机系统中的硬件的故障、软件的错误、操作员的失误及恶意的破坏都是不可避免的，这些故障轻则造成当前运行事务非正常中断，影响数据库中数据的正确性，重则会破坏数据库，导致数据库中的全部或部分数据丢失，因此必须要有恢复子系统。

恢复子系统的功能：把数据从错误状态恢复到某一已知的正确状态，也称为一致状态或完整状态。

3、日志文件的内容，作用，登记原则 P283

日志文件是用来记录事务对数据库所做的更新操作的文件。

目的：进行事务故障恢复；进行系统故障恢复；协助后背副本进行截至故障恢复。

内容：

事务标识（标明是哪个事务）；

操作的类型（插入、删除或修改）；

操作对象（记录内部标识）；

更新前数据的旧值（对插入操作而言，此项为空值）

更新后数据的新值（对删除操作而言，此项为空值）

作用：

1、事务故障恢复和系统故障恢复必须用到日志文件。

2.在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效地恢复数据库。

3、在静态转储方式中，也可以建立日志文件。当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件，把已完成的事务进行重做处理，对鼓掌发生时尚未完成的事务进行撤销处理。

登记原则：

1、登记的次序严格按并发事务执行的时间测序。

2、必须先写日志文件，后写数据库。

4、数据库恢复的基本技术

P279

5、故障的种类及其影响以及相应的恢复策略

1、事务内部故障的恢复：事务内部故障的恢复由 DBMS 自动完成，对用户而言是透明的。DBMS 执行的恢复步骤如下：

（1）反向扫描文件日志（即从后向前扫描日志文件），查找该事务的更新操作。

（2）对该事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库。

（3）继续反向扫描日志文件，进行同样的处理。

（4）如此继续下去，直至独到此事务的开始标记，该事务故障恢复就完成了。

2、系统故障的恢复：会造成数据库处于不一致的状态，主要是一方面，为完成事务对数据库所做的更新可能已写入数据库；另一方面，已提交事务对数据库做的更新可能尚留在缓冲区，未能及时写入数据库。因此恢复操作就是撤销（UNDO）故障发生时为完成的事务，重做（REDO）已完成的事务。恢复步骤如下：

（1）正向扫描日志文件，找出在故障发生之前已经提交的事务队列（REDO 队列）和为完成的事务队列（UNDO 队列）。

（2）对于撤销队列中的各个事务进行 UNDO 处理。进行 UNDO 处理的方法是：反向扫描日志文件，对每个 UNDO 事务的过呢更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库中。

（3）对重做队列中的各个事务进行 REDO 处理。进行 REDO 处理的方法是：正向扫描日志文件，对每个 REDO 事务重新执行日志文件中所登记的操作，激将日志记录中“更新后的值”写入数据库。

3、截至故障的恢复：恢复方法是重装数据库，然后重做已完成的事务，具体操作如下：

（1）DBA 装入最新的数据库后备副本（离故障发生时刻最近的转储副本）,使数据库回复到转储时的一致性状态。

（2）DBA 装入转储结束时的日志文件副本。

（3）DBA 启动系统恢复命令，有 DBMS 实现恢复功能，即重做已完成的事务。

6、具有检查点的恢复技术，检查点记录以及包括的内容 P287

检查点记录是一类新的日志记录。包括以下内容：

1、建立检查点时刻所有正在执行的事务的清单，如书图 T。

2、这些事务最近一个日志记录的地址，如书图 D。

第十一章：数据库并发控制技术

1、数据库中为什么要并发控制？能保证事务的哪些特性

数据库是共享资源，通常有许多事务同时运行。

当多个事务并发存取数据库中的数据时，会产生同时读取和/或修改同意数据的情况。若对并发操作不加以控制，可能会存取和存储不正确的数据，破坏数据库的一致性。所以，数据库管理系统必须提供并发控制机制。

并发控制可以保证事务的一致性和隔离性，保证数据库的一致性。

2、并发操作可能产生哪几类数据不一致

1、丢失修改：两个事务 1、2 同时读入同意数据并进行修改，2 所提交的结果破坏（覆盖）了 1 提交的结果，导致 1 所做的修改被丢失。

2、不可重复读：事务 1 读取某一数据后，事务 2 对其执行更新操作，使 1 无法再现前一次读取的结果。

3、读‘脏’数据：事务 1 修改某一数据，将其协会磁盘，事务 2 读取同意数据后，事务 1 由于某种原因被撤销，这是事务 1 已修改过的数据将恢复原值，事务 2 所读取的数据就与数据库中的数据不一致，则事务 2 所独到的就是‘脏’数据，即不正确的数据。

避免不一致性的方法是并发控制机制。最常用的并发控制技术是封锁技术。

3、活锁和死锁的概念 P296

4、封锁以及不同的封锁类型和相关的相容性矩阵

5、并发调度的可串行性,冲突可串行性化调度的概念，如何判断一个并发调度是正确的？ P299

6、两段锁协议的概念 P301

7、封锁的粒度，多粒度封锁协议 P302