数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。
每个数据库都有一个或多个不同的 API 用于创建，访问，管理，搜索和复制所保存的数据。
我们也可以将数据存储在文件中，但是在文件中读写数据速度相对较慢。
所以，现在我们使用关系型数据库管理系统（RDBMS）来存储和管理大数据量。所谓的关系型数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。

RDBMS 术语

在我们开始学习 MySQL 数据库前，让我们先了解下 RDBMS 的一些术语：

- **数据库:** 数据库是一些关联表的集合。
- **数据表:** 表是数据的矩阵。在一个数据库中的表看起来像一个简单的电子表格。
- **列:** 一列(数据元素) 包含了相同类型的数据, 例如邮政编码的数据。
- **行：**一行（元组，或记录）是一组相关的数据，例如一条用户订阅的数据。
- **冗余**：存储两倍数据，冗余降低了性能，但提高了数据的安全性。
- **主键**：主键是唯一的。一个数据表中只能包含一个主键。你可以使用主键来查询数据。
- **外键：**外键用于关联两个表。
- **复合键**：复合键（组合键）将多个列作为一个索引键，一般用于复合索引。
- **索引：**使用索引可快速访问数据库表中的特定信息。索引是对数据库表中一列或多列的值进行排序的一种结构。类似于书籍的目录。
- **参照完整性:** 参照的完整性要求关系中不允许引用不存在的实体。与实体完整性是关系模型必须满足的完整性约束条件，目的是保证数据的一致性。
- 表头(header): 每一列的名称;
- 列(col): 具有相同数据类型的数据的集合;
- 行(row): 每一行用来描述某条记录的具体信息;
- 值(value): 行的具体信息, 每个值必须与该列的数据类型相同;
- **键(key)**: 键的值在当前列中具有唯一性。、

虽然**查找表**和**视图表**在形式上都可以被视为“表”，但它们之间有几个重要的区别：

特性	**查找表（物理表）**	**视图表（虚拟表）**
数据存储	存储实际数据	不存储数据，只包含查询逻辑 基于查询结果动态生成
性能	查询时直接操作存储的数据	查询时动态执行 SQL 语句，可能会有额外开销

特性	查找表（物理表）	视图表（虚拟表）
更新操作	数据可更新，直接修改表中的数据	对视图的更新可能会有限制，特别是当视图涉及多个表时
复杂查询封装	不支持复杂的查询封装	支持复杂查询封装，可以简化访问
查询方式	直接查询表	查询视图会间接查询底层表。作为表使用

您可以使用 MySQL 二进制方式进入到 mysql 命令提示符下来连接 MySQL 数据库，格式如下：

```
mysql -u your_username -p
```

数据库技术中的四个名词

- DB：数据库（Database),DB 是统一管理的相关数据的集合。
- DBMS：数据库管理系统（Database Management System)，DBMS 是位于用户与操作系统之间的一层数据管理软件，为用户或应用程序提供访问 DB 的方法，包括 DB 的建立、查询、更新及各种数据控制。DBMS 总是基于某种数据模型，可以分为层次型、网状型、关系型、面向对象型 DBMS。
- DBS：数据库系统（Database System),DBS 是实现有组织地、动态地存储大量关联数据，方便多用户访问的计算机软件、硬件和数据资源组成的系统，即采用了数据库技术的计算机系统。
- DBA：数据库管理员。是 DBS 中的一类人员，或者公司的一个岗位名称。是从事管理和维护数据库管理系统(DBMS)的相关工作人员的统称，他属于运维工程师的一个分支，主要负责业务数据库从设计、测试、部署交付到运行监控的全生命周期管理。
- 数据库技术：是一门研究数据库结构、存储、管理和使用的软件学科。

DBMS 的主要功能:

（1）数据库的定义功能 DDL 打开 SSMS（客户端工具：图形化/SQL 语句）
（2）数据库的操纵功能 DML（对数据进行增删改：Insert、delete、Update（更新））
（3）数据库的保护功能(实现保护的四个子系统）安全性、完整性、并发控制、备份恢复访问控制
（4）数据库的存储管理 （物理存储）DBA 工作
（5）数据库的维护功能 （数据字典：DD 元数据：数据的数据 Master 数据库中的 Sys 表，系统配置）
DBS
3.1 DBS 的组成
DBS 是采用了数据库技术的计算机系统。DBS 是一个实际可运行的， 按照数据库方法存储、维护和向应用系统提供数据支持的计算机系统。
DBS 由四部分组成：数据库(DB)、硬件、软件、数据库用户
数据库用户可分为：DBA，专业用户，专业程序员，最终用户。
(主要理解 DB 和 DBA)
DBA 是控制数据整体结构的人，负责 DBS 的正常运行。DBA 可以是一个人，在大型系统中也可以是由几个人组成的小组。

DBA 的主要职责有五点：
（1）概念模式定义
（2）内模式定义
（3）根据要求修改数据库的概念模式和内模式
（4）对数据库访问的授权
（5）完整性约束的说明

4.1 数据库应用软件结构的发展

从单用户、Client/Server 模式（局域网）、Browser/Server 模式（互联网）、C/S 和 B/S 混合结构

4.2 C/S

C/S 是将需要处理的业务合理地分配到客户端和服务器端，这样可以大大降低通信成本，但是升级维护相对困难。
了解客户端和服务器端的分工。

4.3 B/S

B/S 结构是随着互联网的发展，web 出现后兴起的一种网络结构模式。这种模式统一了客户端，让核心的业务处理在服务端完成。B/S 维护和升级方式更简单。由于客户端是浏览器，基本不需要维护，只需要维护升级服务器端。但安全性上不如 C/S 可控。

数据库的体系结构

5.1 三级模式结构

三级模式、二级映像的目的：为了提高数据的独立性。
数据库的体系结构分为三级：内部级、概念级和外部级 (数据抽象的三个级别)
(1)外部级：单个用户所能看到的数据特性；
单个用户使用的数据视图的描述称“外模式”，又称“子模式”。（最接近用户）
(2)概念级：涉及到所有用户的数据定义，是全局的数据视图；
全局数据视图的描述称“概念模式”，又称“模式”。
(3)内部级：最接近于物理存储，涉及到实际数据存储的结构；
物理存储视图的描述称为“内模式”。
(进一步理解三级模式的具体含义，并掌握两个映象。)
数据按外模式的描述提供给用户（应用程序），按内模式的描述存储在磁盘中， 而概念模式提供了连接这两级的相对稳定的中间观点，并使得任何一级的改变都不受另一级的牵制。
(1)概念模式 （模式）：具体含义；（模式 DDL）
(2)外模式：具体含义；（外模式 DDL）
外模式又称为“用户模式”或“子模式”，通常是概念模式的逻辑子集。
(3)内模式：具体含义；（内模式 DDL）
(4)模式/内模式映象：用于定义概念模式和内模式之间的对应性。一般在内模式中描述。保证数据的物理独立性。
(5)外模式/模式映象：用于定义外模式和概念模式间的对应性。在外模式中描述。
保证数据的逻辑独立性
(1) 物理数据独立性：修改内模式时尽量不影响概念模式及外模式，从而达到不影响既有应

用程序的目的。
(2) 逻辑数据独立性：修改概念模式时尽量不影响外模式和应用程序，从而达到不影响既有应用程序的目的。
考核要求：达到“识记”层次
知识点：主要是一些基本概念

三个世界：现实世界、信息世界、数据世界
对应模型：信息世界：概念模型（ER 图或者 UML 表达概念模型）
数据世界：先设计出逻辑模型再转化为物理模型（我们只讨论关系数据模型，其他模型只需大致了解）
对应概念：　对象-→实体 -→元组（记录）
一类对象—>实体集—>关系（表）
特性—>属性 ☒属性
域 ☒ 域
码 ☒ 码（候选码、主码）
对象间的联系—》实体集间的联系—》属性、关系（表），外码
采用 ER 方法进行数据库概念设计分成三步进行：
实体、属性、实体集、联系、域、码
联系的类型（阶）：1：1、1：N（或者 N：1）、M：N

(1)设计局部 ER 模式
确定局部结构范围；实体定义；联系定义；属性分配
(2)设计全局 ER 模式
确定公共实体类型；局部 ER 模型的合并；消除冲突
(3)全局 ER 模式进行优化
实体类型的合并；冗余属性的消除；冗余联系的消除
(1) 二维表格 在关系模型中，一张二维表格对应一个关系。
(2) 元组 表中的一行（即一个记录），表示一个实体；关系是由元组组成的。
(3) 域 是一组具有相同数据类型的值的集合。（属性的取值范围）
(4) 笛卡尔积 域上的一种集合运算。
(5) 关系：是一个元数为 K(K>=1)的元组的集合。 一张二维表格对应一个关系。表中的一行称为关系的一个元组；表中的一列称为关系的一个属性。
在关系模型中，对关系作了下列规范性的限制：（关系的六条基本性质）

关系中每一个属性值都是不可分解的；
关系中不允许出现相同的元组（没有重复元组)；
不考虑元组间的顺序，即没有行序；
在理论上，属性间的顺序（即列序）也是不存在的；
列是同质的，即每一列中的分量是同一类型的数据，来自同一个域；
不同的列可出自同一个域，不同的属性要给予不同的属性名。

(6) 超码（Super Key）：在关系中能唯一标识元组的属性集称为关系模式的超码；
(7) 候选码(Candidate Key)：*不含有多余属性的超码称为候选码*；关系中能唯一地标识一个元组而不包含多余属性的属性（或属性组），叫码或者候选码，一个关系可以有多个候选

码。

(8) 主码（Primary Key）：用户选作元组标识的一个候选码。 如果一个关系有多个候选码，则选择其中一个作为主码。

在以上概念中，主码一定可作候选码，候选码一定可作超码；反之，则不成立。

比如，在学生表中，如果有“学号”、“姓名”、“出生年月”等字段， 其中学号是唯一的，那么（学号）属于超码，（学号，姓名）的组合也是超码。 同时，（学号）是候选码而（学号，姓名）由于含有多余属性，所以不是候选码。 在这三个概念中，主码的概念最为重要，它是用户选作元组标识的一个关键字。 如果一个关系中有两个或两个以上候选码用户就选其中之一作为主码。

(1) 实体完整性规则 要求关系中元组在组成候选码的属性上不能有空值。如果出现空值，那么候选码就起不了唯一标识元组的作用。（对关系的码的约束） 主码 Primary key 也叫主键

(2) 参照完整性规则 要求外码值必须是（另）一个关系的主码的有效值，或者是空值。（对关系外码的约束）

外码：（Foreign Key 也叫外键）将一个关系的主码（比如 学生关系 S 中的 S#） 放到另一个关系（比如选课关系 SC）中，此时称 Sno 是关系 SC 的外码。

注意事项：
外码和相应的主码可以不同名，只要定义在相同值域上即可；
两个关系可以是同一个关系模式，表示了属性之间的联系。
外码值是否允许空，应视具体情况而定

(3) 用户定义的完整性规则：这是针对某一具体数据的约束条件，由应用环境决定， 例如，学生的年龄限制为 15~30 周岁。 用户定义的完整性规则反映某一具体应用涉及的数据必须满足的语义要求。 系统提供定义和检验这类完整性的机制。

