#### 并发控制

并发控制带来的数据不一致性：

1. **丢失修改**：两个事务T1和T2读入同一数据并修改，T2的提交结果破坏了T1提交的结果，导致T1的修改被丢弃。
2. **不可重复读**：事务T1读取数据后，事务T2执行更新操作，是T1无法在现前一次读取的结果。
3. **读脏数据**：事务T1修改某一数据，并将其写回磁盘。事务T2读取同一数据后，T1由于某种原因被撤销，这时T1已经修改过的数据恢复原值，T2读到的数据就与数据库中的数据不一致。

**数据不一致的原因**：由于并发操作破坏了事务的隔离性

**并发控制的主要技术**：封锁

**封锁**：事务T在对某个数据对象操作之前，先向系统发出请求，对其加锁。加锁后事务对数据对象有了一定的控制，在事务T释放它的锁之前，其他事务不能更新此数据对象。

**基本锁类型**：

1. **排它锁**（X），写锁，若事务T对数据对象A加上X锁，则只允许T读取和修改A，其他事务都不能再对A加任何类型的锁，直到T释放A上的锁。
2. **共享锁（**S），读锁，保证其他事务可以读A，但在T释放A上的S锁之前，不能对A做任何修改。

**三级封锁协议**

1. **一级封锁协议**：事务T在修改数据R之前必须对其加X锁，直到事务结束。防止丢失修改
2. **二级封锁协议**：一集封锁协议加上事务T在读取数据R之前必须先对其加S锁，读完后即可释放S锁。 防止丢失修改和读脏数据
3. **三级封锁协议**：一级封锁协议加上事务T在读取数据R之前必须先对其加S锁，直到事务结束后才释放S锁。防止丢失修改、读脏数据和不可重复读。

**封锁协议带来的问题**：死锁和活锁

**避免活锁**：采用先来先服务策略

**避免死锁**：诊断并解除死锁（超时法、等待图法）

**等待图法**：并发控制子系统周期性地生成事务等待图，检测事务。如果发现图中存在回路，则表示系统出现了死锁。解除死锁：选择一个处理死锁代价最小的事务，将其撤销，释放事务持有的所有锁，使其他事务能继续运行。

并发调度的可串行性：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。

**可串行性**：是并发事务正确调度的准则。

**冲突操作**：是指不同事务对同一数据的读写操作和写写操作

不能交换的动作：

1. 同一事务的两个操作
2. 不同事务的冲突操作

**冲突可串行化调度**：一个调度Sc在保证冲突操作的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度Sc’，如果Sc‘是串行的，称调度Sc是冲突可串行化调度。如果一个调度是冲突可串行化，则一定是可串行化的调度。

**两端锁协议**：数据库管理系统普遍采用两端锁协议的方法实现并发调度的可串行，从而保证调度的正确性。在对任何数据进行读、写操作之前，事务首先获得对数据的封锁，在释放封锁后，事务不再申请和获得任何其他封锁。

**封锁的粒度**：封锁对象的大小称为封锁粒度。

**多粒度封锁**：

1. 需要处理多个关系的大量元祖的用户事务：以数据库为封锁单位
2. 需要处理大量元祖的用户事务：以关系为封锁单元
3. 只处理少量元祖的用户事务：以元祖为封锁单位。

**多粒度树**：允许多粒度树中每个结点被独立地加锁。对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。

**在多粒度封锁中一个数据对象可能以两种方式封锁**：显式封锁和隐式封锁

**显式封锁**：直接加到数据对象上的封锁

**隐式封锁**：由于其上级结点加锁而是数据对象加上锁

显示和隐式效果一样，所以系统检查封锁冲突时要同时检查两种类型的锁。

**意向锁**：提高对某个数据对象加锁是系统检查效率。如果对一个结点加意向锁，则说明该节点的下层结点正在被加锁。对任一结点加基本锁，必须先对它的上层结点加意向锁。

#### 数据库恢复技术

**故障种类**：1、事务内部的故障；2、系统故障；3、介质故障；4、计算机病毒。

**事务内部故障恢复**：事务撤销

1. 强行回滚该事务
2. 撤销该事务已经作出的任何对数据库的修改，使得该事务像根本没有启动一样。

**系统故障时，一些尚未完成的事务结果可能写入物理数据库，造成数据库可能处于不正确的状态。**

**系统故障恢复**：系统重新启动时，恢复程序让所有非正常终止的事务回滚，强行撤销所有未完成的事务。

发生系统故障时，有些已完成的事务的修改结果可能有一部分甚至全部留在缓冲区，尚未写入磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失。

