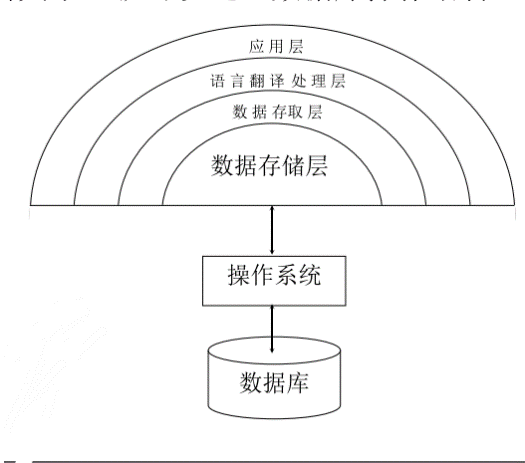
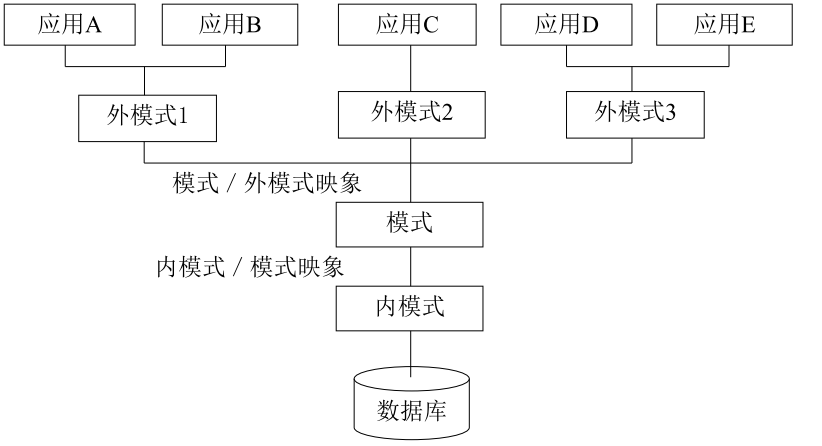
**Lecture 1**

* **关系数据库**：将将数据操作从具体的计算软件模型和物理存储模式中独立出来
  + 只需了解数据间的逻辑关系，就可以通过数据库操作语言来管理和分析数据
* **数据模型**：用来描述数据、组织数据和对数据进行操作的模型
  + 数据结构 – 静态特性：行列、键值对…
  + 数据操作 – 动态特性
  + 完整性约束 – 完整性约束条件：
    - All rows must have the same number of columns
    - All values in one column must have the same type
    - A child cannot have two parents
* **概念数据模型：**独立于计算机系统，主要用于数据库设计、与DBMS无关
* **逻辑数据模型(logical model)**：直接面向数据库的逻辑结构，通常有一组严格定义的，无二义性的语法和语义的数据库语言，可以用这种语言来定义、操纵数据库中的数据；
  + 与DBMS有关，用概念数据模型表示的数据必须转化为逻辑数据模型表示的数据，才能在DBMS中实现
  + 既要面向用户，也要面向实现
  + 分类：
    - 层次模型 (Hierarchical Model)
      * 条件：
        + 有且只有一个结点没有双亲结点，这个结点称为根结点
        + 根以外的其它结点有且只有一个双亲结点
      * 特点：
        + 只能直接处理一对多的实体联系
        + 任何记录值只有按其路径查看时，才能显出它的全部意义，没有一个子女记录值能够脱离双亲记录值而独立存在
      * 数据操作：查询、插入、删除、更新
      * 完整性约束：
        + 无相应的双亲结点值就不能插入子女结点值
        + 如果删除双亲结点值，则相应的子女结点值也被同时删除
      * 优点：
        + 模型简单，对具有一对多的层次关系的部门描述自然、直观，容易理解
        + 性能优于关系模型，不低于网状模型
        + 提供了良好的完整性支持
      * 缺点：
        + 多对多联系表示不自然
        + 对插入和删除操作的限制多
        + 查询子女结点必须通过双亲结点
        + 层次命令趋于程序化
    - 网状模型 (Network Model)
      * 数据操作：查询、插入、删除、更新
      * 优点：
        + 能够更直接地描述现实世界，特别是一个结点可以有多个双亲
        + 具有良好的性能，存取效率较高
      * 缺点：
        + 结构比较复杂，而且随着应用环境的扩大，数据库的结构就变得越来越复杂，不利于最终用户掌握
        + DDL(Data Definition Language)、DML(Data Manipulation Language)语言复杂，用户不容易使用
      * 网状模型与层次模型的区别
        + 网状模型允许多个结点没有双亲结点
        + 网状模型允许结点有多个双亲结点
        + 网状模型允许两个结点之间有多种联系（复合联系）
        + 网状模型可以更直接地去描述现实世界、层次模型实际上是网状模型的一个特例
    - 关系模型 (Relational Model) ——结构化数据
      * 优点：
        + 简单：一个数据库由多个关系组成，每一个关系就是一个规范化了的二维表，关系模型中的许多概念与二维表是一一对应的
        + 易访问：可以使用高级的数据查询语言构造出复杂的查询，对数据库中的数据进行访问
      * RDBMS：关系数据库管理系统- Oracle，MySQL，MS SQL Server，PostgreSQL，DB2等
      * 三级模式结构：



* + - * 模式（逻辑模式）：用逻辑数据模型对数据库中全部数据的逻辑结构和特性的描述，是数据库所有用户的公共数据视图
        + 一个数据库只有一个模式
        + 定义模式时要定义：

数据的逻辑结构（数据项的名字、类型、取值范围等）

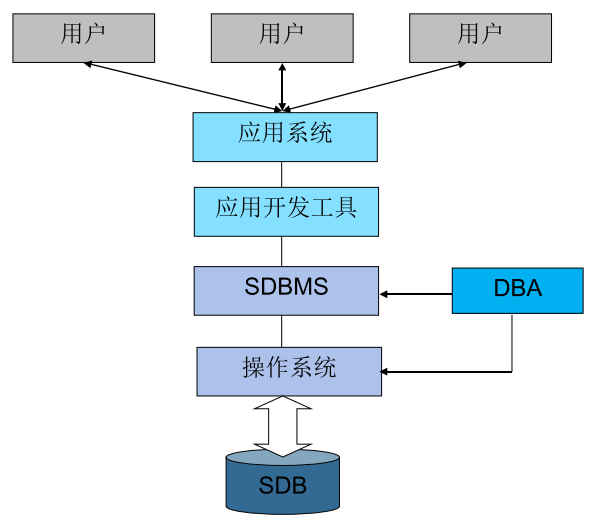
数据之间的联系

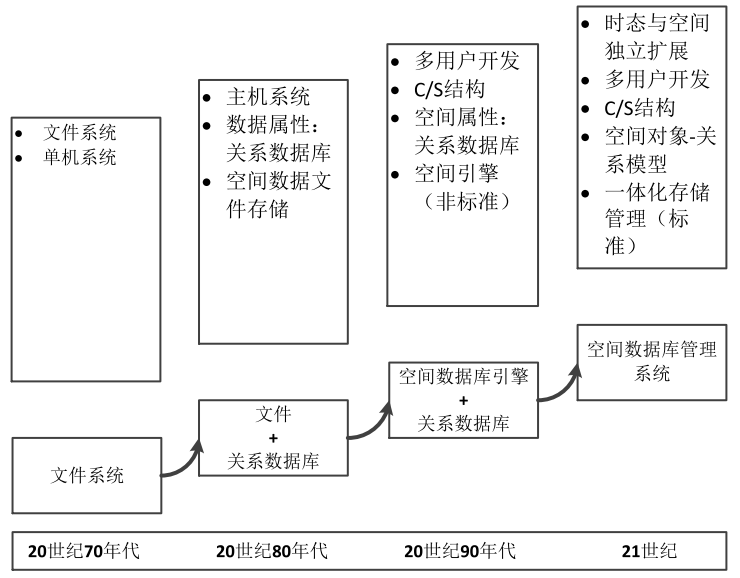
数据有关的安全性、完整性要求

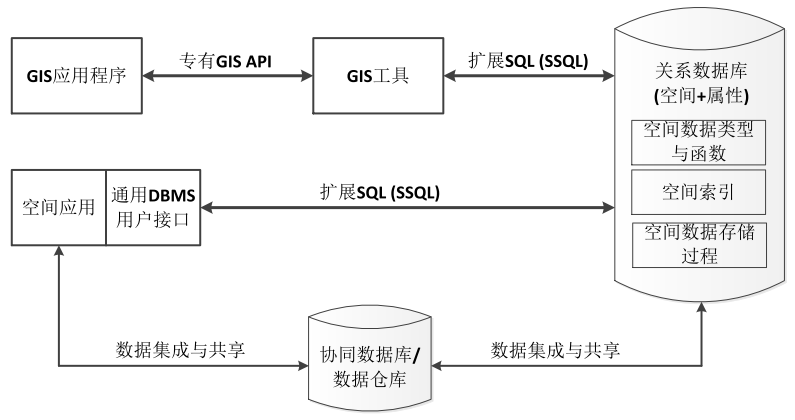
* + - * 外模式（子模式或用户模式）：对用户所用到的那部分数据的描述（不同用户的外模式不一定相同）
        + 一个数据库可以有多个外模式
        + 外模式是模式的一部分或是从模式推导而来的
      * 内模式（存储模式）：数据物理结构和存储方式的描述，是数据在数据库内部的表现方式（是B+树结构存储还是hash方法存储，是否压缩、是否建立索引…）
        + 一个数据库只有一个内模式
      * 外模式/模式映象：定义外模式与模式之间的对应关系，每一个外模式都对应一个外模式/模式映象
        + 用途：保证数据的**逻辑独立性**