3 数据模型和关系数据模型的基本概念

3.1 数据模型三要素

考核要求：达到“识记”层次

知识点：数据结构、数据操作、完整性约束

结合下面的关系数据模型进行理解

3.2 二维表格的基本术语

考核要求：达到“识记”层次

知识点：主要是一些基本概念

(1) 二维表格 在关系模型中，一张二维表格对应一个关系。

(2) 元组 表中的一行（即一个记录），表示一个实体；关系是由元组组成的。

(3) 域 是一组具有相同数据类型的值的集合。（属性的取值范围）

(4) 笛卡尔积 域上的一种集合运算。

(5) 关系：是一个元数为 K(K>=1)的元组的集合。 一张二维表格对应一个关系。表中的一行称为关系的一个元组；表中的一列称为关系的一个属性。

在关系模型中，对关系作了下列规范性的限制：（关系的六条基本性质）

关系中每一个属性值都是不可分解的；
关系中不允许出现相同的元组（没有重复元组)；
不考虑元组间的顺序，即没有行序；
在理论上，属性间的顺序（即列序）也是不存在的；
列是同质的，即每一列中的分量是同一类型的数据，来自同一个域；
不同的列可出自同一个域，不同的属性要给予不同的属性名。

(6) 超码（Super Key）：在关系中能唯一标识元组的属性集称为关系模式的超码；

(7) 候选码(Candidate Key)：*不含有多余属性的超码称为候选码*；关系中能唯一地标识一个元组而不包含多余属性的属性（或属性组），叫码或者候选码，一个关系可以有多个候选码。

(8) 主码（Primary Key）：用户选作元组标识的一个候选码。 如果一个关系有多个候选码，则选择其中一个作为主码。

在以上概念中，主码一定可作候选码，候选码一定可作超码；反之，则不成立。

比如，在学生表中，如果有“学号”、“姓名”、“出生年月”等字段， 其中学号是唯一的，那么（学号）属于超码，（学号，姓名）的组合也是超码。 同时，（学号）是候选码，而（学号，姓名）由于含有多余属性，所以不是候选码。 在这三个概念中，主码的概念最为重要，它是用户选作元组标识的一个关键字。 如果一个关系中有两个或两个以上候选码，用户就选其中之一作为主码。

3.3 关系模型的形式定义

考核要求：达到“识记”层次

知识点：三个组成部分的了解

关系模型有三个组成部分：数据结构、数据操作和完整性约束条件
关系模型的的数据结构是关系；
关系模型提供一组完备的高级关系运算（关系代数+关系演算），支持数据库的各种操作；
关系模型包括三类完整性规则。

3.4 关系模型的三类完整性规则

考核要求：达到“领会”层次，第四章后达到熟悉掌握，是课程重点之一。

知识点：三类完整性规则的理解

(1) 实体完整性规则 要求关系中元组在组成候选码的属性上不能有空值。如果出现空值，那么候选码就起不了唯一标识元组的作用。（对关系的码的约束） 主码 Primary key 也叫主键
(2) 参照完整性规则 要求外码值必须是（另）一个关系的主码的有效值，或者是空值。（对关系外码的约束）

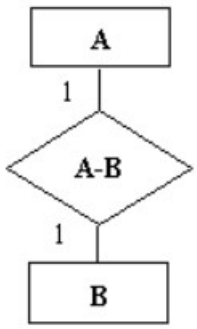
外码：（Foreign Key 也叫外键）将一个关系的主码（比如 学生关系 S 中的 S#） 放到另一个关系（比如选课关系 SC）中，此时称 Sno 是关系 SC 的外码。

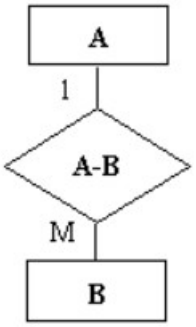
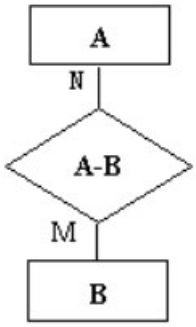
注意事项：
外码和相应的主码可以不同名，只要定义在相同值域上即可；
两个关系可以是同一个关系模式，表示了属性之间的联系。
外码值是否允许空，应视具体情况而定

假设数据库有如下关系：
学生关系 S (S#、SNAME、AGE、SEX)
课程关系 C（C#、CNAME、TEACHER）
学习关系 SC（S#、C#、GRADE）
那么
(1)S#是关系 S 的主码，因此在关系 S 中不能为空；（实体完整性规则）
(2)C#是关系 C 的主码，因此在关系 C 中不能为空；（实体完整性规则）
关系 SC 中：
（S#，C#）的组合为主码，因此 S#、C#各自都不能为空；（实体完整性规则）
S#是来自 S 的外码，因此它必须和关系 S 中某个元组的 S#相同。（参照完整性规则）
C#是来自 C 的外码，因此它必须和关系 C 中某个元组的 C#相同。（参照完整性规则）

(3) 用户定义的完整性规则：这是针对某一具体数据的约束条件，由应用环境决定， 例如，学生的年龄限制为 15~30 周岁。 用户定义的完整性规则反映某一具体应用涉及的数据必须满足的语义要求。 系统提供定义和检验这类完整性的机制。

4 概念模型向关系数据模型的转化（即 ER 图转化为表结构）
考核要求：达到“熟练掌握”层次
知识点：转换的实际操作
ER 模型转换关系数据库的 一般规则：
（1） 将每一个实体类型转换成一个关系模式，实体的属性为关系模式的属性。
（2） 对于二元联系，按各种情况处理，如下面所示。

二元关系	ER 图	转换成的关系	联系的处理	主码	外码
1：1		（2 个关系） 模式 A 模式 B	（有两种） 处理方式（1） （1）把模式 B 的主码，联系的属性加入模式 A 处理方式（2）： （2）把模式 A	（略）	（依据联系的处理方式） 方式（1）：模式 B 的主码为模式 A 外码 方式（2）：表 A 的主码为表 B 的外

			的主码，联系的属性加入模式B		码
1：M		（2个关系） 模式A 模式B	把模式B的主码，联系的属性加入模式A	（略）	模式A对应模式B的主码为模式A的外码
M：N		（3个关系） 模式A 模式B 模式A-B	联系类型转换成关系模式A-B; 模式A-B的属性 (a)联系的属性 (b)两端实体类型的主码	两端实体类型的主码一起构成模式A-B主码	两端实体类型的主码分别为模式A-B的外码

SQL数据库的体系结构也是三级结构，但术语与传统关系模型术语不同，在SQL中，关系模式称为"基本表"，存储模式称为"存储文件"，子模式称为"视图"，元组称"行"，属性称"列"。

SQL数据库体系的结构要点如下：

(1)一个SQL数据库是表的汇集。

(2)一个SQL表由行集构成，行是列的序列，每列对应一个数据项。

(3)表或者是基本表，或者是视图。基本表是实际存储在数据库中的表，视图由是由若干基本表或其他视图构成的表的定义。

(4)一个基本表可以跨一个或多个存储文件，一个存储文件也可存放一个或多个基本表。存储文件与物理文件对应。

(5)用户可以用SQL语句对表进行操作，包括视图和基本表。

(6)SQL的用户可以是应用程序，也可以是终端用户。

常用SQL语句分为三类：

(1)数据定义：DDL：CREATE、ALTER、DROP，定义SQL模式，可以是基本表、视图和索引。

(2)数据操纵：DML：SELECT、INSERT、UPDATE、DELETE，包括数据查询和数据更新(增、删、改)。

(3)数据控制：DCL：GRANT、REVOKE，包括对基本表和视图的授权、完整性规则的描述，

事务控制等。（第四章安全性讲）
2.1 SQL 提供的基本数据类型
数值型：包括 int、smallint、decimal(p，d)、numeric(p，d)
字符串型：char(n) 、nchar(n)、varchar(n)、nvarchar(n)前 2 者是定长，后 2 者为变长串
时间型：datetime。 。

2.2 基本表的创建、修改和撤消
(1)基本表的创建：(建立表结构)
CREATE TABLE 基本表名
(列名，类型，
……
完整性约束…)
完整性约束包括主键子句(PRIMARY KEY)、检查子句(CHECK)和外键子句(Foreign KEY).
(2)基本表结构的修改
增加新列：
ALTER TABLE 基本表名 ADD 列名 类型
删除原有的列：
ALTER TABLE 基本表名 DROP COLUMN 列名
修改数据列，只能修改数据列的数据类型，注意其中的限制：
ALTER TABLE 基本表名 ALTER COLUMN 列名 新的数据类型
(3)基本表的撤消
DROP TABLE 基本表名

2.3 索引的创建和撤消
(1)索引的创建：
CREATE [CLUSTERED][UNIQUE] INDEX 索引名 ON 表名(列名[ASC|DESC])
(2)索引的撤消：
DROP INDEX 表名.索引名
索引的作用，索引的分类（聚餐索引和非聚餐索引）
3.1 SELECT 语句的基本句法
考核要求：达到“综合应用”层次
知识点：SELECT-FROM-WHERE 句型的应用

(1) SELECT-FROM-WHERE 句型
SELECT 列名表(逗号隔开)
FROM 基本表或视图序列
WHERE 条件表达式
Notice:掌握条件表达式中各种运算符的应用.

算术比较运算符=，>,<,<>或!=,<=,>=；
逻辑运算符 AND、OR、NOT；
集合成员资格运算符：IN，NOT IN；like， not like；between and ；
谓词:EXISTS(存在量词)，NOT EXISTS；
聚合函数：（在下面介绍）

(2) SELECT 句型使用实例
本节内容需要多看例题，多作习题进行掌握。
Notice:嵌套的 SELECT 语句的用法 ，体会结构化的含义。
(3)聚合函数
注意各个函数的含义：

COUNT（*）	计算元组的个数
COUNT（列名）	求一列中值的计算个数
COUNT(DISTINCT 列名)	求一列中值的种类数
SUM（列名）	求一列中值的总和
AVG（列名）	求一列中值的平均值
MAX（列名）	求一列中值的最大值
MIN（列名）	求一列中值的最小值

如果语句没有 Group By 则返回单个值；如果语句带 Group By 则每个小组返回一个值。
SELECT 列名表(逗号隔开)
FROM 基本表或视图序列
[WHERE 条件表达式] (行条件子句)
[GROUP BY 列名序列] (分组子句)
[HAVING 组条件表达式] (组条件子句)
[ORDER BY 列名[ASC|DESC]..] (排序子句)
(1)SELECT 子句中的规定

- 如果要求输出表格中不允许出现重复元组，则在 SELECT 后加一 DISTINCT
- SELECT 子句中允许出现 +,-,*,/,以及列名、常数、函数的算术表达式

(2) 条件表达式的算术比较操作

- WHERE 子句中可以用 BETWEEN...AND...来限定一个值的范围

(3)列和基本表的改名操作

- 同一个基本表在 SELECT 语句中多次引用时可用 AS 来增加别名

(4)字符串的匹配操作

- WHERE 子句中字符串匹配用 LIKE 和两个通配符，%和下划线_.