恢复策略：系统重新启动时，恢复程序需要重做所有已提交的事务。

**恢复操作的基本原理**：冗余。利用存储在系统别处的冗余数据来重建数据库中已被破坏或不正确的那部分。

**如何建立冗余数据**：

1. 数据转储
2. 登记日志文件

**静态转储**：在系统无运行事务时进行的转储操作。

1. 转储开始数据处于一致性状态。
2. 转储期间不允许对数据库的任何存取、修改活动
3. 得到的一定是一个数据一致性的副本

**一致性**：数据库中只包含事务提交的结果

**隔离性**：一个事务的执行不能被其他事务干扰

**持续性**：一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。

**原子性**：事务是数据库的逻辑工作单位

**动态转储**：转储期间允许对数据进行存取或修改。利用此技术，需要把动态转储期间各事务对数据库的修改活动登记下来，建立日志文件。后备副本加上日志文件就能把数据库恢复到某一时刻的正确状态。

**日志文件格式**：

1. 以记录为单位的日志文件
2. 以数据块为单位的日志文件

**以记录为单位的日志文件，每条日志记录的内容**：

1. 事务标识
2. 操作类型
3. 操作对象
4. 更新前旧值
5. 更新后新值

**以数据块为单位的日志文件，每条日志记录的内容**：

1. 事务标识
2. 被更新的数据块

**日志文件用途**：

1. 进行事务故障恢复
2. 进行系统故障恢复
3. 协助后备副本进行介质故障恢复

**登记日志文件原则**：

1. 登记次序严格按照并发事务执行的时间次序。
2. 必须先写日志文件，后写数据库

**恢复策略**：

1. 事务故障的恢复：由恢复子系统用日志文件撤销次事务对数据库进行的修改。

具体操作：反向扫描文件日志，查找该事务的更新操作；然后对事务的更新操作执行逆操作；继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

1. 系统故障的恢复：撤销故障发生时未完成的事务。重做已经完成的事务

具体操作：首先正向扫描，重做完成的事务。然后反向扫描，撤销未完成的事务。

1. 介质故障的恢复：重装数据库；重做已经完成的事务

具体操作：首先扫描日志文件，找出故障发生时已提交的事务的表示，将其计入重做队列。然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中的更新后的值写入数据库。

**具有检查点的恢复技术：解决搜索整个日志文件消耗大量时间以及重做处理浪费大量时间的问题。**

1. 在日志文件增加检查点记录
2. 增加重新开始文件
3. 恢复子系统在登录日志文件期间动态维护日志。

**检查点记录的内容**：

1、建立检查点时刻所有正在执行的事务清单

2、这些事务最近一个日志记录地址

**重新开始文件的内容**：

记录各个检查点记录在日志文件中的地址。

**动态维护日志文件的方法**：

**周期性的执行如下操作**：建立检查点，保存数据库状态。

**具体步骤**：

1. 将当前日志文件缓冲区中的所有日志文件记录写入磁盘的日志文件上。
2. 在日志文件中写入一个检查点记录
3. 将当前数据缓冲区的所有数据记录写入磁盘的数据库中
4. 把检查点记录在日志文件中的地址写入一个重新开始文件中。

**利用检查点的恢复策略**：

1. 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，有该地址在日志文件中找到最后一个检查点记录。
2. 由该检查点记录得到检查点建立时刻所有正在执行的事务清单ACTIVE—LIST，然后建立一个撤销和重做队列。把ACTIVE-LIST暂时放入撤销队列，重做队列暂时为空
3. 从检查点开始正向扫描日志文件，直到日志文件结束。如有新开始的事务T，把T暂时放入撤销队列。如有提交事务，那么把它放入重做队列中
4. 对撤销队列中的事务撤销，对重做队列的事务重做。

**数据库镜像**：介质故障恢复比较耗时，提高数据库可用性。

**数据库镜像**：数据库管理系统自动把整个数据库或其中的关键数据复制到另一个磁盘上，数据库管理系统自动保证镜像数据与主数据的一致性，每当主数据库更新时，数据库管理系统自动把更新后的数据复制过去。

**当出现介质故障时**：1、可由镜像磁盘继续提供使用，同时数据库管理系统自动利用吉祥磁盘数据进行数据库的恢复。

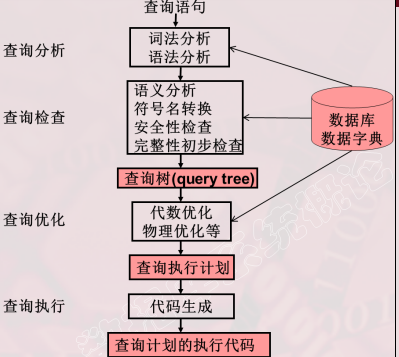
**没有出现故障时**：可用于并发操作。一个用户对数据加排它锁修改数据，其它用户可以读镜像数据库上的数据，而不必等待该用户释放锁。