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

* + - 面向对象模型 (Object Oriented Model)
    - 对象关系模型 (Object Relational Model)
* **物理数据模型(physical model)**：是对数据最底层的抽象，描述数据在磁盘或磁带上的存储方式和存取方法，面向计算机系统
  + 每种逻辑数据模型在实现时，都有其对应的物理数据模型
  + 物理数据模型的实现不仅与DBMS有关，还与操作系统和硬件有关
  + 物理数据模型的设计是在操作效率和系统复杂性之间的一个平衡
* 3 levels of database design：
  + Conceptual model: high level abstract description
  + Logical model: description of a concrete realization
  + Physical model: implementation using basic components
* 空间数据库：以空间目标作为存储对象的专业数据库
* 空间数据：分为矢量数据和栅格数据
* 矢量数据：用点、线、面等基本空间要素来表示人们赖以生存的自然世界的数据
* 空间实体：不可再分的最小单元现象，一般是矢量数据：
  + 空间实体本身的空间位置及属性信息
  + 空间实体相互之间的关系——空间关系 -> 拓扑关系
* 空间数据的特征：
  + 空间特征：
    - 空间对象隐含了空间分布特征
    - 空间数据组织需要考虑它的空间分布特征
    - 除了属性索引外，还需要建立空间索引
  + 非结构化特征：
    - 关系数据库中数据记录是结构化的
    - 结构化数据：满足关系模式的范式基本要求，可以用二维表结构来逻辑表达的数据
    - 非结构化数据：不方便用数据库二维逻辑表来表现的数据，包括文本、图片、XML、HTML、音频、视频
      * 空间数据是一种非结构化数据：
        + 空间实体是不定长的，例如一条弧段可能包含两对坐标点，也有可能10万对坐标点
        + 空间实体是非原子的，有的甚至是嵌套的，例如一个多边形可能包含多条弧段
        + 通用的关系数据库管理系统难以直接管理空间数据
  + 空间关系特征
    - 空间数据包括空间坐标和拓扑关系
      * 方便空间数据的查询和空间分析(几何对象模型和空间网络模型) 
      * 但给空间数据的一致性和完整性维护增加了复杂性
  + 时态特征
    - 反映地理实体的状态和演变过程的重要组成部分
    - 现有的SDB只是描述数据的瞬时状态，基本不具有管理空间数据的时间动态性
  + 多尺度特征：
    - 空间多尺度是指根据地学过程或地理地球系统中各部分规模的大小，可分为不同的层次
    - 时间多尺度是指地学过程或地理特征具有一定的自然节律性，其时间周期长短不一
* 空间数据库的要求：
  + 按一定的数据模型组织、描述和存储，具有较小的冗余度、较高的数据独立性和易扩展性，并可为各种用户共享
* 空间数据库三大要素：
  + 空间数据（Spatial Data Type）
  + 空间分析（Spatial Analysis）
  + 空间索引（Spatial Indexing）
* 空间数据库的特点
  + 数据量大（要求在二维空间上划分块或图幅、在垂直方向上划分层来进行组织）
  + 空间数据与属性数据集合
  + 应用广泛
* 空间数据库管理系统（**SDBMS**）：由空间数据库及其管理软件、应用软件组成，是存储介质、处理对象和管理系统的集合体
  + 功能：
    - 空间数据的定义与操纵（SDDL，SDML）
    - 空间数据的组织、存储和管理（存取效率）
    - 后台的事务管理和运行管理（DB系统管理员）
    - 数据库的建立与维护
  + 组成部分
    - 空间数据库
    - 空间数据库管理系统
    - 数据库管理员
    - 用户和应用程序
* 空间数据管理技术的发展



* 混合管理系统（文件+关系数据库）
  + 文件系统管理几何图形数据
  + 关系数据库管理属性数据
  + 两者之间通过对象目标的唯一标识符OID实现联系
* 对象关系型数据库管理系统：
  + 支持SQL，具有良好的通用性
  + 具有面向对象特性，支持复杂对象及其行为
  + 能够直接存储和管理非结构化的空间数据
  + ORDBMS提供：
    - 类和继承
    - 用户自定义类型、函数、索引和规则

**Lecture 2**

* 关系模型：一种数据模型，和一般的数据模型一样，由三部分组成：
  + 关系数据结构
    - 单一的数据结构——关系
    - 数据的逻辑结构——二维表：从用户角度，关系模型中数据的逻辑结构是一张二维表
    - 关系模型采用关系（Relation）作为数据结构，直观地讲，关系就是简单的表（Table）。一个表一般由表名、表头和数据三部分构成
  + 关系操作集合
  + 关系完整性约束
* 域（Domain）：一组具有相同数据类型的值的集合
* 给定一组域D1，D2，…，Dn，D1，D2，… ，Dn的笛卡尔积(Cartesian Product)为：  
   *D1×D2×…×Dn ＝ {(d1，d2，…，dn) ｜diDi，i＝1，2，…，n}*
  + 称其中每一个元素(d1，d2，…，dn)叫做一个n元组（N-Tuple）或简称元组（Tuple）
  + 元素中每一个值di叫做一个分量
* 属性(Attribute)：关系中的一列即为一个属性 
* 域：属性的取值范围称为该属性的域 
* 码：可以唯一确定一个元组的最小属性集合称为候选码（Candidate Key），或简称为码(Key)
* 分量：元组中的一个属性值 
* 关系模式(Relation Schema)：对关系的描述，一般表示为：关系名（属性1，属性2，…，属性n）
* 关系实例( Relation Instance)：关系的内容
* 一个关系由关系名、关系模式和关系实例组成，分别对应于表名、表头和表中的数据。关系名和关系模式是相对变化的，关系实例会随时间而发生变化
* 关系的三种类型：
  + 基本关系（通常又称为基本表或基表），基本表是实际存在的表，是实际存储数据的逻辑表示
  + 查询表，是查询结果对应的表
  + 视图表，是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据
* 关系操作
  + 特点：
    - 集合操作模式：操作的对象和结果都是集合
    - 非关系数据模型的数据操作方式：一次一记录
  + 常用操作：
    - 查询：选择、投影、连接、除、并、交、差
    - 数据更新：插入、删除、修改
* 关系数据语言
  + 关系代数语言：用对关系的运算来表达查询要求
  + 关系演算语言：用谓词来表达查询要求
  + 介于关系代数和关系演算之间的语言，即结构化查询语言 SQL
* 关系模型中的三类完整性约束
  + 实体完整性：若属性A是基本关系R的主属性，则任何一个元组在属性A上不能取空值（Null）
  + 参照完整性：关系间应用时的完整性——外码
  + 用户定义的完整性
* 关系代数：
  + 一种抽象的查询语言
  + 用对关系的运算来表达查询
* 关系代数运算的三个要素
  + 运算对象：关系
  + 运算结果：关系
  + 运算符：四类：
    - 集合运算符：将关系看成元组的集合，运算是从关系的“行”方向来进行
      * 并、交、差、广义笛卡儿积
    - 专门的关系运算符：不仅设计行而且涉及列
      * 选择：一元运算，从关系中选择若干行
      * 投影：从关系中选出需要的列，形成新的结果关系