(5)集合的并、交、差操作

- 查询结果的结构完全一致时，可将两个查询进行并(UNION)、交（InterSect）、差（Except）

(6)空值的比较操作

- 查询空值操作不是用='null',而是用 IS NULL 来测试。

(7)集合的比较操作

- 集合成员资格比较用 IN/NOT IN

- 集合成员算术比较用元组 θ ANY/ALL (θ 是算术比较运算符）（少用）

(8) 导出表的使用

- 使用 INTO 关键字，可以给子查询结果起个表名，保存相关查询数据。

(9) 基本表的连接操作

- 等值连接操作是用 [INNER] JOIN 来实现的。
- 外连接：左外连接 LEFT [OUTER] JOIN 、RIGHT [OUTER] JOIN、FULL [OUTER] JOIN

自身连接：通过设置别名的方法，一个 SELECT 语句中可以对一个表进行多次扫描，实现连接。

数据插入方式有两种：

(1) 元组值的插入

INSERT INTO 基本表名(列名表)

values(元组值)

(2) 查询结果的插入

INSERT INTO 基本表名(列名表)

SELECT 查询语句

WHERE 子句中使用下列关键字

除了比较符，可以在 WHERE 子句中使用下列

关键字

关键字

含义

IN

判断值是否在所给的范围内

NOT IN

判断值是否不在所给的范围内

LIKE

判断值是否与一个给定的值相似

NOT LIKE

判断值是否与一个给定的值不相似

IS NULL

判断值是否为空

IS NOT NULL

判断值是否不为空

AND

判断是否满足 AND 两边所给的条件

OR

判断是否满足 OR 两边所给的条件中的一个

BETWEEN…AND

判断值是否在所给的值之间

NOT BETWEEN…AND

判断值是否不在所给的值之间

使用 **LIKE 运算符**可以完成对字符串的模糊匹配。

v

基本语法：

SELECT select_list

FROM table_sourse
WHERE EXPRESSION LIKE ‘STRING’
v
其中，EXPRESSION 为用作选择条件的列或表达式
v
指定的字符串中可以包含通配符，包括：
%：代表任意多个字符。例：%jing
_（下划线）：代表单个字符。例：_ouse
[ ]：代表指定范围内的单个字符。例：[mh]ouse
[^]：代表不再指定范围内的单个字符。例：[^mh]ouse

WHERE 子句中，也可以使用逻辑运算符来
连接多个条件。
v
主要有逻辑运算符：
AND
OR
NOT
4.2 SQL 数据删除
考核要求：达到“综合应用”层次

DELETE FROM 基本表名 [WHERE 条件表达式]
（语义：从基本表中删除满足条件表达式的元组）
注意带子查询的数据删除
示例：求有两门以上不及格课程同学的学号及其平均成绩
Select S#, Avg(Score) From SC
Where Score < 60
Group by S# Having Count(*)>2;
Ø 该 SQL 语句求出的是“该同学那几门不及格课程的平均成绩”，而不是 “该同学所有课程的平均成绩” 。因此正确写法为：
Select S#, Avg(Score) From SC
Where S# in
(Select S# From SC
Where Score < 60
Group by S# Having Count(*)>2)
Group by S#
交叉连接
也叫**非限制连接**，将两个表不加任何约束的组合起来，
即两个表的**广义笛卡尔积**。
v
交叉连接后得到的结果集的行数是两个被连接表的行数
的乘积。
v
语法：

SELECT select_list
FROM table1 **CROSS JOIN** table2
或
SELECT select_list
FROM table1 , table2
SQL 高级语法：
Select
列名
[[, 列名**] ...]**
From 表名 **1 [NATURAL]**
[INNER | { LEFT | RIGHT | FULL} [OUTER]] JOIN 表名 **2**
{ ON
连接条件**}**
[***Where*** 检索条件 **] ... ;**
连接中使用 natural
q 出现在结果关系中的两个连接关系的元组在公共属性上取值相等，且 公共属性只出现一次
Ø 连接中使用 on <连接条件>
q 出现在结果关系中的两个连接关系的元组取值满足连接条件，且公共属性出现两次
内连接(Inner Join)
v
指定了 INNER 关键字的连接是内连接，内连接按照 ON 所指定的连接条件合并两个表，返回满足条件的行。
v
内连接是系统默认的，可以省略 INNER 关键字。使用内连接后仍可使用 WHERE 子句指定条件。
v
如：<表名 1>.<列名 1> <> <表名 2>.<列名 2> 2024/6/6
102
等值连接(EquiJoin)
v
使用“=”关系将表连接起来的查询，其查询结果中列出被连接表中的所有列，包括其中的重复列。
v
等值连接
连接运算符为 = 的连接操作
v [<表名 1>.]<列名 1> = [<表名 2>.]<列名 2>
任何子句中引用表 1 和表 2 中同名属性时，都必须加表名前缀。引用唯一属性名时可以加也可以省略表名前缀。
v
表的连接条件经常采用“**主码=外部码**”的形式。2024/6/6
103
v

语法：
SELECT select_list
FROM table1 **[INNER] JOIN** table2
ON table1.column1=table2.column2
和
SELECT select_list
FROM table1 ，table2
WHERE table1.column1=table2.column2
示例：按“001”号课成绩由高到低顺序显示所有学生的姓名(二表连接)
Select
Sname From Student, SC
Where Student.S# = SC.S# and SC.C# = ‘001’
Order By Score DESC;
Ø 多表连接时，如两个表的属性名相同，则需采用表名. 属性名方式来限定该 属性是属于哪一个表
示例：按‘数据库’课成绩由高到低顺序显示所有同学姓名(三表连接)
Select
Sname From Student, SC, Course
Where
Student.S# = SC.S# and SC.C# = Course.C# and Cname = ‘数据库’
Order By Score DESC;
自然连接
等值连接的一种特殊情况，自动判断相同名称的列，
而后形成匹配，把目标列中重复的属性列去掉。
<表名 1>.<列名 1> = <表名 2>.<列名 2>
v
有些 DBMS 产品，如 SQL Server，不支持
NATURAL JOIN 连接符，故不能直接实现自然连接

非等值连接 (Non-EquiJoin)
v
非等值连接
连接运算符不为 = 的连接操作
v
形式：
[<表名 1>.]<列名 1> <比较运算符> [<表名 2>.]<列名 2>
比较运算符：>、<、>=、<=、!=（<>）
[<表名 1>.]<列名 1> BETWEEN [<表名 2>.]<列名 2>
AND [<表名 2>.]<列名 3>
求既学过“001”号课又学过 “002”号课的所有学生的学号
Select
S1.S#
From SC S1, SC

S2 Where
S1.S# = S2.S# and
S1.C#='001'
and
S2.C#='002 ;
示例：求“001”号课成绩比“002”号课成绩高的所有学生的学号
Select S1.S# From SC S1, SC S2 Where
S1.S# = S2.S# and S1.C#='001'
and
S2.C#='002' and S1.Score > S2.Score;
S
外连接
v
指定了 OUTER 关键字的为外连接，外连接的结果表不但包含满足连接条件的行，还包括相应表中的所有行。
v
外连接中，可以只限制一个表，而对另一个表不加限制（即所有的行都出现在结果集中）。
v
外连接分为左外连接、右外连接和全外连接。

自身连接(Self-Join)
v
连接操作是在同一张表内进行**自身连接**，即将同一个表的不同行连接起来。
v
可看作一张表的两个副本之间进行的连接。
v
必须为表指定两个别名，使之在逻辑上成为两张表。
v
由于所有属性名都是同名属性，因此必须使用别名前缀
v
例 36.查询 Employee 表中的同姓的职工。
SELECT e1.Ename,e2.Ename
FROM **Employee e1** JOIN **Employee e2**
ON left(e1.Ename,1)=left(e2.Ename,1)
WHERE e1.Eno<e2.Eno

SQL 并运算
Ø**UNION 中的每一个查询所涉及的列必须具有相同的列数，相同的数据类型，并以相同的顺序出现。**

Ø **最后结果集中的列名来自第一个 SELECT 语句。**
Ø 若 UNION 中包含 ORDER BY 子句，则将对最后的结果集排序。
示例：求学过 002 号课的同学或学过 003 号课的同学学号

```
Select
S#
From
SC
Where
C# = '002'
UNION
Select
S#
From
SC
Where
C# = '003';
```

Ø 上述语句也可采用如下不用 UNION 的方式来进行

```
Select
S#
From
SC
Where
C# = '002'
OR
C# = '003';
```

Ø 内层查询独立进行，没有涉及任何外层查询相关信息的子查询

v
子查询可以多次嵌套。

v
分为两种：子查询返回单个值和子查询返回一个值列表。

v
返回单个值，该值被外部查询的比较操作使用，该值可以是子查询中使用集合函数得到的值。

v
返回一个值列表，该列表被外部查询的 IN、NOT IN、ANY 或 ALL 比较操作使用。

IN 表示属于，NOT IN 表示不属于。

ANY 和 ALL 用于一个值与另一个组值的比较

v
WHERE 子句中不得直接出现聚合函数。
错误用法：

```
SELECT Ename,Age
```

FROM Employee

WHERE Age >**AVG(Age);**

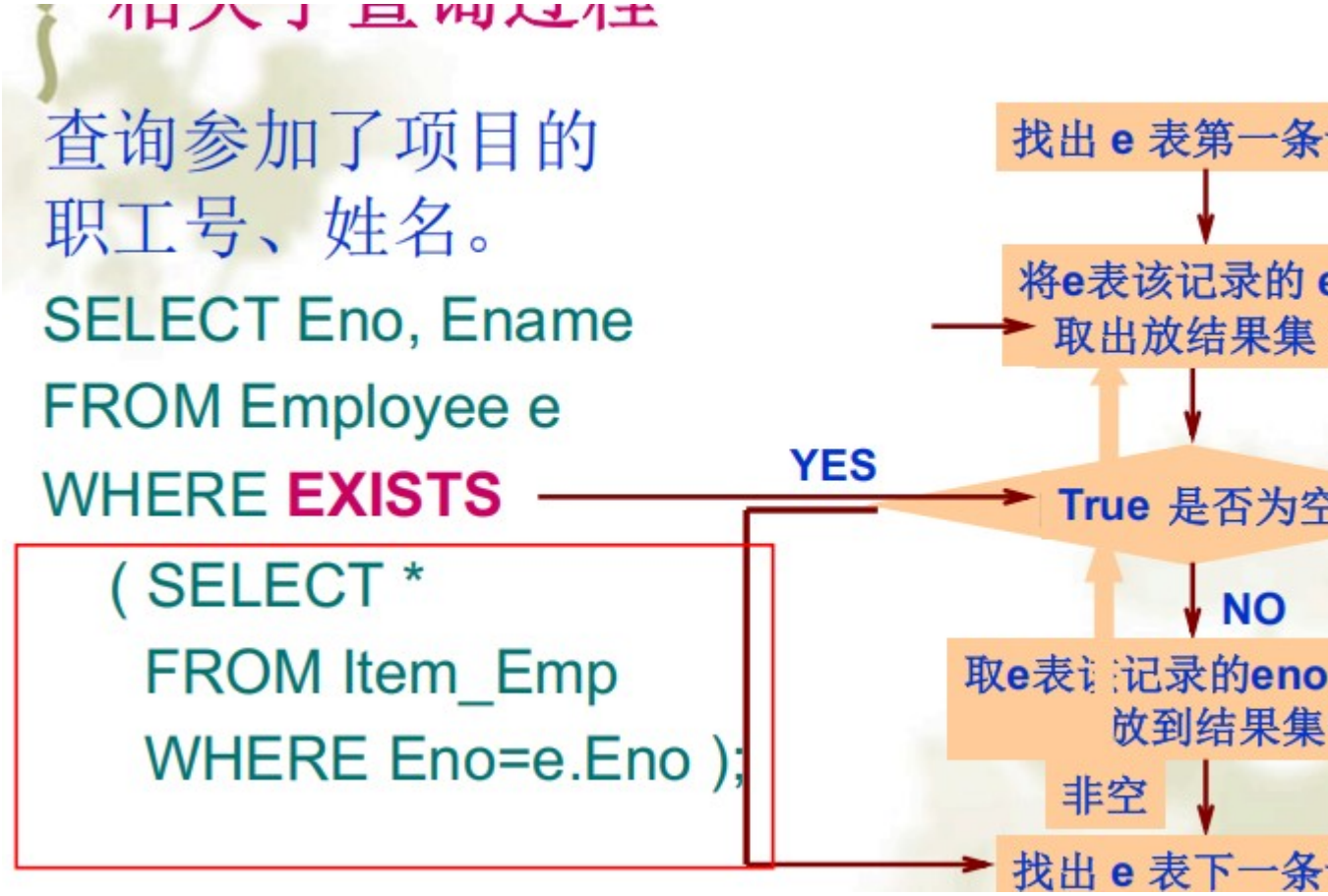
相关子查询

v

执行依赖于外部查询，多数情况下是在子查询的 WHERE 子句中引用了外部查询的表。

v

可以使用 EXISTS 关键字来判断查询结果中是否存在数据，EXISTS 在一个子查询至少返回一行时成立。



4.3 数据修改

考核要求：达到“综合应用”层次

UPDATE 基本表名
SET 列名=值表达式,[列名=值表达式...]
[WHERE 条件表达式]
(语义：修改基本表中满足条件表达式的那些元组中的列值，需修改的列值在 SET 子句中指出)
注意带子查询的数据更新