在实际应用中用户往往只选择对关键数据和日志文件镜像。

#### 关系查询处理和查询优化

查询处理步骤：

1. 查询分析：对查询语句进行扫描、词法分析和语法分析
2. 查询检查：语义检查、视图转换、安全性检查和完整性初步检查
3. 查询优化：代数优化和物理优化
4. 查询执行。



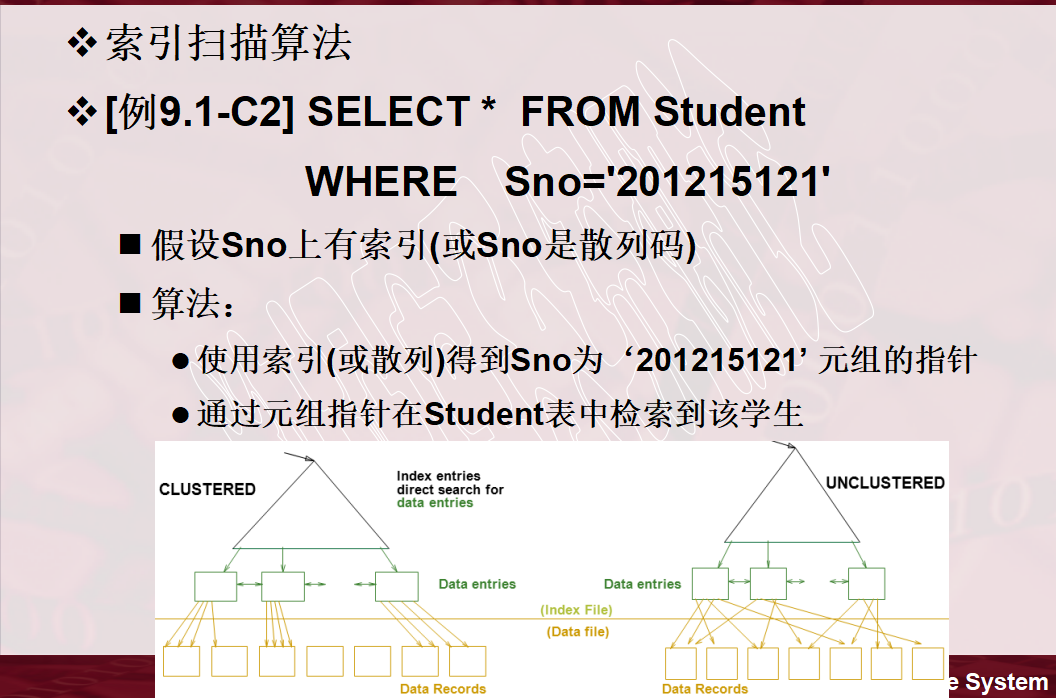
全表扫描算法：

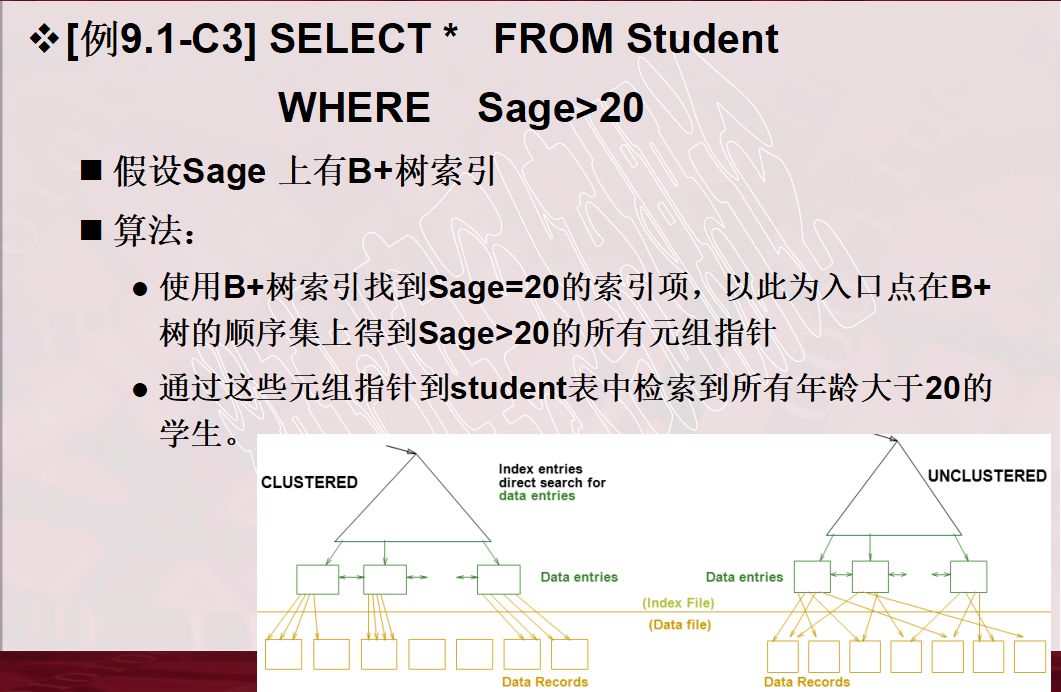
1、假设可以使用的内存块为M块，按照物理次序读关系表的M块到内存

2、检查内存的每个元组，如果满足条件则输出

3、否则，重复1和2步骤

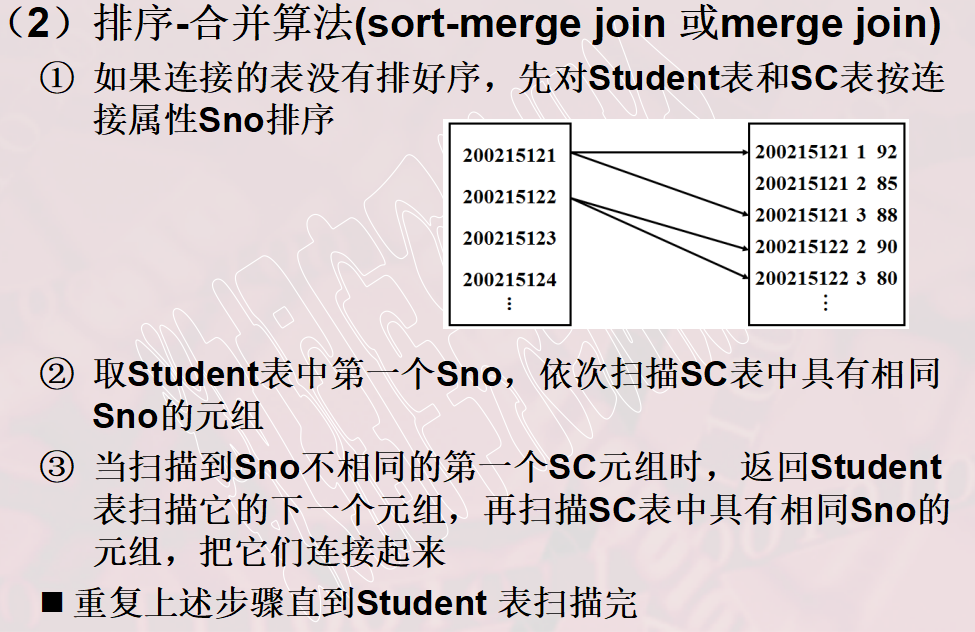
索引扫描算法：索引方法的得到元组指针，通过元组指针检索。