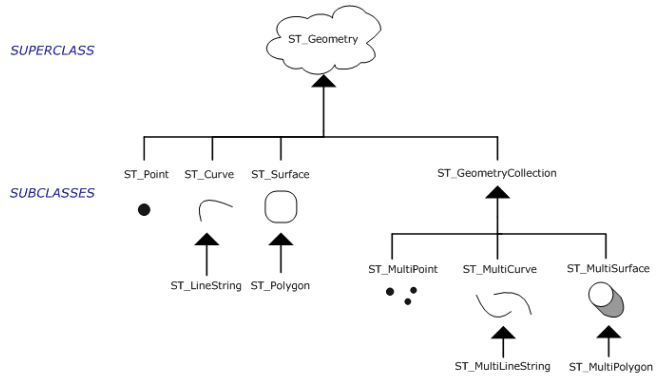
\*选择和投影不满足交换律

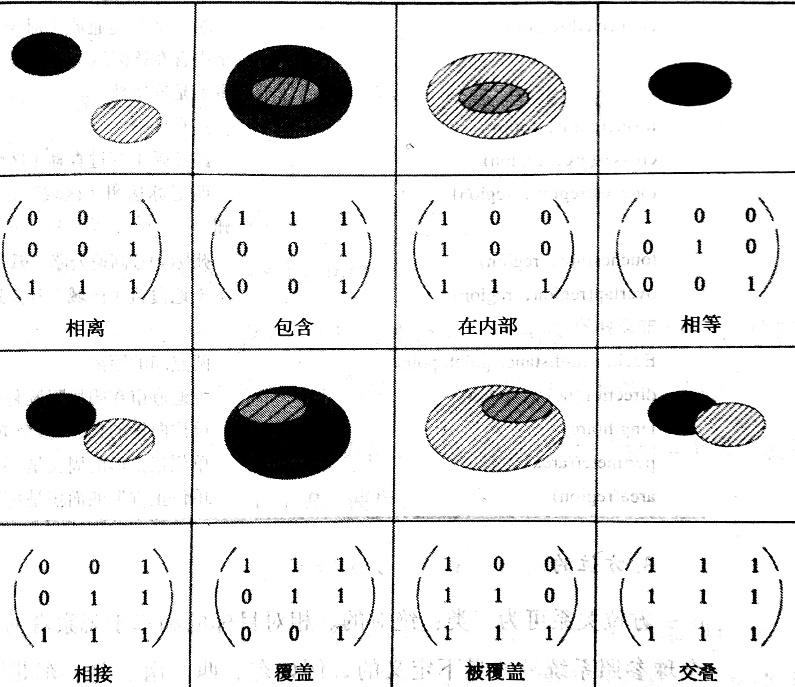
* + - * 连接：从两个关系的笛卡儿积中选取属性间瞒住一定条件的元组（AθB）
        + 等值连接：θ为”=”，即从关系R与S的广义笛卡儿积中选取A、B属性值相等的那些元组
        + 自然连接：是一种特殊的等值连接，要求关系R中的属性A和关系S中的属性B名字相同，并在结果中把重复的属性列去掉
      * 除
    - 算术比较符：辅助专门的关系运算符进行操作
    - 逻辑运算符：辅助专门的关系运算符进行操作

**Lecture ３**

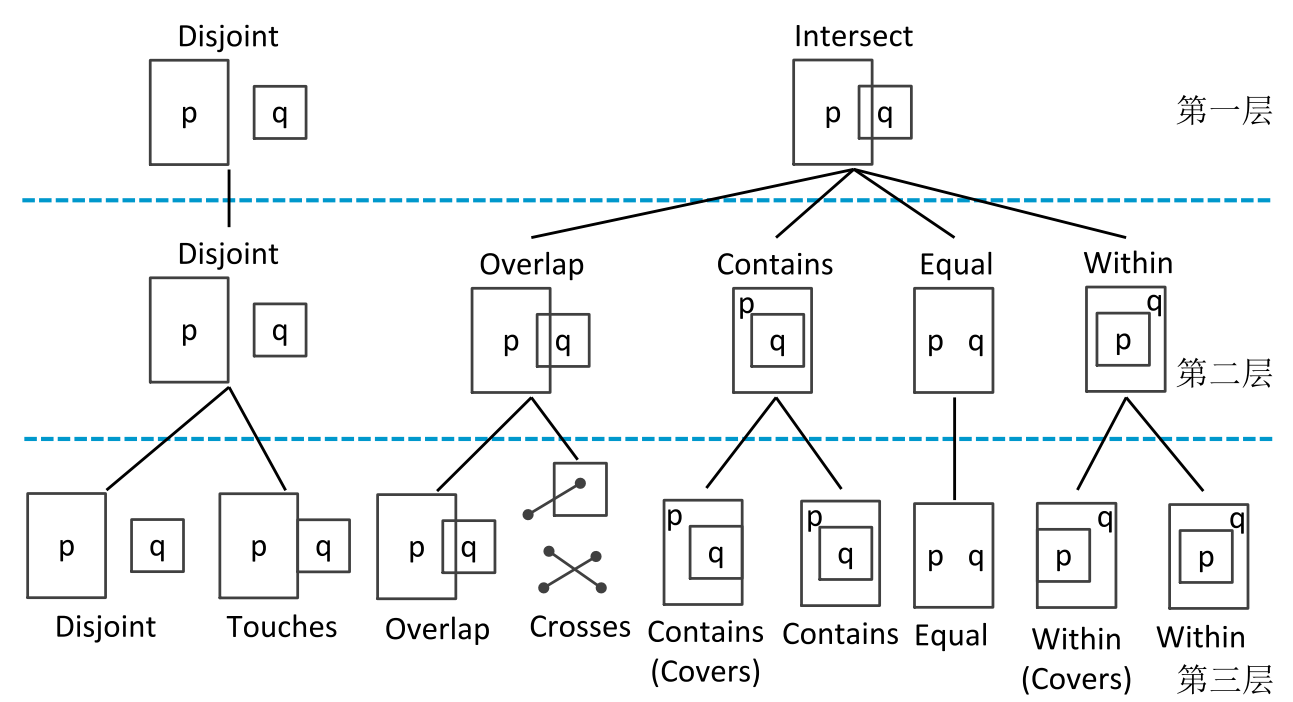
* SQL特点
  + 综合统一：
    - 集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动
    - 在关系模型中实体和实体间的联系均用关系表示，数据结构单一性带来了数据操作符的统一，查找、插入、删除、更新等操作都只需一种操作符
  + 高度非过程化：
    - 非关系数据模型的数据操纵语言是面向过程的语言，在执行一项工作时必须描述“怎么做”
    - SQL语言是非过程语言，使用它进行数据库操作时，只须提出“做什么”，而无须指明“怎么做”
  + 面向集合的操作方式：
    - 非关系数据模型采用的是面向记录的操作方式，操作对象是一条记录
    - SQL语言采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合
  + 以同一种语法结构提供两种使用方式：
    - 既是自含式语言（直接独立地用于联机交互的使用方式，对数据库进行操作）
    - 又是嵌入式语言（嵌入到高级语言——C/C++, Java, Python, C#, JavaScript…中使用）
  + 语言简洁，易学易用
* 数据定义：
  + 定义基本表
  + 定义视图
  + 定义索引
* 完整性约束：
  + 实体完整性：PRIMARY KEY
  + 参照完整性：FOREIGN KEY
  + 用户定义完整性：NOT NULL, UNIQUE, DEFAULT, CHECK
  + 域约束：使用CREATE DOMAIN语句定义新的值域，在定义域时声明域的取值范围
* NULL值：
  + NULL参与的数值或布尔运算，结果都是NULL 
  + WHERE子句只有条件为true才保留这个记录 
  + HAVING子句只有条件为true才保留这个GROUP 
  + JOIN NULL != NULL 
  + GROUP BY NULL算一个GROUP 
  + NULL在ORDER BY时默认排序最前面，有语法可以 改变顺序
  + 对于AGGREGATE函数
    - 如果输入空集，COUNT返回0，其他任何函数返回NULL
    - 如果COUNT(\*)，NULL的记录参与计算，COUNT属性，NULL的记录忽略
    - 其他AGGREGATE函数，忽略NULL