(1)视图的创建：
CREATE VIEW 视图名(列名表) AS SELECT 查询语句
(2)视图的撤消：
DROP VIEW 视图名
语句格式

CREATE VIEW <视图名> [<列名清单>]
AS <子查询>
[WITH CHECK OPTION]
v
说明
<视图名>给出所定义的视图的名称。
<列名清单>，若有则此清单给出了此视图的全部属性的属性名；否则，
此视图的所有属性名即为子查询中 SELECT 语句中的全部目标列。
<子查询>为任一合法 SELECT 语句（但一般不含有 ORDER BY，
UNION 等语法成分）。
有[WITH CHECK OPTION]时，则今后对此视图进行 INSERT、
UPDATE 和 DELETE 操作时，系统自动检查是否符合原定义视图子查
询中的<条件表达式>。
4.4.2 视图的更新
对视图的查询(SELECT)操作，和基本表一样，但是视图的更新操作受到下列三条规则的限制：

- 如果视图是从多个基本表使用连接操作导出的，则不允许更新。
- 如果导出的视图使用了分组和聚合操作，也不允许更新。

如果视图是从单个基本表使用选择、投影操作导出的，并且包括了基本表的主键和没有缺省值的不可为空的列， 那么这样的视图称为“行列子集视图”，则可以执行操作。
有些查询使用一个 Select 语句完成，可能做不到（也可能是逻辑上太绕），可以使用视图存放中间查询的数据，在后续 Select 语句中跟视图进行连接查询实现。
数据库完整性是通过 DBMS 的完整性子系统实现的，它有两个功能：

(1)监督事务的执行，并测试是否违反完整性规则。
(2)如有违反，则采取恰当的操作，如拒绝、报告违反情况，改正错误等方法进行处理。
考核要求：达到“识记”层次
知识点：规则的组成部分及分类

数据库完整性子系统是根据“完整性规则集” 工作的，
完整性规则是由数据库管理员或应用程序员事先向完整性子系统提供有关数据约束的一组规则。

由三部分组成：
（1）什么时候使用规则进行检查；
（2）要检查什么样的错误；
（3）若检查出错误，该怎样处理。

在关系数据库中，这些完整性规则可分为三类：
（1）用户定义完整性
（2）实体完整性
（3）参照完整性
1）主码约束：可用主码子句或主码短语来定义。
如 PRIMARY KEY (S#) (主码子句)

S# CHAR (4) PRIMARY KEY (其中，PRIMARY KEY 为主码短语）

（2）外码约束：可用外码子句来定义

如 FOREIGN KEY (S#) REFERNCE S(S#)

删除基本关系元组时的考虑	外码子句加"ON DELETE"短语
修改基本关系中主码值的考虑	外码子句加"ON UPDATE"短语
三种方式：NO ACTION, RESTRICT, CASCADE, SET NULL (具体含义)	

（3）属性值约束：包括非空值约束(NOT NULL)；基于属性的检查子句(CHECK)；

（4）全局约束：包括基于元组的检查子句(CHECK)

★在 employee 中插入记录，其 dno 值 not null， 但不在 department (dno) 中出现；

★修改 employee 的记录，新 dno 值 not null， 但不在 department (dno) 中出现；

★删除 department 的记录，但相应的 dno 值出现在 employee (dno) 中；

★修改 department 中的记录，但旧的 dno 值出现在 employee (dno) 中。

Restrict（限制策略）——SQL 的默认策略，

针对以上四种情况；

★

Cascade（级联策略）——针对以上后两种情况；

★

Set Null（置空策略）——针对以上后两种情况。

★

三种策略有 On Delete , On Update 可供选

对约束的命名、撤消和添加操作

（1）约束的命名

在定义时，前面加上关键字 CONSTRACT 和约束名即可。

（2）在关系上约束的撤消与添加

在关系定义中，撤消约束用“ALTER TABLE …DROP CONSTRAINT…”语句，添加约束用“ALTER TABLE …ADD CONSTRAINT …”语句

■ Check 约束：

★约束条件可以是任何 Where 中出现的字句；

故具有很强的表述能力；

★常用于数据的值域约束；

三、User Defined Integrity and Default、Check

Col_constr 列约束

➢一种域约束类型，对单一列的值进行约束

{ NOT NULL |

[CONSTRAINT constraintname]

{ UNIQUE

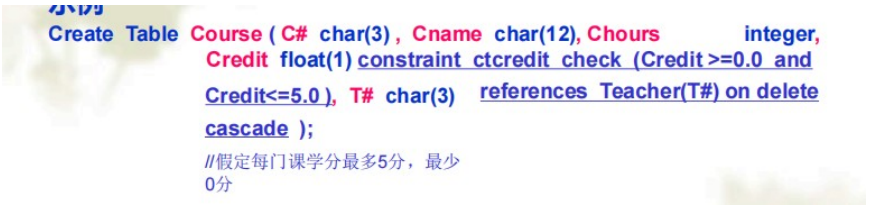
| PRIMARY KEY

| CHECK

(search_cond)

//列值非空

//为约束命名，便于以后撤消
//列值是唯一
//列为主键
//列值满足条件,条件只能使用列当前
值
| REFERENCES tablename [(colname)]
[ON DELETE { CASCADE | SET NULL }]
} }
//引用另一表 tablename 的列 colname 的值，如有 ON DELETE CASCADE 或 ON DELETE SET NULL 语句，则删除被引用表的某列值 v 时，要将本表该列值为 v 的记录删除或列值更新为 null；缺省为无操作 。
➢ Col_constr 列约束：只能应用在单一列上，其后面的约束如 UNIQUE, PRIMARY KEY 及 search_cond 只能是单一列唯一、单一列为主键、和单一列相关



➢一种关系约束类型，对多列或元组的值进行约束

[CONSTRAINT constraintname] //为约束命名，便于以后撤消

{ UNIQUE (colname {, colname. . .}) //几列值组合在一起是唯一

| PRIMARY KEY (colname {, colname. . .}) //几列联合为主键

| CHECK (search_condition) //元组多列值共同满足条件

//条件中只能使用同一元组的前值

| FOREIGN KEY (colname {, colname. . .})

REFERENCES tablename [(colname {, colname. . .})] [

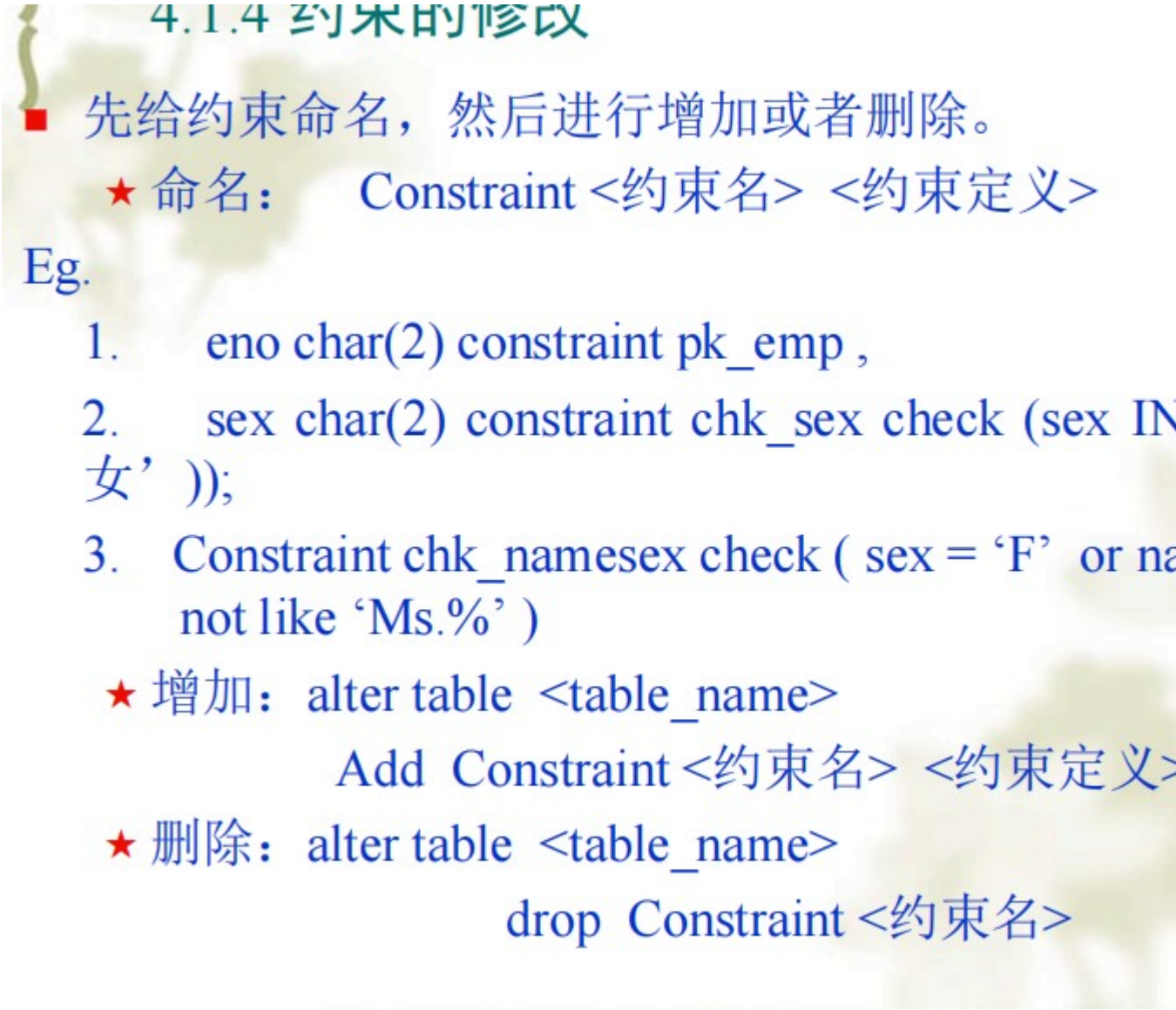
DELETE CASCADE] }

//引用另一表tablename的若干列的值作为外键

➢ table_constr表约束：是应用在关系上，即对关系的多列或

行约束， 列约束是其特例

```
Create Table Student ( S# char(8) not null unique, Sname
char(10), Ssex char(2) constraint ctssex check (Ssex= '男' or Ssex= '
女'), Sage integer check (Sage>1 and Sage<150), D#
        char(2) references Dept(D#) on delete cascade, Sclass char(6) ,
primary key(S#) );
    Create Table Course ( C# char(3) , Cname char(12), Chours integer,
        Credit float(1) constraint ctcredit check (Credit >=0.0 and
      Credit<=5.0 ), T# char(3) references Teacher(T#) on delete
      cascade, primary key(C#),
      constraint ctcc check(Chours/Credit = 20) );
      //假定严格约束20学时一个学分
```