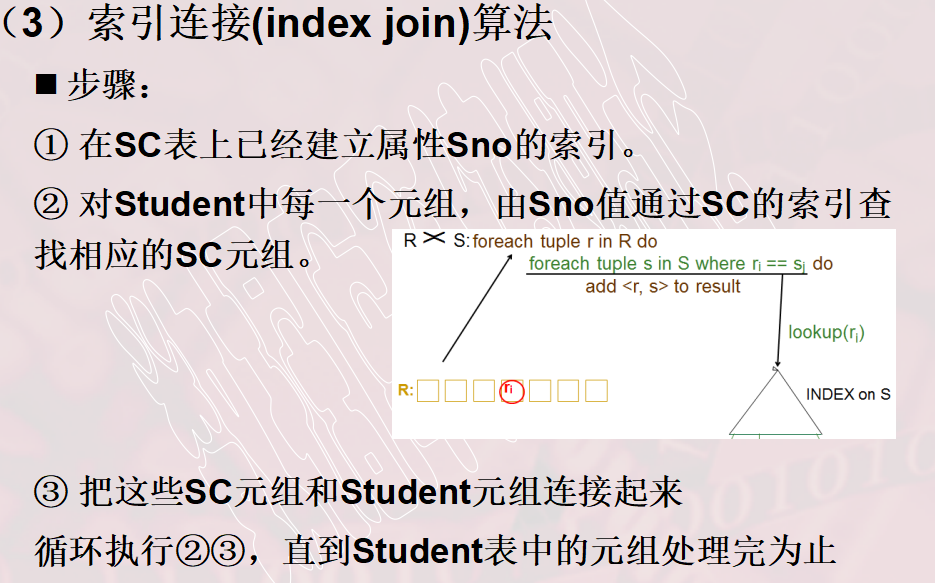


连接操作的实现：

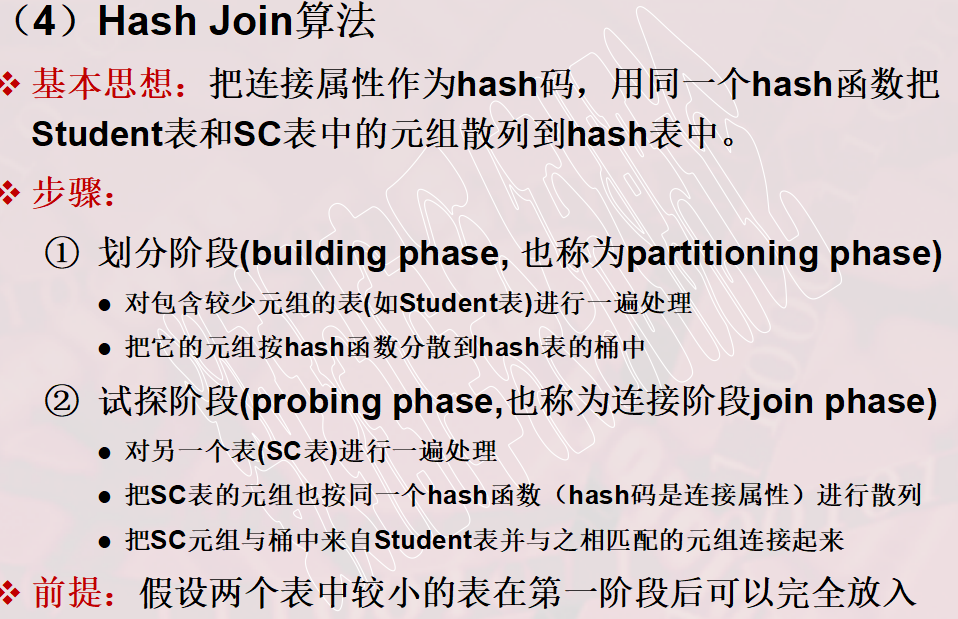
1. 循环嵌套算法：对外层循环的每一个元组，检索内层循环中的每一个元组。
2. 排序-合并算法：



1. 索引连接算法：



1. Hash Join算法



#### 查询代价：

1. 集中式数据库：磁盘存取块数（I/O代价）、CPU代价、查询的内存开销。
2. 分布式数据库：除了上述代价还包括通信代价

代数优化：

1. 关系代数表达式等价变换规则
2. 查询树的启发式优化。

物理优化：选择高效合理的操作算法或存取路径，求得优化的查询计划。

1. 基于启发式规则的存取路径选择优化
2. 基于代价的优化

基于启发式规则的存取路径选择优化：

1. 选择操作的启发式规则：对于小关系使用全表顺序扫描，即使选择列上有索引；对于大关系，启发式规则有

* 对于选择条件为“主码=值”的查询：查询结果最多是一个原则，可以选择主码索引。
* 对于选择条件为“非主属性=值”的查询，并且选择列上有索引，或者
* 对于选择条件是属性上的非等值查询或者范围查询，并且选择列上有索引

那么要估算查询结果的元组数目：如果比例较小（<10%），可以使用索引扫，否则使用全表顺序扫描

* 对于用ANS来连接的合取选择条件：如果有涉及这些属性的组合索引，优先采用组合索引扫描方法。如果某些属性上有一般索引，可以用索引扫描方法。
* 对于用OR来凝结的析取选择条件，一般使用全表顺序扫描。

1. 连接操作的启发式规则

* 如果2个表都已经按照连接属性排序，选择排序-合并算法
* 如果一个表在连接属性上有索引，使用索引连接方法
* 如果上面2个规则都不适用，其中一个表较小，可以使用Hash join或者嵌套循环方法。