**Lecture 4**

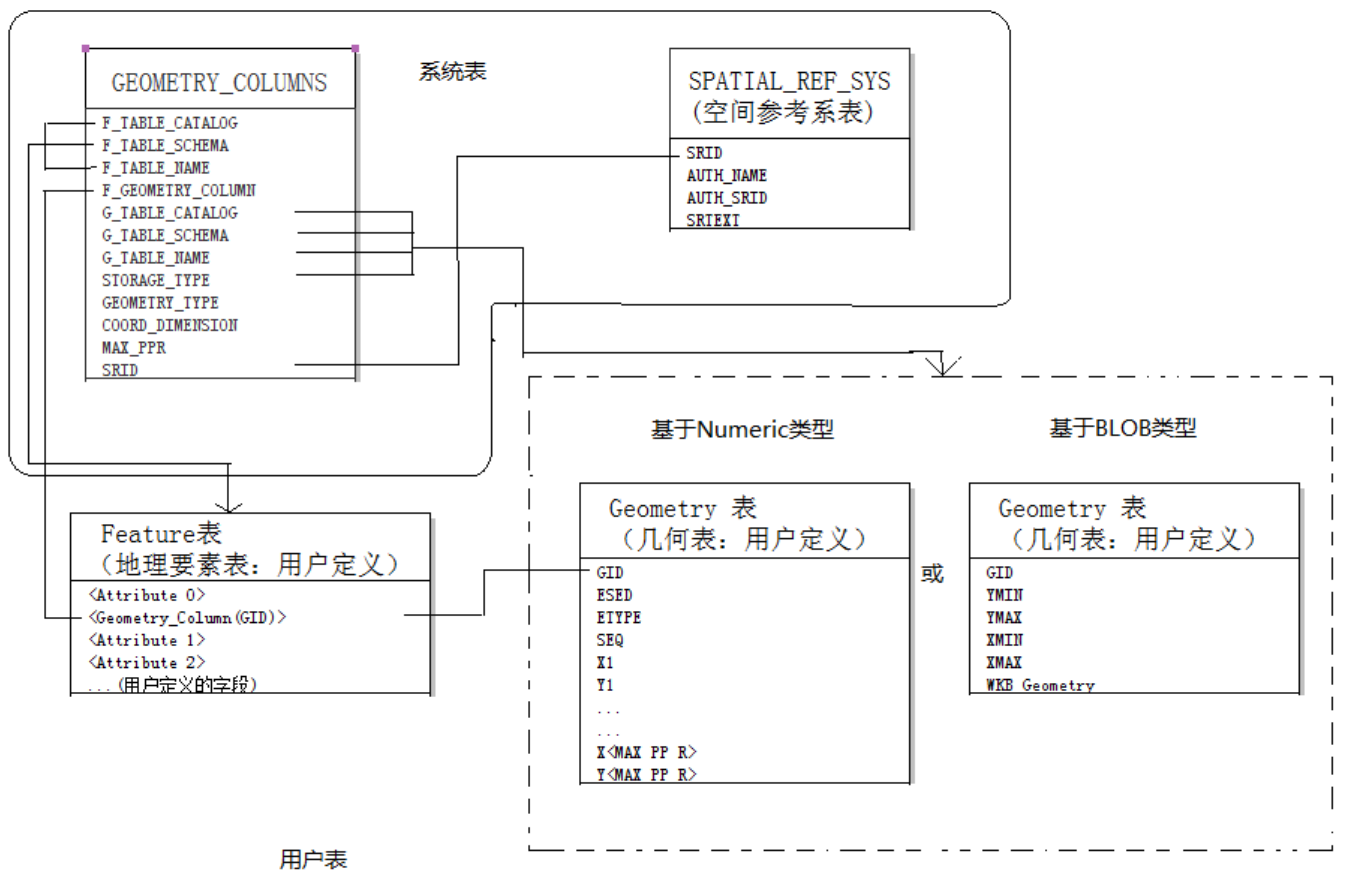
* 空间数据模型：是空间信息的一种数据组织方式，可以分为对象模型、场模型、网络模型
  + 矢量模型：利用点、线、多边形的形式来表达现实世界 
    - 具有定位明显，属性隐含的特点 
    - 不可再分的最小单元现象称为空间实体
    - 优点：
      * 数据结构紧凑，冗余度低，表达精度高，图形显示质量好，有利于网络和检索分析等
      * 在GIS中应用广泛，特别在小区域(大比例尺)制图中充分利用了它的精度高的优点
  + 栅格模型：以二维矩阵的形式来表示空间地物或现象分布的数据组织方式 
    - 每个矩阵单位称为一个栅格单元，单元值表示地物或现象的属性数据
    - 优缺点：
      * 数据结构简单、空间分析和地理现象的模拟较为容易等
      * 数据量大、投影转换比较困难
      * 随着遥感技术广泛应用，数据压缩技术、计算机性能的提高，栅格模型可能会在将来占主导
* 地理要素（feature）
  + 对现实世界空间现象的抽象
  + 由几何(geometry)、属性(attribute)、行为(behavior)等三类信息构成
  + 地理要素的属性和行为等信息的建模是由应用系统的设计者，根据实际应用需求进行建模
  + 几何的建模是数据库管理系统关心的基础问题：OGC（开放地理空间信息联盟）模型规范
* 几何对象模型：
  + 一个依赖于空间参考系(Spatial Reference System)和测量参考系(Measure Reference System)的几何 (Geometry)类
  + 派生出点(Point)、线(Curve)、面(Surface)、多点(MultiPoint)、多线(MultiCurve)、多面(MultiPolygon)等类型
    - 点(Point)
      * 零维几何对象类，代表空间中的一个点，如城市
    - 曲线(Curve)
      * 由点序列描述一维的几何对象类，如街道、管线
      * 相邻两点间的插值方法：线性插值和非线性插值
    - 折线(LineString)
      * 曲线的子类，采用线性插值
    - 线段(Line)
      * 折线的特例，只有两个点的线串
    - 环线(LineRing)
      * 由折线派生而来，闭合的、不自相交或相切的折线
    - 面(Surface)
      * 二维几何对象类，代表一个外边界、零到多个内边界组成的几何对象
      * 在三维空间中，可能是一个同构的曲面
    - 多边形(Polygon)
      * 二维坐标空间中由一个外边界、零到多个内边界定义的平坦表面，由一个或一个以上的线环聚合而成，如省份
      * 仅支持由折线串围成的多边形，暂不支持曲线
    - 体表面(PolyhedraSurface)
      * 由简单面沿着它们的边界“缝合”而成
      * 三维空间中的多面体曲面总体上可以不平坦
      * 相互接触的一对多边形的公共边可以表达为有限折线的集合
      * 由于体表面违反了“多边形元素只能相交在有限数量的点上”的规则，所以体表面不是多多边形
    - 三角形(Triangle)
      * 多边形类的一个特例
    - 不规则三角网(Triangulated irregular network, TIN)
      * 体表面的一个特例，由多个共享公共边的连续三角形聚合而成
    - 几何集合(GeometryCollection)
      * 由一个或多个几何对象组成的集合，其中的元素必须具有相同的空间参考系和测量参考系
      * “Mulit-“
* 坐标维数和集合维数的区别：
  + 坐标维数是指在一个坐标系统描述一个位置所需的测量或坐标轴的个数（空间维数） 
    - 现实世界是三维的，需要(x, y, z)三个坐标进行描述
  + 几何维度是在一定前提下描述一个几何对象所需的参数个数 
    - 点是0维，线是1维，面是2维，体是3维
* OGC仅能表达和处理简单(Simple)的几何对象——简单几何对象指不自相交的几何对象
* 任何几何模型都有其边界(boundary)、内部(interior)和外部(exterior)
  + 边界：一个几何实体界限的集合，几何维数是其本身几何形状的维数减一 
    - 点：本身（**内部为空**） 
    - 线：端点 
    - 曲线及其子类：起始点和终止点 
    - 多曲线及其子类：各曲线的起始点和终止点 
    - 面：构成它的线串
  + 内部：几何对象除边界外的所有直接位置(direct position)的集合 
    - 直接位置是用坐标参考系中的一组坐标描述的位置 
    - 几何维数与其本身的维数一致 
    - 所有几何对象都有内部，即几何对象形状减去其边界后的部分
  + 外部：空间全域与几何闭包之差 
    - 任意几何对象外部的维数总是2 
    - 所有的几何形状都有外部，即其几何形状的补集
* M值：线性参考系统中，除了x,y,z坐标外的另一个坐标（例：高速公路的里程碑点）
* 几何对象的方法
  + 常规方法：
    - Dimension() : Integer  用于获取几何对象的几何维数
    - CoordinateDimension() : Integer  用于获取几何对象的坐标维数
    - GeometryType() : String  用于获取几何的数据类型，如点、线、面等
    - SRID() : Integer  用于获取几何类型的空间参考系
    - Envelope() : Geometry  用于获取Geometry的最小边界矩形
    - AsText() : String  返回WKT (Well-Known Text)的表达形式，不包含SRID元数据
    - AsBinary() : String  返回WKB (Well-Known Binary)的表达形式，不包含SRID元数据
    - IsEmpty() : Boolean  判断几何类型是否为空
    - IsSimple() : Boolean  判断几何类型是否是简单的
    - Is3D() : Boolean  判断几何类型是否有z坐标
    - IsMeasured() : Boolean  判断几何类型是否有M值
    - Boundary() : Geometry  获取几何类型的边界
  + 常规GIS分析方法（空间分析）
    - Distance(another: Geometry) : Distance  求本Geometry与另一个Geometry间的距离
    - Buffer(distance: Distance) : Geometry  求本Geometry满足某个距离要求的缓冲区
    - ConvexHull() : Geometry  求本Geometry的凸包
    - Intersection(another : Geometry) : Geometry  求本Geometry与另一个Geometry的交
    - Union(another : Geometry) : Geometry  求本Geometry与另一个Geometry的并
    - Difference(another : Geometry) : Geometry  求本Geometry与另一个Geometry的差
    - SymDifference(another : Geometry) : Geometry  求本Geometry与另一个Geometry的对称差
  + 空间查询方法（拓扑分析）
    - Equals(another : Geometry) : Boolean  判断本Geometry与另一个Geometry是否相等
    - Disjoint(another : Geometry) : Boolean  判断本Geometry与另一个Geometry是否相离
    - Intersects(another : Geometry) : Boolean  判断本Geometry与另一个Geometry是否相交
    - Touches(another : Geometry) : Boolean  判断本Geometry与另一个Geometry是否相接
    - Crosses(another : Geometry) : Boolean  判断本Geometry是否穿越另一个Geometry
    - Within(another : Geometry) : Boolean  判断本Geometry是否包含于另一个Geometry
    - Contains(another : Geometry) : Boolean  判断本Geometry是否包含另一个Geometry
    - Overlaps(another : Geometry) : Boolean  判断本Geometry与另一个Geometry是否交叠
    - Relates(another : Geometry, matrix : String) : Boolean  判断本Geometry与另一个Geometry是否符合给定的９交矩阵
    - LocateAlong(mValue : Double) : Geometry  选取M值为mValue的点，形成一个新的Geometry
    - LocateBetween(mStart : Double, mEnd : Double) : Geometry  选取M值在mStart和mEnd之间的点，形成一个新的Geometry
* 九交模型