是拒绝、级联删除、置空

1. 属性（列）级约束

- **NOT NULL**
  禁止该列出现 **NULL** 值（必须填写）。
- **UNIQUE**
  确保该列的值唯一（允许 **NULL**，但多个 **NULL** 可能被视为不同值，取决于数据库实现）。
- **CHECK**
  定义列值的检查条件，如 **CHECK (age >= 18)**。
- **DEFAULT**
  设置默认值，如 **DEFAULT 'unknown'**。

2. 表（外键）级约束
这些约束用于维护**参照完整性**（Referential Integrity），定义当**父表（被引用表）**的记录被更新或删除时，**子表（引用表）**如何处理相关记录：

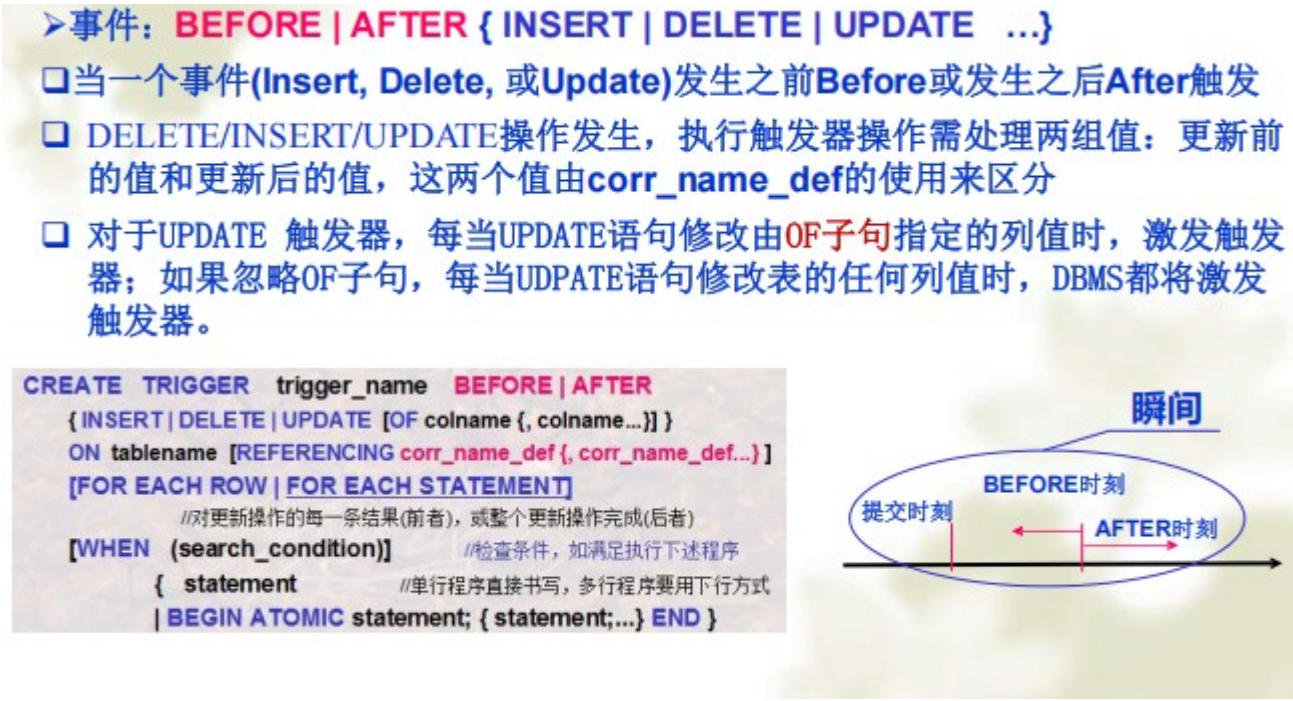
约束操作	说明
RESTRICT（拒绝）	**默认行为**，阻止删除或修改父表记录（如果子表有引用）。
NO ACTION	类似于 **RESTRICT**，但某些数据库（如 PostgreSQL）会在事务结束时检查。
CASCADE（级联）	父表记录被删除或更新时，**自动删除或更新子表对应的记录**。
SET NULL（置空）	父表记录被删除或更新时，子表的外键列设为 **NULL**（要求外键列允许 **NULL**）。
SET DEFAULT	父表记录被删除或更新时，子表的外键列设为默认值（需定义 **DEFAULT**）。

■ Check 约束：
★约束条件可以是任何 Where 中出现的字句；
故具有很强的表述能力；
★常用于数据的值域约束；
三、User Defined Integrity and Default、Check
Eg.
1. sex char(2) check (sex IN (‘男’,‘女’)) default‘男’，
2. age int check (age>=16 and age<=70)，
Eg.
1. sex char(2) check (sex IN (‘男’,‘女’)) default‘男’，
2. age int check (age>=16 and age<=70)，

触发器 Trigger
➢Create Table 中的表约束和列约束基本上都是静态的约束，也基本上都是对单一列或单一元组的约束(尽管有参照完整性)，为实现动态约束以及多个元组之间的完整性约束，就需要触发器技术 Trigger
➢Trigger 是一种过程完整性约束(相比之下，Create Table 中定义的都是非过程性约束),是一段 SQL 程序，该程序在对表或视图执行 UPDATE、INSERT 或 DELETE 操作时自动触发执行。
➢基本语法
CREATETRIGGER
trigger_name
BEFORE | AFTER

{ INSERT | DELETE | UPDATE [OF colname {, colname...}] }
ON tablename [REFERENCING corr_name_def {, corr_name_def...}]
[FOR EACH ROW | FOR EACH STATEMENT]
//对更新操作的每一条结果(前者:行级触发器)，或整个更新操作完成(后者：语句级触发器)
[WHEN (search_condition)]
{ statement
//检查条件，如满足执行下述程序
//单行程序直接书写，多行程序要用下行方式
| BEGIN ATOMIC statement; { statement;...} END }

➢ **触发器 Trigger 意义：当某一事件发生时(Before|After),对该事件产生的结果(或是每一元组，或是整个操作的所有元组),检查条件 search_condition，如果满足条件，则执行后面的程序段。条件或程序段中引用的变量可用 corr_name_def 来限定。**



属性上的约束条件：NOT NULL
UNIQUE
CHECK
表上的约束条件 CASCADDE NO ACTION RESTRICT

2.1 安全性
考核要求：**达到“识记”层次**
知识点：**安全性定义**

数据库的完整性是指尽可能避免对数据库的无意的滥用；
数据库的安全性是指尽可能避免对数据库的恶意的滥用。

数据库系统的安全措施是建立在计算机系统基础之上的，通常有五个方面。

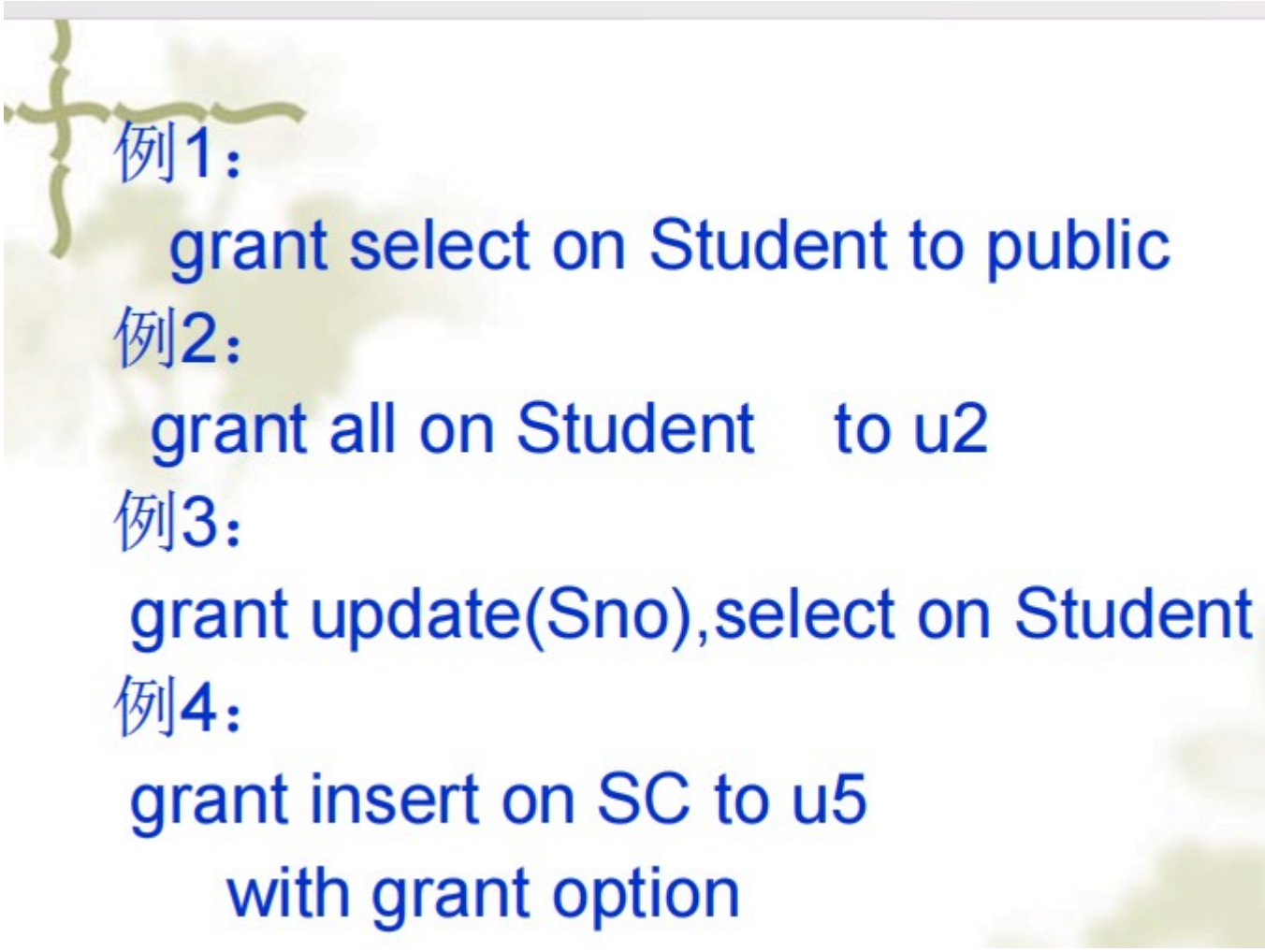
（1） 用户标识和鉴定
（2） 存取控制
（3） 定义视图
（4） 审计
（5） 数据加密
用户或应用程序使用数据库的方式称为权限。

授权子系统可以保证用户只能进行其权限范围内的操作，并允许有特定权限的用户有选择地和动态地把这些权限授予其他用户。
（1）用户权限
（2）授权语句
GRANT <权限表> ON <数据库对象> TO <用户名> [WITH GRANT OPTION]
（3）回收权限
REVOKE <权限表> ON <数据库对象> FROM <用户名> [CASCADE]
用户与角色：它们之间的关系、权限的继承性、角色授权的高效和便捷性
会判断用户加入某个角色后的最终的权限是什么（自己被授予的权限加上继承了角色的权限，但要去掉被 DENY 掉的权限）（详见第四章安全性补充 PPT）
视图是从一个或几个基本表导出的表，是虚表，视图定义后可以像基本表一样用于查询和删除， 但其更新操作(增、删、改)会受到限制。

视图机制把用户可以使用的数据定义在视图中，这样用户就不能使用视图定义外的其他数据，从而保证了数据库的安全性。
视图机制使系统具有三个优点：数据安全性、数据独立性和操作简便性。

事务是一个操作序列。 这些操作要么什么都做，要么都不做，是一个不可分割的工作单位

在应用程序中，事务以 **BEGIN TRANSACTION** 语句开始，以 **COMMIT(**提交)语句或 **ROLLBACK**(回退或撤消)语句结束。 一个程序的执行可通过若干事务的执行序列来完成。事务是不能嵌套的，可恢复的操作必须在一个事务的界限内才能执行.