基于代价的优化：统计信息，根据统计信息进行代价估算。

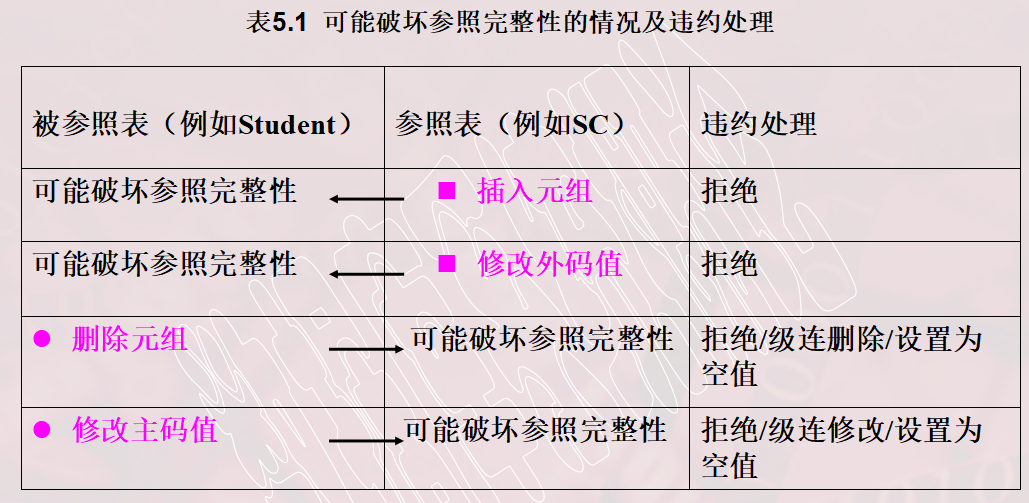
#### 数据库完整性

数据库的完整性：防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。

实体完整性：主码值唯一并且主码的各个属性不能为空。

检查实体完整性中主码值是否唯一的一种方法是进行全表扫描，但耗时，一般采用的方法是在主码上建立一个索引（B+树索引）

参照完整性检查：对参照表和被参照表进行增删改操作时有可能破坏参照完整性，必须检查。



对于参照完整性，除了定义外码，还应定义外码列是否允许空值。

用户定义的完整性：

1. 属性上的约束条件:拒绝处理
2. 元组上的约束条件：拒绝处理。

完整性约束语句：对约束语句用一个变量代替，方便以后对约束语句的操作处理。

断言：SQL中，可以使用CREATE ASSERTION语句，通过声明性断言来指定更具一般性的约束。可以定义涉及多个表的或聚集操作的比较复杂的完整性约束。如果违背断言，则拒绝操作。

触发器：是用户定义在关系表上的一类由事件驱动的特殊过程。

1. 触发器保存在数据库服务器中
2. 任何用户对表的增、删、改操作均有服务器自动激活相应的触发器。
3. 触发器可以实施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。

触发器又叫做事件-条件-动作规则。当特定的系统事件发生时，对规则的条件进行检查，如果条件成立则执行规则中动作，否则不执行该动作。

触发器类型：

1. 行级触发器(for each row)
2. 语句级触发器(for each statement)

触发动作体：

如果是行级触发器，可以在过程体中使用new和old引用事件之后的新值和事件之前的就职。

如果是语句级触发器，则不能在触发动作体是使用new 或old进行引用。

#### 数据库设计

数据库设计的6个阶段：

1. 需求分析
2. 概念结构设计
3. 逻辑结构是设计
4. 物理结构设计
5. 数据库实施
6. 数据库运行和维护

数据字典：数据字典是关于数据库中数据的描述，即元数据，不是数据本身。数据字典在需求分析阶段建立，在数据库设计过程中不断修改、充实、完善。数据字典是进行详细的数据收集和数据分析所获得的主要结果。

数据字典的内容：

1. 数据项：数据项是最小数据单元
2. 数据结构：数据结构反映了数据之间的组合关系。
3. 数据流：是数据结构在系统内传输的路径
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。
5. 处理过程：处理过程的具体逻辑一般用判定表或判定树来描述。数据字典中只需要描述处理过程的说明性信息。

概念结构设计：E-R模型

1. R图提供了表示实体型、属性和联系的方法。

实体与属性划分原则：

为了简化E-R图的处置，现实世界的事物能作为属性对待的，尽量作为属性对待。

两条准则：

1. 作为属性，不能再具有需要描述的性质。属性必须是不可分的数据项，不能包含其他属性。
2. 属性不能与其他实体具有联系。

#### 数据库的安全性

数据库的安全性：保护数据库以防止不合法使用所造成的数据泄露，更改或破坏。

数据库安全性控制的常用方法：

1. 用户标识和鉴定

系统提供的最外层安全保护措施，由用户名和用户标识号组成

1. 存取控制

定义用户权限，并将用户权限登记到数据字典中。常用方法：

* 自主存取控制：授予用户权限
* 强制存取控制：授予对象加密级别，每个用户授予某个级别的许可证，对于任意一个对象，具有合法许可证的用户才可以存取。仅当主体的许可证级别大于或等于客体的密级时，主体才能读相应的客体。仅当主题的许可证级别小于或等于客体的密级时，该主体才能写相应的客体。

1. 视图：把要保密的数据对无权存取这些数据的用户隐藏起来，对数据提供一定程度的安全保护；并且间接地实现支持存取谓词的用户权限定义。
2. 审计：启用一个专用的审计日志，记录用户对数据库的所有操作。
3. 数据加密：防止数据库中数据在存储和传输中失密的有效手段。分为存储加密和传输加密。