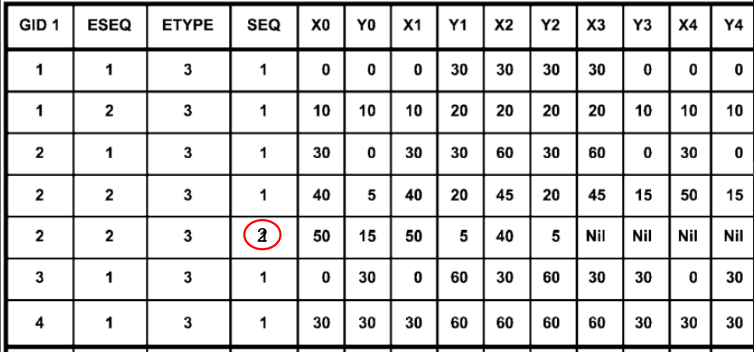
* 八种空间拓扑关系：



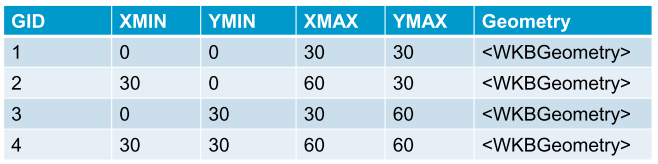
* + 相离(disjoint) ：若𝑎 ∩ 𝑏 = ∅，则a和b相离
  + 相交(intersects) ：若𝑎 ∩ 𝑏 ≠ ∅，则a和b相交  a.Intersects(b) 等价于 !a.Disjoint(b)
  + 相等(equals) ：若𝑎包含于𝑏，且𝑎包含𝑏，则a和b相等
  + 交叠(overlaps)：
    - 如Dim(𝐼 (a)) = Dim (𝐼 (b)) = Dim(𝐼 (𝑎) ∩ 𝐼(𝑏))，且𝑎 ∩ 𝑏 ≠ 𝑎，同时 𝑎 ∩ 𝑏 ≠ 𝑏，则a和b交叠
    - 点/点、线/线、面/面交叠关系，没有线/面等交叠关系
  + 包含于(within)：若𝑎 ∩ 𝑏 = 𝑎，且I 𝑎 ∩ 𝐸 𝑏 = ∅，则a包含于b内
  + 包含(contains)：若b包含于a，则a包含b  a.Contains(b) 等价于 b.Within(a)
  + 相接(touches)：
    - 若I 𝑎 ∩ 𝐼 𝑏 = ∅，且𝑎 ∩ 𝑏 ≠ ∅，则a和b相接 
    - 面/面、线/线、线/面、点/面、点/线相接关系 
    - 但没有点/点相接关系
  + 穿越(crosses) 
    - 若Dim(𝐼(𝑎)∩𝐼(𝑏)) < max(Dim(𝐼 (a)), Dim(𝐼 (b)))，且Dim(𝐼 (b)) != 0，𝑎 ∩ 𝑏 ≠ 𝑎，𝑎 ∩ 𝑏 ≠ 𝑏，则a穿越b
    - OGC定义：I 𝑎 ∩ 𝐼 𝑏 ≠ ∅ ， 𝑎 ∩ 𝑏 ≠ 𝑎， 𝑎 ∩ 𝑏 ≠ 𝑏 
    - 点/线、点/面、线/线、线/面穿越关系
* 线性参考系查询方法：
  + 交通地理信息系统（GIS-T）的两项关键技术：
    - 线性参考系统
    - 动态分段技术
  + 查询方法：
    - LocateAlong和LocateBetween用于动态构造线性要素的 各个部分的方法
    - LocateAlong用于选取几何中M值为mValue的点，并构造 成一个新的几何对象
    - LocateBetween用于选取几何中M值在mStart和mEnd之 间的点，并构造一个新的几何对象
* 几何对象逻辑模型:
  + 基于预定义数据类型的实现：



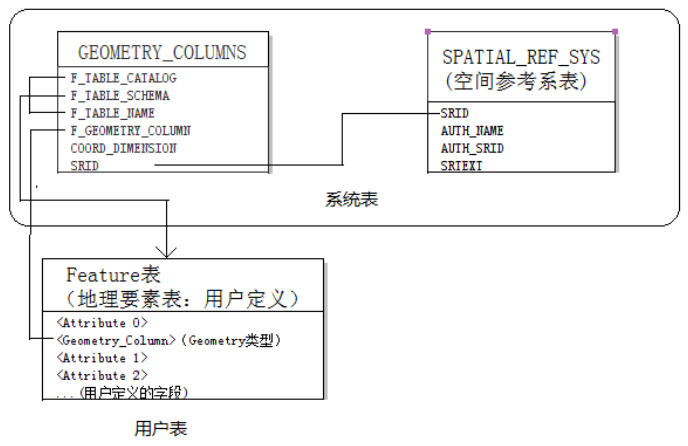
* + - 利用关系数据库中已有的数字(numeric)类型、二进制大对象(BLOB)类型实现空间数据的存储和管理
      * 基于numeric类型或BLOB类型来建立几何表
    - Feature表：
      * 记录一组具有相同属性和行为的地理要素的集合，要素表的列代表要素的属性，而不同的行代表不同的要素
      * Geometry\_Column列是几何对象的逻辑几何数据类型，其存储的是几何对象的唯一标识(geometry ID, GID)，而 几何数据实际存储在Geometry表中，因此，可以将GID 作为指针到Geometry表找到其空间数据
    - (基于Numeric类型的)Geometry表
      * 将几何类型的空间坐标作为数值对存储在表中，每行最多可存储MAX\_PPR个空间坐标
      * 若Geometry的空间坐标超过MAX\_PPR则折行存储
      * 其中的GID是Geometry对象的唯一标识
      * 在由多个元素组成的GeometryCollection中ESEQ用于表示不同的元素
      * SEQ则用于标识统一Geometry折行存储后的行序列号
      * ETYPE用于标识Geometry表中指定的几何对象的类型(如 ，点、线、面、多点等)