3.2 事务的性质

事务的 ACID 性质：
原子性(atomicity)、一致性(consistency)、隔离性(isolation)和持久性(durability)。

3.3 故障类型和恢复方法

事务故障	非预期的事务故障，以由事务程序自动处理
系统故障	在硬件故障、软件错误的影响下，虽引起内存信息丢失，但未破坏外存中的数据。重新启动时,恢复子系统将所有非正常终止的事务回退，恢复到正确状态。
介质故障 计算机病毒等	需要 DBA 的介入， 通过 DBMS 把其他务份磁盘或第三级介质中的内容再复制回来

3.4 恢复的基本原则和实现方法

恢复的基本原则很简单，就是“冗余”，即数据的重复存储。
实现方法有：
(1)定期对数据库进行复制或转储(dump)。
Notice:几个概念：静态转储、动态转储、海量转储和增量转储。

- 静态存储是指转储期间不允许对数据库进行任何存取、修改活动；
- 动态存储是指转储期间允许对数据库进行存取、修改；即转储和用户事务可以并发执行；
- 海量存储是指每次转储全部数据库；
- 增量存储则指每次只转储上次转储后更新过的数据。

(2)建立“日志”文件。
(3)恢复。发生故障时有两种处理方法，如数据库已破坏， 则由 DBA 装入最近备份的数据库然后利用"日志文件"执行 REDO(重做)操作。如数据库未被损坏， 但某些数据不可靠，则系统会自动执行 UNDO 操作恢复数据。
注意每种故障策略的不同恢复手段。
并发控制带来的三类问题
(1)丢失更新
(2)不可重复读
(3)读“脏数据”(在数据库技术中，未提交的随后又被撤消的数据为“脏数据”。)
以上三类问题要做到理解。

解决并发控制带来的问题，通常要采用封锁(locking)技术，常用的封锁有：排它型封锁(X 封锁)和共享型封锁(S 封锁)两种。

4.2 封锁
考核要求：达到“领会”层次
知识点：排它锁（X LOCK），共享锁（S LOCK），并发事务的可串行化

封锁就是事务 T 可以向系统发出请求，对某个数据对象（通常是记录）加锁。 于是事务 T 对这个数据对象就有一定的控制，其他事务不能更新此数据直到 T 释放(Unlock)它的锁为止。

X 封锁：如果事务 T 对数据 R 实现 X 封锁，那么其他的事务要等 T 解除 X 封锁以后，才能对这个数据进行封锁。 只有获准 X 封锁的事务，才能对被封锁的数据进行修改。

S 封锁--如果事务 T 对某数据 R 加上 S 封锁，那么其它事务对数据 R 的 X 封锁便不能成功，而对数据 R 的 S 封锁请求可以成功。 这就保证了其他事务可以读取 R 但不能修改 R，直到事务 T 释放 S 封锁。

事务的执行次序称为“调度”。如果多个事务依次执行，则称为事务的串行调度。 如果利用分时的方法，同时处理多个事务，则称为事务的并发调度。
可串行化调度：如果一个并发调度的结果与某一串行调度执行结果等价，那么这个并发调

度称为是可串化的调度。

4.3 活锁和死锁

考核要求：达到“领会”层次

知识点：活锁及其避免方法；死锁及消除方法

活锁：是指某个事务永远处于等待状态，得不到执行的现象。

避免活锁的方法是采用“先来先服务”策略。

死锁：有两个或以上的事务处于等待状态，每个事务都在等待另一个事务解除封锁， 它才能继续执行下去，结果任何一个事务都无法执行，这种现象就是死锁。

解除死锁的方法是由 DBMS 中的“死锁测试程序”来检查，如发现死锁则牺牲一个事务，并做回退操作,解除它的所有封锁。

4.4 两段封锁法

考核要求：达到“识记”层次

知识点：两段封锁法

两段封锁协议规定所有事务都要遵守下列规则：

(1)在对任何数据进行读写操作之前，事务首先要获得对该数据的封锁；

(2)在释放一个封锁之后，事务不再获得任何其他封锁。

事务：扩展阶段——可以申请封锁，但是不能解除任何已取得的封锁

收缩阶段——可以释放封锁，但不能申请新的封锁

1. 触发器

1.1 触发器的概念、作用、使用场景

1.2 创建触发器的语法：

Create trigger　触发器名

On/After　insert | update | Delete

AS

需要触发完成的 SQL 语句

1.3 触发器中使用游标：

游标使用五部曲：

（1）说明游标

（2）打开游标

（3）推进游标指针并取当前记录

（4）关闭游标

（5）释放游标占用的资源

```
DECLARE @no    char(7), @name char(10), @age tinyint
DECLARE SX CURSOR FOR    SELECT Sno, Sname, Sage
            FROM Student                    /* 说明游标 */
OPEN SX ;                                   /* 打开游标 */
FETCH SX INTO    @no,@name,@age ;
            游/*进游标指针推进一行，然后从结果集中取当前行，送相应主量
WHILE @@fetch_status= 0      /* 如果取到数据，用循环结构逐条处理结果集中的记录 */
BEGIN                                       /* 若出现 SQL 语句错误，则退出循环 */
                ......                  /*可以根据三个变量的值做相应的处理*/
     FETCH SX INTO    @no,@name,@age ;
            /*游/*标指针向前推进一行，然后从结果集中取当前行，送相应主量
*/
END
CLOSE SX;                   /* 关闭游标 */
DEALLOCATE SX;          /* 释放游标所占用的系统资源*/
```

1.4 触发器的调用　（触发器无需用户调用，是在对表进行数据增删改时自动触发的）

2. 存储过程

2.1 存储过程的概念、作用、使用场景

2.2 创建存储过程的语法：

2.3 存储过程的调用

详见教材 P123 5.5 存储过程

2 关系代数运算

2.1 关系代数的五个基本操作

考核要求：达到“简单应用”层次

知识点：五个基本操作的含义和运算应用

(1)并(**∪**)：两个关系需有相同的关系模式，并的对象是元组，由两个关系所有元组构成。
RUS≡{t| t∈R ∨t∈S}
(2) 差(-)：同样，两个关系有相同的模式，R 和 S 的差是由属于 R 但不属于 S 的元组构成的集合。
R-S≡{t| t∈R ∧t 不属于 S}
(3)笛卡尔积（×）：对两个关系 R 和 S 进行操作，产生的关系中元组个数为两个关系中元组个数之积。
R×S≡{t| t=< t^r,t^s>∧t^r∈R∧t^s ∈S}
(4) 投影(**π**)：对关系进行垂直分割，消去某些列，并重新安排列的顺序。
(5) 选择(**σ**)：根据某些条件关系作水平分割，即选择符合条件的元组。
1)交(∩)：R 和 S 的交是由既属于 R 又属于 S 的元组构成的集合。
(2)连接:包括 **θ**(算术比较符)连接和 **F(**公式)连接.
选择 R×S 中满足 iθ(r+j)或 F 条件的元组构成的集合；
概念上比较难理解，关键理解运算实例
等值连接(θ 为等号“=”的连接)。
(3)自然连接(R⋈S)：在 R×S 中，选择 R 和 S 公共属性值均相等的元组，并去掉 R×S 中重复的公共属性列。 如果两个关系没有公共属性，则自然连接就转化为笛卡尔积。
(4)除法(÷)：首先除法的结果中元数为两个元数的差，

> R÷S 的操作思路如下---把 S 看作一个块，如果 R 中相同属性集中的元组有相同的块， 且除去此块后留下的相应元组均相同，那么可以得到一条元组， 所有这些元组的集合就是除法的结果

对于上述的五个基本操作和四个组合操作，应当从实际运算方面进行理解和运用。

应用举例：
设有关系 R 和 S（如下：）

R

A	B	C
3	6	7
2	5	7
7	2	3
4	4	3

S:

A	B	C
3	4	5
7	2	3

计算：

$R\cup S$，$R-S$，$R\cap S$，$R\times S$，$\pi_{3,2}(S)$，$B<'5'(R)$，$R\underset{2<2}{\bowtie}S$，$R\bowtie S$。

$R\cup S$

A	B	C
3	6	7
2	5	7
7	2	3
4	4	3
3	4	5

$R-S$

A	B	C
3	6	7
2	5	7
4	4	3

$R\cap S$

A	B	C
7	2	3

$R\times S$

R.A	R.B	R.C	S.A	S.B	S.C
3	6	7	3	4	5
3	6	7	7	2	3
2	5	7	3	4	5
2	5	7	7	2	3
7	2	3	3	4	5
7	2	3	7	2	3
4	4	3	3	4	5
4	4	3	7	2	3

$\pi_{3,2}(S)$

C	B
5	4
3	2

$\sigma_{B<'5'}(R)$

A	B	C
7	2	3
4	4	3

$R\underset{2<2}{\bowtie}S$

R.A	R.B	R.C	S.A	S.B	S.C
7	2	3	3	4	5

$R\bowtie S$.

A	B	C
7	2	3

有关实际应用，应该多看例题，多做习题，必须达到以下要求： 能够根据给出的关系代数表达式计算关系值，也能够根据相应查询要求列出关系表达式。

（1）在列关系表达式时，通常有以下形式：

π...(σ...(R×S))或者 π...(σ...(R⋈S))

首先把查询涉及到的关系取来，执行笛卡尔积或自然连接操作得到一张大的表格，然后对大表格执行水平分割(选择)和垂直分割(投影)操作。

（2）当查询涉及到否定或全部的逻辑时，往往要用到差或除法操作。

学生关系 S(SNO,SNAME,AGE,SEX)

学习关系 SC(SNO,CNO,GRADE)

课程关系 C(CNO,CNAME,TEACHER)