* 存储加密：

透明存储加密：内核级别的加密保护方式，将数据写到磁盘时对数据加密，读数据时解密。

非透明存储加密：通过多个加密函数实现。

* 传输加密：

链路加密：在链路层加密，报文和报头加密

端到端加密：在发送端加密，接收端解密。只加密报文不加密报头。

数据库角色：被命名的一组与数据库操作相关的权限。角色是权限的集合，可以为一组具有相同权限的用户创建一个角色。

一个角色的权限：直接授予这个角色的全部权限加上其他角色授予这个角色的全部权限。

#### 关系数据库

超码：若关系中的某一属性组的值能唯一地标识关系中元组，则称该属性组为超码。

候选码：最小化的超码（即任何真子集都不是超码）

全码：关系模式的所有属性组成这个关系模式的候选码，称为全码。

主属性：候选码的属性称为主属性。

主码：若一个关系中有多个候选码，则选定其中一个为主码。

三类关系

1. 基本关系：实际存在的表，是实际存储数据的逻辑表示。
2. 查询表：查询结果对应的表。
3. 视图表：有基本表或其他视图导出的表，是虚表，不对应实际存储的数据。

关系模式：是对关系的描述，是型

#### 关系数据库理论

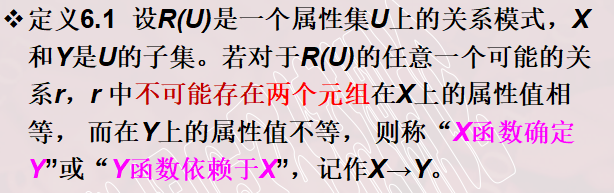
数据依赖：

1. 是一个关系内部属性与属性之间的一种约束关系。
2. 是现实世界属性间相互联系的抽象
3. 是数据内在的性质
4. 是语义的体现。

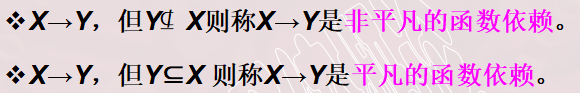
数据依赖的主要类型：

1. 函数依赖
2. 多值依赖

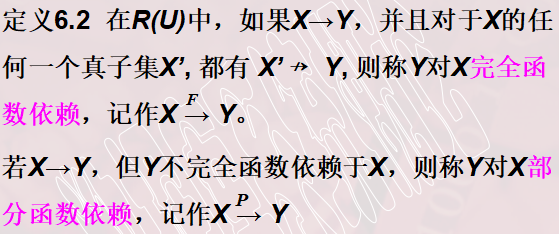
函数依赖：



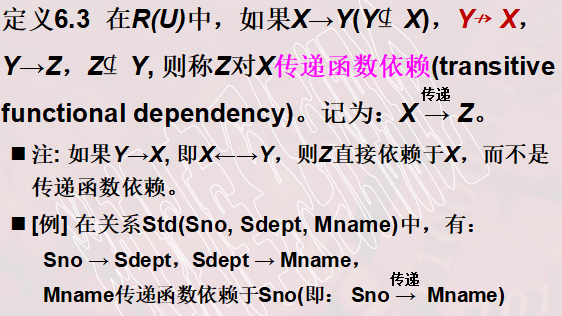
平凡函数依赖与非平凡函数依赖：



完全函数依赖与部分函数依赖：



传递函数依赖：



**规范化**：一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化。

**第二范式：**



不符合第二范式：

1. 插入异常
2. 删除异常
3. 修改复杂

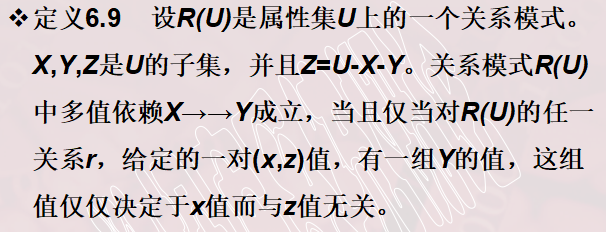
**第三范式**：关系模式在满足第一范式的情况下，非主属性不存在对主属性的传递依赖。

BCNF：如果一个关系数据库中的所有关系模式都属于BCNF，那么在函数依赖范畴内，它已经实现了模式的彻底分解，达到了最高的规范化程度，消除了插入异常和删除异常。

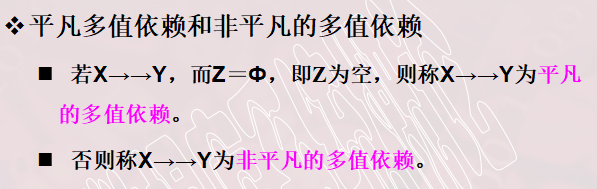
1. 所有的非主属性都完全函数依赖于每个候选码
2. 所有主属性都完全函数依赖与每个不包含它的候选码。
3. 没有任何属性完全函数依赖于非码的任何一组属性。