* + - (基于BLOB类型的)Geometry表
      * 将空间数据以WKB(Well-Known Binary Representation) 形式存储在名为WKB\_Geometry的BLOB类型的字段中
      * 不会出现折行存储Geometry的情况
      * 一行存储一个Geometry对象
      * 每行GID是该Geometry对象的唯一标识
      * YMIN, YMAX, XMIN, XMAX用于存储该对象的四至
      * 该表的主码是GID



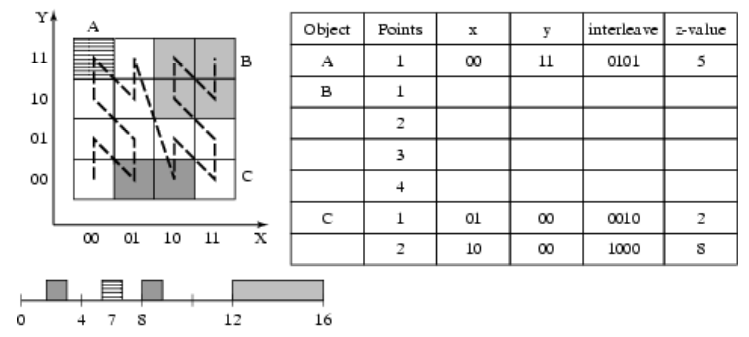
* + - GEOMETRY\_COLUMNS表
      * 记录数据库中所有要素表及其几何列的属性
      * 该表的前三列可以唯一标识Feature表
      * STORAGE\_TYPE表示Geometry表的存储类型
      * GEOMETRY\_TYPE表示Geometry表中几何对象的类型
      * COORD\_DIMENSION表示几何对象的坐标维数
      * MAX\_PPR表示每行存储的空间坐标点个数
      * SRID描述了Feature表的空间参考系
    - SPATIAL\_REF\_SYS表
      * 记录了该空间数据库所支持的所有空间参考系
      * SRID为主码:
        + 4326 – WGS 84 Long Lat 
        + 4269 – NAD 83 Long Lat 
        + 3395 – WGS 84 World Mercator 
        + 2163 – US National Atlas Equal Area
  + 基于扩展几何类型的实现



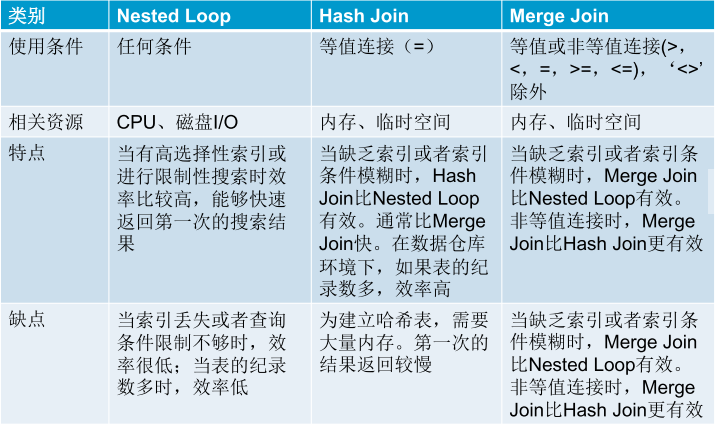
* + - GEOMETRY\_COLUMNS和SPATIAL\_REF\_SYS是 系统表，用于存储元数据信息
    - Feature表是用户表，用于存储空间数据
      * 属性数据存在用户定义的Attributes列中
      * 空间数据存放在扩展几何类型的Geometry\_Column中
    - 空间数据存放在扩展几何类型的Geometry\_Column
      * 从数据库底层来看，扩展几何类型还是存在BLOB字段中
      * 不同的是空间数据库可能根据扩展几何类型的定义，对该 BLOB数据进行解释，在SQL可以使用几何类型的相关函数
* 几何对象物理模型：
  + WKB表达：SFA SQL给出的一种较为紧凑的几何数据的二进制方式的存储格式
  + WKT表达：OGC提供的一种基于文本格式几何数据交汇标准表达方式
    - 例如POINT(10, 10)，POLYGON((10 10, 10 20, 20 20, 20 15, 10 10))
    - OGC也为空间参考系提供了投影、地理和地心三种坐标系的不同WKT表达形式
  + KT和WKB只支持二维几何对象的表达，并没有任何有关空间参考系的信息

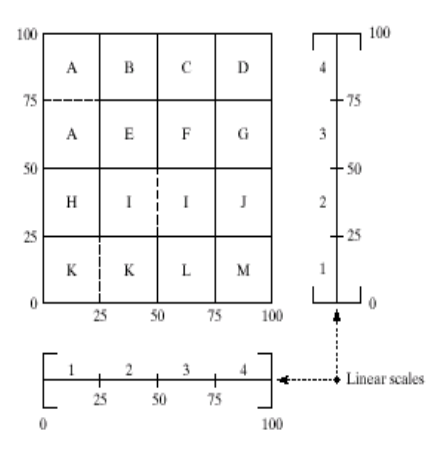
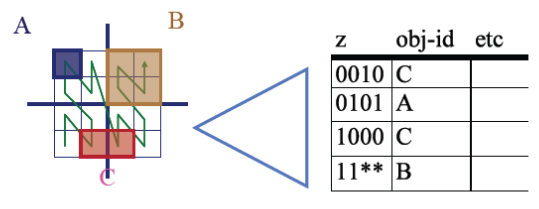
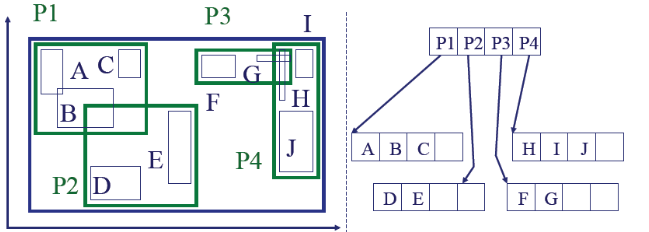
**Lecture 5**

* 在关系数据库管理系统的物理模型的基础上，通过引入空间填充曲线来对点进行排序，从而实现使用循序文件和搜索树的可能性
* 常见的空间查询：
  + 点查询（Point query）：找出包含给定点的矩形
  + 范围查询（Range query）：找出所有位于给定矩形中的点
  + 最邻近查询（Nearest neighbor）：找出距离查询点最近的点
  + 交叉查询（Intersection query）：找出所有与给定矩形相交的矩形
* 存储设备：
  + 主存（Main mamories）：很快，但存储内容在断电后会丢失
  + 二级存储设备（Secondary storage）：慢一些，可以在断电后保存数据（eg. 磁盘驱动器）
  + 三级存储设备（Tertiary storage）：最慢，可以断电保存，极大的容量
* 在DBMS中：
  + 在二级存储设备中管理数据
  + 使用主存来改善操作的表现
  + 使用三级存储设备进行备份、存档
* 磁盘结构：
  + 一系列圆形磁盘片放置在主轴上并旋转
  + 磁道：圆盘片上向边缘延伸的同心圆环
  + 柱面（Cylinder）：相同直径的磁道的集合
  + 扇区：每一个磁道均由驱动器厂商规定划分为整数倍个扇区
  + 磁盘块（或简称为页面）是磁盘与主存之间的最小传输单元
* 磁盘读取：
  + 寻道时间*ts*：磁头到达特定磁道所用的时间
  + 延迟时间*tl*：要获取的磁盘块旋转到磁头下方所用时间
  + 传输时间*tt*：置于正确位置后磁头读或写块中数据的实际时间
  + *ts* > *tl* > *tt*
  + 磁盘读取的速度主要受到：
    - 数据在磁盘上存储的位置和方式：
      * 最常用的数据应当存储在中间的磁道上，以最小化寻道时间
      * 较大数据集应当存储在同一个柱面的扇区中，以最小化扫描整组数据的时间
    - 寻道时间
    - 扇区的尺寸：
      * 更大的扇区使读写较大数据集时速度更快
      * 但是在存储较小数据集时会造成存储空间的浪费
* 软件视角的数据存储：
  + Field：关系或实体的属性
  + Records：表中关系模式的属性字段的集合
    - 比一个扇区（页面）要小；一个扇区（页面）是槽（slot）的集合，存储有多条记录（每个槽包含一条记录）
  + Files：records的集合
    - 一个文件可能会跨越多个页面
* 缓冲区管理器
  + 保证被频繁访问的数据缓存于主存之中
* 文件结构
  + 效率：
    - I/O cost：从二级存储中读取的扇区数
    - CPU cost：使用的CPU指令数
  + 操作：
    - Find, FindNext, Insert, …
  + 文件结构类型：
    - 堆（Heap）/无序：insert很快，其他很慢
    - 有序（Ordered）：findnext很快，其他功能也可以接受
    - 散列（Hashed）：find, insert, delete很快，但是findnext很慢
      * 使用散列函数，选择一个主码域，并将其值映射到一个散列单元中，每个单元存储到独立的磁盘页面
      * 也因此散列结构的文件不适合范围查询
    - 聚类（Clustered）
      * 空间数据没有天然的数据‘
      * 使用填充曲线来实现从高维到低维的映射
* 空间填充曲线
  + 降低空间维度的方法
  + 一条连续曲线，自身没有任何交叉
  + 通过访问所有单元格来填充包含均匀网格的四边形：
    - Z曲线