下面用关系代数表达式表达每个查询语句。

(1) 检索学习课程号为 C2 的学生学号与成绩。

$\pi_{SNO,\ GRADE}(\sigma_{CNO='C2'}(SC))$

(2) 检索学习课程号为 C2 的学生学号与姓名

$\pi_{SNO,\ SNAME}(\sigma_{CNO='C2'}(S\bowtie SC))$

由于这个查询涉及到两个关系 S 和 SC，因此先对这两个关系进行自然连接，同一位学生的有关的信息，然后再执行选择投影操作。

此查询亦可等价地写成：

$\pi_{SNO,\ SNAME}(S)\bowtie(\pi_{SNO}(\sigma_{CNO='C2'}(SC)))$

这个表达式中自然连接的右分量为"学了 C2 课的学生学号的集合"。这个表达式比前一个表达式优化，执行起来要省时间，省空间。

（3）检索选修课程名为 MATHS 的学生学号与姓名。

$\pi_{SNO，SANME}(\sigma_{CNAME='MATHS'}(S \bowtie SC \bowtie C))$

（4）检索选修课程号为 C2 或 C4 的学生学号。

$\pi_{SNO}(\sigma_{CNO='C2' \vee CNO='C4'}(SC))$

（5） 检索**至少**选修课程号为 C2 或 C4 的学生学号。

$\pi_1(\sigma_{1=4 \wedge 2='C2' \wedge 5='C4'}（SC \times SC）)$

这里（SC×SC）表示关系 SC 自身相乘的乘积操作，其中数字 1，2，4，5 都为它的结果关系中的属性序号。 这个大家理解即可，太绕了，可以用除法实现：

$\pi_{SNO,CNO}(SC) \div \pi_{CNO}(\sigma_{CNO='C2' \vee CNO='C4'}(SC))$

（6） 检索不学 C2 课的学生姓名与年龄。

$\pi_{SNAME，AGE}（S）- \pi_{SNAME，AGE}(\sigma_{CNO='C2'}（S \bowtie SC）)$

这个表达式用了**差运算**，差运算的左分量为"全体学生的姓名和年龄"，右分量为"学了 C2 课的学生姓名与年龄"。

（7）检索学习全部课程的学生姓名。

编写这个查询语句的关系代数过程如下：

(a) 学生选课情况可用 $\pi_{SNO,CNO}(SC)$表示；

(b) 全部课程可用 $\pi_{CNO}(C)$表示；

(c) 学了全部课程的学生学号可用**除法操作**表示。

操作结果为学号 SNO 的集合，该集合中每个学生（对应 SNO)与 C 中任一门课程号 CNO 配在一起都在 $\pi_{SCO，CNO}$（SC） 中出现（即 SC 中出现），所以结果中每个学生都学了全部的课程（这是"除法"操作的含义）：

$\pi_{SNO,CNO}(SC) \div \pi_{CNO}(C)$

(d) 从 SNO 求学生姓名 SNAME，可以用自然连结和投影操作组合而成：

$\pi_{SNAME}(S \bowtie (\pi_{SNO,CNO}(SC) \div \pi_{CNO}(C)))$

这就是最后得到的关系代数表达式。

（8） 检索所学课程包含 S3 所学课程的学生学号。

注意：学生 S3 可能学多门课程，所以要用到除法操作来表达此查询语句。

学生选课情况可用操作 $\pi_{SNO,CNO}(SC)$表示；

所学课程包含学生 S3 所学课程的学生学号，可以用除法操作求得：

$\pi_{SNO,CNO}(SC) \div \pi_{CNO}(\sigma_{SNO='S3'}(SC))$

查询优化的目的就是为了系统在执行时既省时间又能提高效率。
在关系代数运算中，通常是先进行笛卡尔积或连接运算，再进行选择和投影。笛卡尔积或连接运算却往往花费教多的时间。
因此，恰当地安排选择、投影和连接的顺序直接影响到整个操作所需要的时间和空间。 如何安排若干关系的运算操作步骤，是查询优化所要考虑的问题。

3. 2 关系代数表达式的等价变换规则

考核要求：达到“ 识记” 层次
知识点：等价变换规则

两个关系代数表达式等价是指用同样的关系实例代替两个表达式中相应关系时所得到的结果是完全一样的。

等价变换规则有很多，不要死记，从语义上理解。
其实，只要在前面的学习中已经掌握关系运算的真正含义，就可以判断两个关系代数表达式是否等价。

3. 3 优化的策略

考核要求：达到“ 领会” 层次
知识点：优化的策略及其简单应用

优化的策略主要有以下几点：
(1) 在关系代数表达式中尽可能早地执行选择操作；
(2) 把笛卡尔积和随后的选择操作合并成 F 连接运算；
(3) 同时计算一连串的选择和投影操作；
(4) 保留同一子表达式的结果；
(5) 适当对关系文件进行预处理 ；
(6) 计算表达式之前先估计一下怎么计算合算。

以上优化策略要求会简单应用：先做选择，运用投影去除多余属性等等。

3. 4 关系代数表达式的优化算法

考核要求：达到“ 简单应用” 层次
知识点：语法树

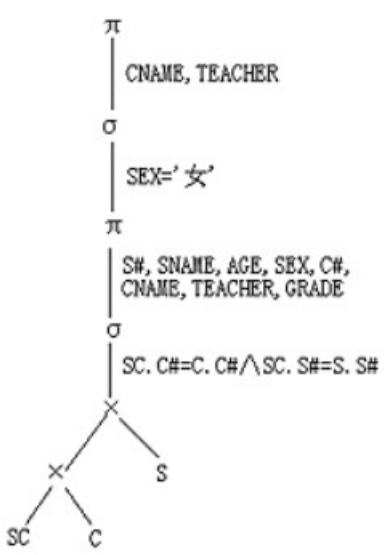
学会画语法树，并掌握优化算法。

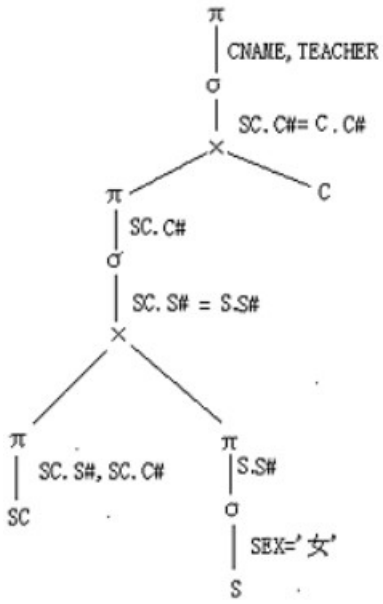
举例
在教学数据库 S、SC、C 中，用户有一查询语句：检索女同学选修课程的课程名和任课教师名。
(1) 试写出该查询的关系代数表达式；
(2) 试写出查询优化的关系代数表达式；
(3) 画出该查询初始的关系代数表达式的语法树；
(4) 使用 2. 4. 4 节的优化算法，对语法树进行优化，并画出优化后的语法树。

(1)$\pi_{CNAME,TEACHER}(\sigma_{SEX='女'}(S \bowtie SC \bowtie C))$

(2)优化为：$\pi_{CNAME，TEACHER}(C \bowtie \pi_{C\#}(\pi_{S\#,C\#}(SC) \bowtie \pi_{S\#}(\sigma_{SEX='女'}(S))))$

(基本思路：尽量提前做选择操作；在每个操作后，应做个投影操作，去掉不用的属性值。





关系数据库是以关系模型为基础的数据库，它利用关系来描述现实世界。

一个关系既可以用来描述一个实体及其属性，也可以用来描述实体间的联系。 关系实质上就是一张二维表，表的行称为元组，列称为属性。 关系是元组的集合，关系模式就是这个元组集合的描述。

关系模式是用来定义关系的， 一个关系数据库包含一组关系，也就是包含一组二维表，这些二维表结构体的集合就构成数据库的模式(也可以理解为数据库的结构)。

关系数据库设计理论包括三个方面内容：数据依赖、范式、模式设计（分解）方法。

核心内容是数据依赖。

2 函数依赖(FD)

2.1 函数依赖的定义

考核要求：达到“领会”层次

知识点：函数依赖的定义

设有关系模式 R(A1，A2，...An) 或简记为 R(U)，X，Y 是 U的子集，r 是 R的任一具体关系， 如果对 r 的任意两个元组 t1, t2, 由 t1[X]=t2[X] 导致 t1[Y]=t2[Y]，则称 X 函数决定 Y，或 Y 函数依赖于 X， 记为 X→Y。X→Y 为模式 R的一个函数依赖。

该定义理解如下：有一张设计好的二维表，X，Y 是表的某些列(可以是一列，也可以是多列)，若在表中的第 t1 行，和第 t2 行上的 X 值相等， 那么必有 t1 行和 t2 行上的 Y 值也相等，这就是说 Y 函数依赖于 X。

比如，有如下二维表

学号	姓名	成绩	成绩等级
00001	李里	77	C
00002	丁力	91	A
00003	李小红	85	B
00004	马琳	85	B
00005	王佳怡	66	D
00006	胡林	70	C
...	.....	.....	.....

在表中，凡成绩相同的，对应的“成绩等级”也必是相同的，因此，“成绩等级”函数依赖于"成绩"。但是反过来则不成立。

Notice:

(1)在这张表中，任何一行的关系均应符合函数依赖的条件，如果有一行不符合函数依赖的条件，则函数依赖对于这个关系就不成立。

(2)函数依赖是否成立是不可证明的，只能通过属性的含义来判断.

码是唯一标识实体的属性集。

码和函数依赖的关系满足以下两个条件：

设关系模式 R(A1,A2...An)，F 是 R 上的函数依赖集，X 是 R 的一个子集，如果

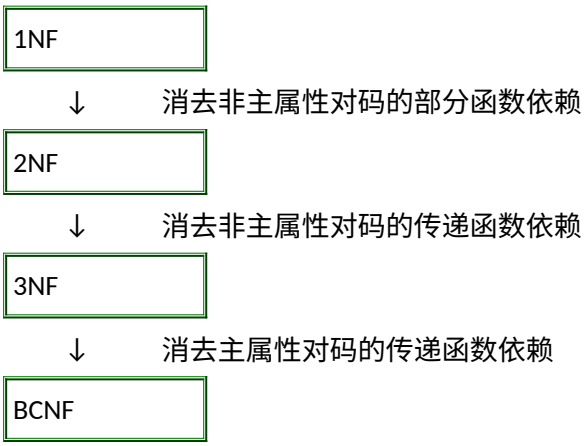
(1)X→A1A2...An∈F+

(X 能够决定唯一的一个元组)

(2)不存在 X 的真子集 Y，使得 Y→A1A2...An 成立
（X 能满足（1）但又没有多余的属性集）
则 X 就是 R 的一个候选码。
包含在任何一个候选码中的属性称为主属性，不包含在任何码中的属性为非主属性(非码属性)， 注意主属性应当包含在候选码中（只需要出现在一个候选码中就是主属性）。
1NF：第一范式—— 即关系模式中的属性的值域中每一个值都是不可再分解的值。
如果某个数据库模式都是第一范式的，则称该数据库模式是属于第一范式的数据库模式。
如果关系模式 R 为第一范式，并且 R 中每一个非主属性完全函数依赖于 R 的某个候选码，则称为第二范式模式。

首先温习、理解“非主属性”、“完全函数依赖”、“候选码”这三个名词的含义。
（1）候选码：可以唯一决定关系模式 R 中某元组值且不含有多余属性的属性集。
（2）非主属性：即非码属性，指关系模式 R 中不包含在任何建中的属性。
（3）完全函数依赖：设有函数依赖 W→A，若存在 X⊆W，有 X→A 成立，那么称 W→A 是局部依赖，否则就称 W→A 是完全函数依赖。
如果关系模式 R 是第二范式，且每个非主属性都不传递依赖于 R 的候选码，则称 R 为第三范式模式。
传递依赖的含义： 在关系模式中，如果 Y→X，X→A，且 X　Y(X 不决定 Y)和 A　X(A 不属于 X)，那么 Y→A 是传递依赖。 Notice:要求非主属性都不传递依赖于候选码。

上一小节例子中 student（学号，姓名），study（学号，课程，成绩）都是 3NF
这个范式和第三范式有联系，它是 3NF 的改进形式。
若关系模式 R 是第一范式，且每个属性都不传递依赖于 R 的候选码。这种关系模式就是 BCNF 模式。
四种范式，可以发现它们之间存在如下关系：
BCNF⊆3NF⊆2NF⊆1NF



SQL 语言
数据定义语言 DDL (Data Definition Language)
用于创建、更改或删除数据库对象，如包括定义 Database, Table, View, Index,完整性约束条件等，也包括定义对象(RowType 行对象, Type 列对象)
语句包括：**CREATE（建立）、ALTER（修改）、DROP（撤销）**
v

数据操纵语言 DML (Data Manipulation Language)
用于操纵数据库对象（如表）。
ü 各种方式的更新与检索操作，如直接输入记录，从其他 Table(由 SubQuery 建立)输入
ü 各种复杂条件的检索，如连接查找，模糊查找，分组查找，嵌套查找等
ü 各种聚集操作，求平均、求和、…等，分组聚集，分组过滤等
语句包括：**SELECT、INSERT、UPDATE、DELETE**
v
数据控制语言 DCL (Data Control Language)
用于定义控制访问对象（如表）。
v 安全性控制：授权和撤消授权
语句包括：**GRANT、REVOKE**
SQL 的数据定义语句（DDL）包括以下语句：
创建 删除 修改
表 CTEATE TABLE DROP TABLE ALTER TABLE
视图 CTEATE VIEW DROP VIEW ALTER VIEW
索引 CTEATE INDEX DROP INDEX
n 注意：索引无修改语句!