3NF的不彻底性表现在可能存在主属性对码的部分依赖和传递依赖。

多值依赖：



平凡的多值依赖和非平凡的多值依赖：



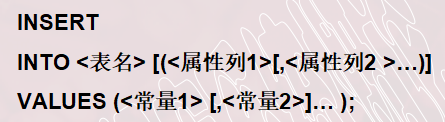
多值依赖的性质：

第四规范：

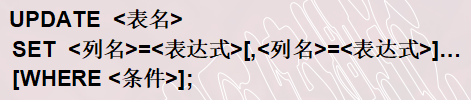


#### 关系数据库标准语言SQL

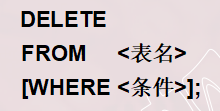
插入：



修改：



删除：

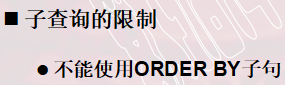
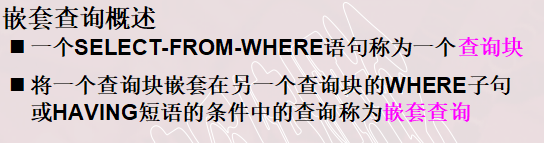


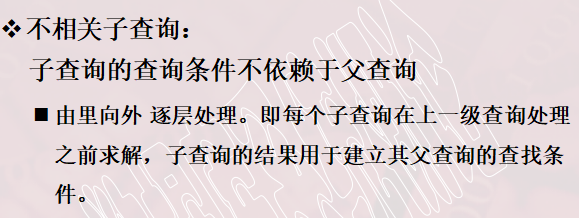
视图：

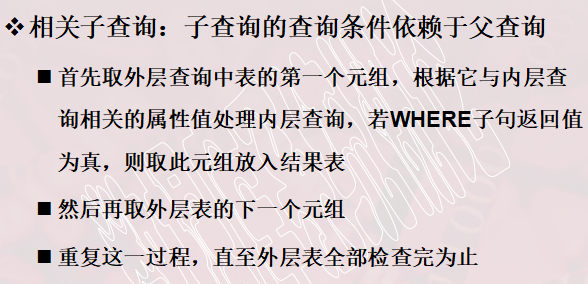
虚表，是从一个或几个基本表导出的表。基本表中的数据发生变化时，从视图中查询出的数据也随之改变。

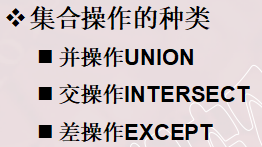
视图的作用：

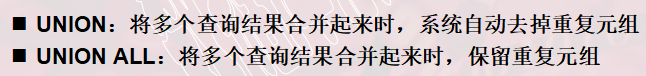
1. 视图能够简化用户的操作。
2. 视图使用户能以多种角度看待同一数据
3. 视图对重构数据库提供了一定程度的逻辑独立性
4. 视图能够对机密数据提供安全保护
5. 适当的利用视图可以更清晰的表达查询。

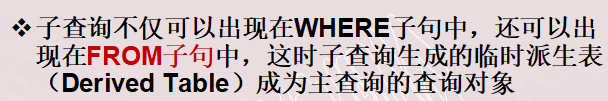


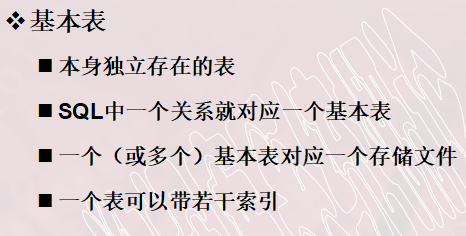


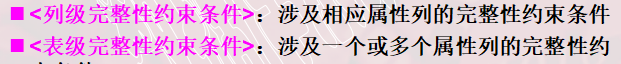


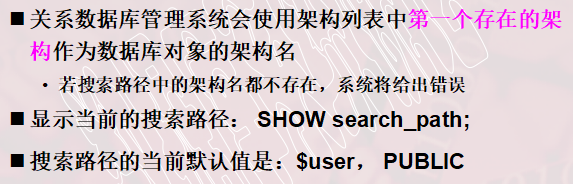


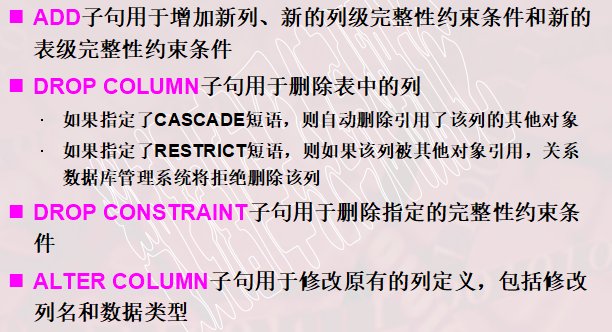












数据字典：