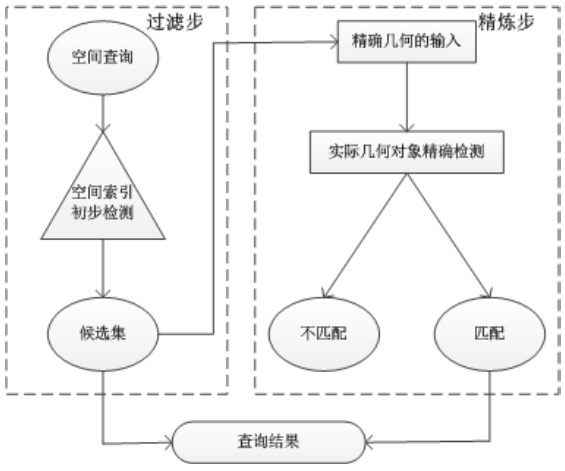
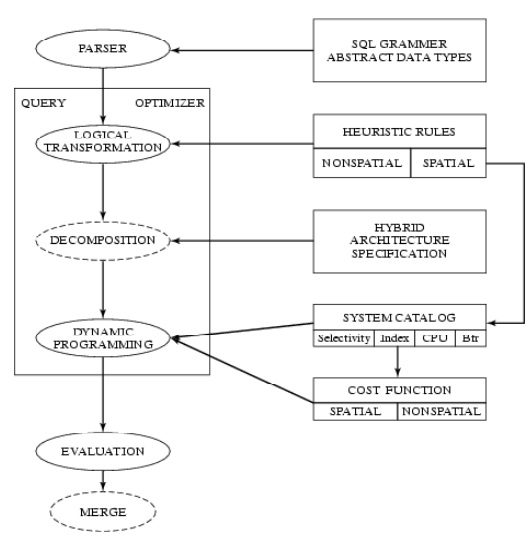


* + - Hilbert曲线
  + Z曲线和Hilbert曲线共同特点：
    - 填充曲线值临近的网格，其空间位置通常也相对临近
    - 任何一种空间排列都不能完全保证二维数据空间关系的维护（编号相邻，空间位置可能很远）
  + 不同点：
    - Hilbert曲线的数据聚集特性更优，Z曲线的数据聚集特性较差
    - Hilbert曲线的映射过程较复杂，Z曲线的映射过程较简单
* 表连接

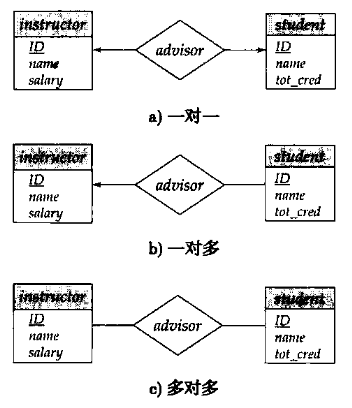
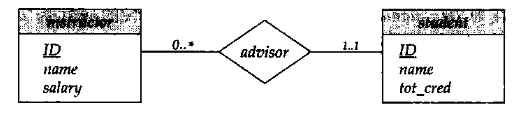
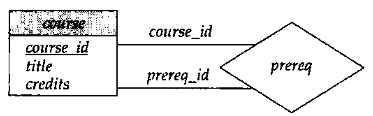


* 空间索引
  + 用一组桶（bucket）（通常对应二级存储的页面）来组织对象
  + 每个桶有一个关联的桶区域，包含了存储在桶中全部对象的一部分空间
  + 桶区域通常是矩形的，彼此之间可能互相交叠（矩形数据结构）
  + 提供空间索引的方法：
    - 在系统中加入专门的外部空间数据结构，为空间属性提供如同B树之于线性属性的功能
    - 使用空间填充曲线，将空间对象映射到一维空间并存储在标准的一维索引中
* 网格索引：
  + 将数据存储在每一个cell中（一个cell是一个桶，对应一块扇区）
  + find, insert, nearest neighbor的效率很高
  + 但是会浪费磁盘空间——当空间分布不均匀的时候
  + 改进：网格文件（Grid File）：
    - 两次访问：
      * 一次访问获得目录项
      * 一次访问得到实际桶以获取实际记录
    - 两个部分：
      * n维网格目录，目录中每一项指向一个数据桶
      * 由线性比例（linear scale）组成的一维数组（n个这样的数组），用来表示网格目录的索引，引用包含对象的块或桶
    - 优缺点：
      * 优点：简单，易于实现，具有良好的可扩展性
      * 缺点：网格大小影响网格索引检索性能
* 四叉树索引：
  + 优缺点：
    - 与网格索引相比，四叉树在一定程度上实现了地理要素真正被网格分割，保证了桶内要素不超过某个量，提高了检索效率
    - 对于海量数据，四叉树的深度会很深，影响查询效率
    - 可扩展性不如网格索引：当扩大区域时，需要重新划分空间区域，重建四叉树，当增加或删除一个对象，可能导致深度加一或减一，叶节点也有可能重新定位
* 空间填充曲线索引：
  + Z曲线：
    - 将Z值作为主键，建立B树（\*,0,1）
    - 通过Z值也可以快速地判断contain/intersect等拓扑关系（eg. 1100 contained un 11\*\*）
  + 当进行空间查询时，先把给定的查询区域转换为覆盖此区域的一系列网格的编码，再到一维的网格编 码索引中，找到对应的记录作为过滤步的候选集
* R树
  + 使用分等级的矩形（最小外包矩形，MBR）来组织数据
  + 是B树在k维上的自然扩展
  + 特性：
    - 每个叶结点包含m至M条索引记录，（m≤M/2），除非他是根节点
    - 一个叶结点上的每条索引记录了（***I****，*元组标识符），***I***是MBR在空间上包含了所指元组表达的k维数据对象
    - 每个非叶结点都有m至M个子结点，除非他是根结点
    - 对于非叶结点中的每个项（***I****，*子结点指针），***I***是在空间上包含其子结点中矩形的最小外包矩形
    - 根结点至少有两个子结点，除非他是叶结点
    - 所有叶结点出现在同一层
    - 所有MBR的边与一个全局坐标系的轴平行
  + 每一个结点对应一个磁盘页面
  + 插入与分裂：
    - 当插入一个元素从而结点需要分裂时，遵循以下规则：
    - 选取两个MBR作为“seeds”
    - 从两个“seeds”开始，不断包括其他MBR，分配的原则为：分配后增长的面积越小越好
  + packed R树：
    - 数据库第一次建立时，就力图将结点矩形的覆盖和交叠最小化
  + R+树：
    - MBR可能会被树中非叶结点的矩形分割：
    - 特点：
      * 对于中间结点的每个项（***I****，*子结点指针），当且仅当***R***被***I***覆盖时，以子结点指针指向的结点为根的子树包括一个矩形R，唯一的例外是当***I***是一个叶结点的矩形，这种情况下，两者只交叠
      * 对中间结点的任何个项，***I1***和***I2***之间的交叠是0
* 空间索引加速：
  + 空间函数通常计算量大，扫描全表两两计算比较耗时，先用Envelope判断两个几何是否有关系，如果有，再进行空间函数
  + 对于8种空间查询方法，除Disjoint和Relate外，可以使用索引：
    - 这些函数只需返回True/False
    - 可以利用索引快速判断获得False