创建 Database
Ø 数据库**(Database)**是若干具有相互关联关系的 **Table/Relation** 的集合
Ø 数据库可以看作是一个集中存放若干 **Table** 的大型文件
Ø**create database** 的简单语法形式：
create
database
数据库名;
示例：创建课程学习数据库 SCT
create
database
SCT;

嵌入式 sql
v
SQLCA： SQL Communication Area
n SQLCA 是一个数据结构
v
SQLCA 的用途
n SQL 语句执行后，RDBMS 反馈给应用程序信息
Ø 描述系统当前工作状态
Ø 描述运行环境
n 这些信息将送到 SQL 通信区 SQLCA 中
n 应用程序从 SQLCA 中取出这些状态信息，据此决定接下
来执行的语句

存储过程

存储过程（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，可供应用程序在需要时调用。
允许模块化程序设计，含有控制语句
存储过程能够实现较快的执行速度
存储过程能够减少网络流量
存储过程可被作为一种安全机制来充分利用
带参数的存储过程举例

```
CREATE PROCEDURE pro_student_sno
@givensno char(7)
AS
SELECT sno,sname
FROM student
WHERE sno=@givensno;
```

n 执行

```
EXEC pro_student_sno @givensno ='2000101'
EXEC pro_student_sno '2000101';、
```

可授权的数据对象：数据库、模式（全局模式、
外模式），数据（表、属性）
可以授权的操作：
数据库的建立、删除、连接
模式的建立、修改、检索
数据的建立、检索、修改
对应的 SQL 语句：
GRANT, REVOKE，数据库安全性控制是属于
DCL 范畴。
可以结合视图加强对数据库存取的控制 GRANT 命令：把权限授予某一用户，以允许该用户执行针
对该对象的操作或允许其运行某些语句。
❖REVOKE 命令：用来撤销用户对某一对象或语句的权限，使
其不能执行操作，除非该用户是角色成员，且角色被授权。
❖DENY 命令：用来禁止用户对某一对象或语句的权限，它不
允许该用户执行针对数据库对象的某些操作或不允许其运行某
些语句。对象
对象类
型
操作权限
属性列 Column Select,insert,update,delete,all
视图
view
Select,insert,update,delete,all
基本表
table

Select,insert,update,delete,alter,

index ,all

数据库 datebase createdbGRANT {all PRIVILEGES | privilege {,privilege...}} ON[TABLE] tablename | viewname

TO {public | user-id {, user-id...}} [WITH GRANT OPTION];

❑ user-id ，某一个用户账户，由 DBA 创建的合法账户

❑ *public*,

允许所有有效用户使用授予的权利

❑privilege 是下面的权利

✓ SELECT | INSERT | UPDATE | DELETE | ALL PRIVILEDGES

❑WITH GRANT OPTION 选项是允许被授权者传播这些权利取消权限

取消权限的 SQL 语句是：

REVOKE <权限列表>

ON <表名或视图名>

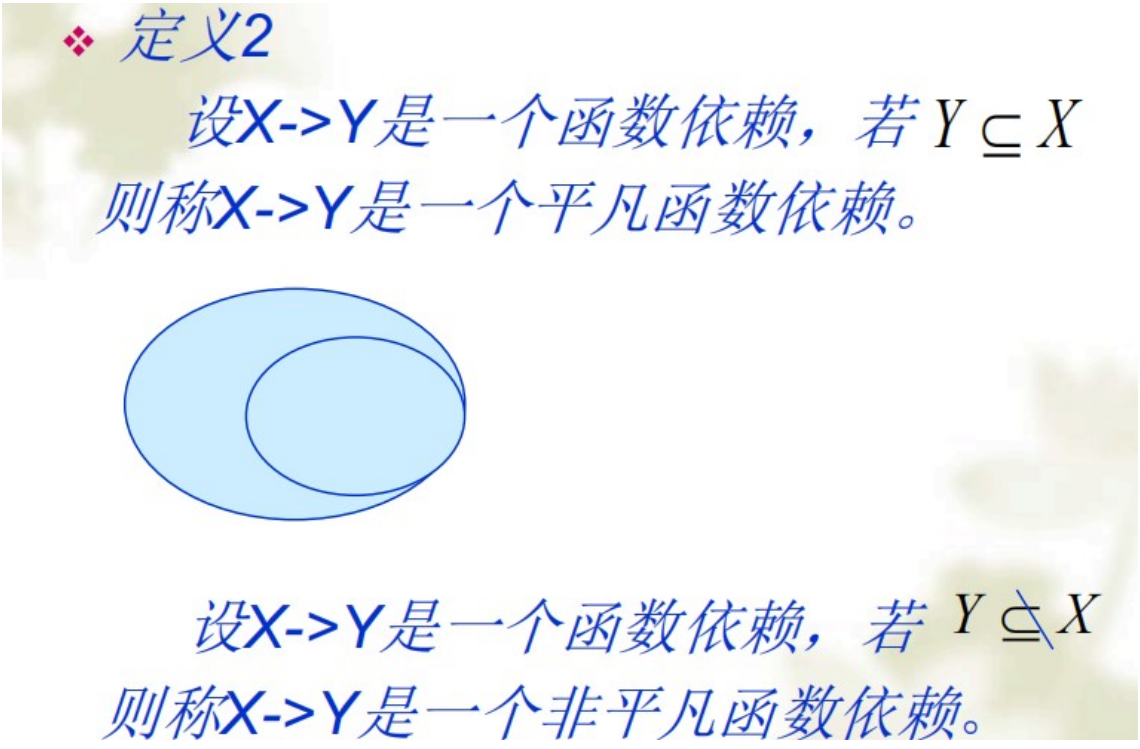
FROM <用户列表>

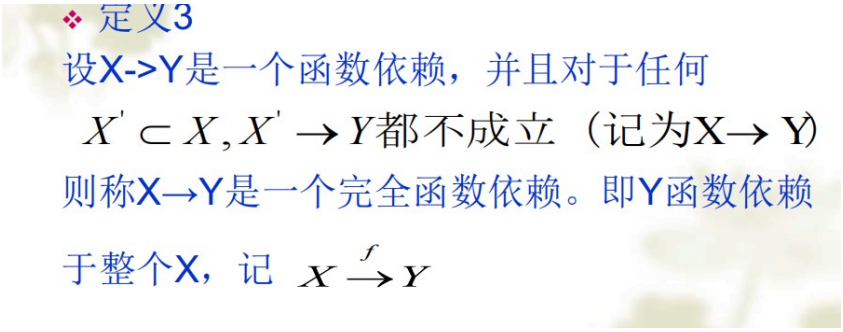
[CASCADE|RESTRICT]；

函数依赖

定义 1

设 R(U)是属性集 U 上的关系模式，X，Y 是 U 的子集。若对于 R(U)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





设 X->Y 是一个函数依赖,但不是完全函
数依赖,则称 X->Y 是一个部分函数依赖,或称
Y 函数依赖于 X 的某个真子集,记 *X Y* ***闭包求解与码***
关系 R(U,F) ***中，其中某个给定属性集*** K Í U ***，当且仅当*** K
关于给定函数依赖集 F ***的闭包*** K+ ***是*** R ***的所有属性集合*** U ***时，***
K ***即为关系*** R ***的超码。***
当且仅当属性集 K ***中不存在任一真子集*** K' ***的闭包***(K') +***也***
是 R ***的所有属性集合*** U ***时，即属性集*** K ***是最小属性集合构***
成的超码时， K ***就是该关系*** R(U,F) ***的候选码。候选码的求解理论和算法***
v
对于给定的关系 R （A1A2...An）***和函数依赖集*** F，***可***
将其属性分为 4 ***类：***
L ***类：仅出现在*** F ***函数依赖左部的属性***
R ***类：仅出现在*** F ***函数依赖右部的属性***
N ***类：在*** F ***函数依赖的左右两部均未出现的属性***
LR ***类：在*** F ***函数依赖的左右两部均出现的属性***
v
定理：对于给定的关系模式 R ***及其函数依赖集*** F，***若*** X
是 R ***的*** L ***类属性，则*** X ***必为*** R ***的任一侯选码的成员。***
v
推论：对于给定的关系模式 R ***及其函数依赖集*** F，***若*** X
是 R ***的*** L ***类属性，且*** X+ ***包含了*** R ***的全部属性，则*** X ***必为*** R
的唯一侯选码

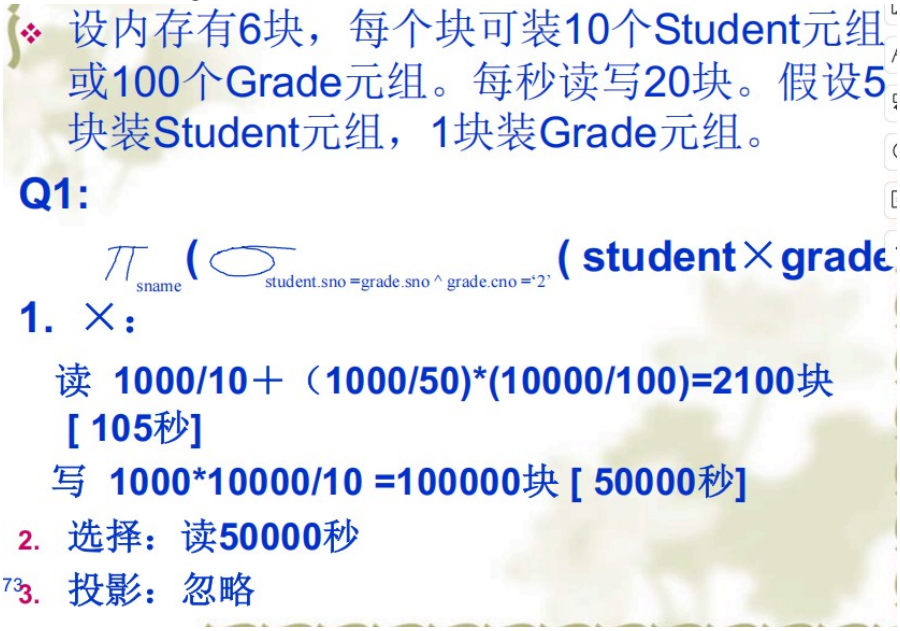
最小依赖集

1、 右部最小化

2、 除本闭包（如果推出了本身，就不需要这个以来条件）

3、 左部最小化

假设有 1000 块 grade 元组，1000 块 student 元组



1、一共装了 50 个 student 元组、100 个 grade 元组

首先从内存中读取数据。

读取 student 表的所有块

查询优化的一般策略

- 选择尽可能先做
- 连接前预处理
  - 建索引
  - 排序
- 选择与投影合并
- 投影与其他双目运算合并
- 选择与笛卡儿积合并成连接
- 公共子表达式只计算一次

从下往上写