**Lecture 6**

* 查询处理在空间数据库和关系数据库之间的区别：
  + 空间数据库没有固定的运算符集合可以充当查询计算的基本构件
  + 空间数据库要处理非常大量的复杂对象，这些对象具有空间范围，且不能自然地排序成一维数组
  + 需要用到计算量极大的算法，不能再假定I/O代价再CPU的处理代价中占主导地位
* 空间操作：
  + 更新操作
  + 选择操作：
    - 点查询
    - 范围或区域查询
  + 空间连接：
    - intersect
    - contains
    - overlap
    - distance
    - ……
  + 空间聚集
* 空间数据库中空间查询操作一般分为过滤和精炼两步
  + 过滤步是利用空间对象索引信息以及空间对象的近似形状，检索出可能满足该空间查询条件的对象候选集
  + 精炼步是对候选集中的空间对象按查询要求进行精确的处理计算，以获得满足查询条件的最终结果
  + 空间索引主要用于空间查询执行的过滤步
* 查询优化：
  + 一次查询优化：
    - 起始：一个SQL查询语句
    - 终点：一个执行计划（execution plan）
    - 中间步骤：
      * 查询树
      * 逻辑树转换（Logical tree transforms）
      * 策略选择（Strategy selection）
* 多表连接的几种方式：
  + 连接方式的选择主要看：
    - 当前的优化器模式（ALL\_ROWS or RULE）
    - 表的大小
    - 是否有索引
    - 是否已排序
  + **Hash Join：**
    - 两个表差别很大，且较小表（HASH表）可以完全放入内存
    - 或者两个表都可以放入内存
  + **Merge Join：**
    - 没有索引，并且数据已经排序
    - 不等价（＞, ＜, ≤, ≥, <>）关联
  + **Nested Loop：**
    - 一张表比较小（<10000），作为驱动表；另一张表有索引（且索引选择性较好），作为inner表

**Lecture 7**

* 实体-联系(E-R)数据模型：
  + 实体集：是相同类型，即具有相同性质（或属性）的一个实体（现实世界中可区别于所有其他对象的一个“事务”或对象）集合
    - 实体集不必互不相交
    - 实体集通过一组属性来表示，属性是实体集中每个成员所拥有的描述性性质
  + 联系集：是相同类型联系（多个实体间的相互关联）的集合，是n≥2个（可能相同的）实体集上的数学关系
    - 实体集之间的关联称为参与，即E1, E2, …参与联系集R，
    - E-R模式中的一个联系实例表示现实中实体间的一个关联
    - 实体在联系中扮演的功能称为实体的“角色”（role）
    - 联系也可以具有描述性属性
    - 参与联系集的实体集的数目称为联系集的度（degree）
* 数据库的设计：用户需求（高级别的表达方式/文字）→概念设计（选择数据模型，并将其转化为数据库的概念模式），构建实体-模式图→功能需求规格说明→逻辑设计，将高层概念模式（E-R模型定义的）映射到将使用的数据库系统的实现数据模型上→物理设计阶段，指明数据库的物理特征（文件组织格式和索引结构）
* 数据库模式设计的两个主要缺陷：
  + 冗余：理想情况下，信息应该只出现在一个地方
  + 不完整：关系应该是双向的
* 每个属性都有一个可取值的集合，称为“域”，属性是将实体集映射到域的函数
* E-R模型中的属性可以按照如下属性类型来划分：
  + 简单和复合属性（姓名分为姓和名）
  + 单值和多值属性（多个电话号码）
  + 派生属性：由生日派生出年龄，此时生日称为基属性，派生属性的值不储存，而在需要时计算出来
* 映射基数/基数比率：一个实体通过一个联系集能关联的实体的个数
* 码（key）可以用于区分实体，也可以用于区分联系（联系集主码的构成依赖于同联系集R相关联的属性集合）
* 一个好的E-R设计不包含冗余的属性（使用联系集来避免冗余）
* E-R图：
  + 一对多情况下，联系集advisor到实体集instructor的连线采用了箭头，表示一名学生至多拥有一名导师，但一位教师可以指导多位学生
  + 在联系集上的数字标注表示联系集的基数约束，最小值1表示实体集中的每个实体在联系集中的至少一个联系中出现；最大值为1表示实体参与之多一个联系；最大值为\*表示没有限制
  + 在连线上也可以通过标注来表示角色
* 弱实体集：没有足够的属性以形成主码（例子见*数据库系统概念，158*）
  + 弱实体集必须域另一个称作“标识”或者“属主实体集”的实体集关联才能有意义，即弱实体集存在依赖于标识实体集，其联系称为标识性联系
  + 弱实体集通过分辨符（若干个属性的集合）来区分实体，也称为部分码
* 模式的冗余：连接弱实体集与其所依赖的强实体集的联系集的模式是冗余的，而且在基于E-R图的关系数据库设计中不必给出
* 用实体集（加关联）还是用属性：*数据库系统概念，164*
* 用实体集还是用联系集：使用联系集更加紧凑，但是实体集可以用在与其他信息关联时。一个原则是：当描述发生在实体间的行为时采用联系集。*数据库系统概念，164*

**Lecture 8**

* 数据依赖

**Lecture 9**

* 数据完整性：
  + DBMS必须要提供一种机制来检查数据库中的数据，来检查其是否满足语义规定的条件
  + 这些加在数据库数据上的语义约束条件称为**数据库完整性约束条件**
  + DBMS中检查数据是否满足完整性约束条件的机制称为完整性检查
* 完整性约束条件：
  + **关键字为CONSTRAINT**
  + 增加：ALTER TABLE S ADD CONSTRAINT Gender CHECK ( Ssex IN (‘男’, ’女’) );
  + 删除：ALTER TABLE S DROP CONSTRAINT SKey;
* 触发器（Trigger）：触发器中规定用户在对数据库表（关系）执行INSERT、UPDATE、DELETE等操作时，数据库系统应该执行什么相关的操作以保证数据的完整性
  + 当特定事件(event)发生时，检查条件，如果条件为真，则执行特定的操作（Event-Condition-Action Rules）。
  + 用于强制引用完整性
  + 语句级（Statement-level）触发器，在CREATE TRIGGER语句中不包含FOR EACH ROW子句。语句级触发器对于触发事件只能触发一次，且不能访问受触发器影响的每一行的列值。一般用语句级触发器处理有关引起触发器触发的SQL语句的信息——例如，由谁来执行和什么时间执行。
  + 行级（Row-level）/元组级（tuple-level）触发器，在CREATE TRIGGER语句中包含FOR EACH ROW子句。行级触发器可对受触发器影响的每一行触发，并且能够访问原列值和通过SQL语句处理的新列值。行级触发器的典型应用是当需要知道行的列值时，执行一条事务规则。
* 视图：
  + 作用：
    - 简化用户的操作
    - 视图使用户能以多种角度看待同一数据
    - 视图对重构数据库提供了一定程度的逻辑独立性
    - 视图能够对机密数据提供安全保护
  + 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，不会出现数据冗余
    - 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
  + 可以建立在：
    - 单个表
    - 多个表
    - 一个或多个视图
    - 表和视图
* 视图可更新标准（SQL standard）
  + 视图来源（Select without distinct）于单一的表T
  + 视图外的变量可以时null或者有缺省值
  + 子查询与T无关
  + 没有经历GROUP BY或是集聚

\* 以**SELECT \*** 方式创建的视图可扩充性差，应当尽可能避免

* 视图可更新标准（PostgreSQL standard）
  + The view must have exactly one entry in its FROM list, which must be a table or another updatable view.
  + The view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  + The view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  + All columns in the view's select list must be simple references to columns of the underlying relation. They cannot be expressions, literals or functions. System columns cannot be referenced, either.
  + No column of the underlying relation can appear more than once in the view's select list.
  + The view must not have the security\_barrier property
* 事务
  + 是用户定义的一个数据库操作系列，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
  + 以BEGIN TRANSACTION开始
  + 以COMMIT或ROLLBACK结束：
    - COMMIT表示提交，即提交事务的所有操作，将所有对数据库的更新写回磁盘上的物理数据库中，事务正常结束
    - ROLLBACK表示回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，事务中所有已完成的更新操作全部撤消，滚回到事务开始时的状态
  + 事务的四个特性（ACID）
    - 原子性（Atomicity）：事务中的操作，要么全做，要么全不做
    - 一致性（Consistency）：事务执行的结果必须使数据库从一个一致性状态变到另一个一致性状态
    - 隔离性（Isolation）：
      * 为提高事物的吞吐率，大多数DBMS允许同时执行多个事务
      * 在多个事务同时执行的情况下，由于事务要存取数据库中的共享数据，所以事务之间会相互干扰
      * DBMS要保证多个事务的并发执行的效果要等同于系统一次只执行一个事务，一个事务执行完毕，再执行下一个事务，即串行执行事务的效果
    - 持久性（Durability）：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的。接下来的其他操作或故障不应该对其有任何影像
    - 对ACID的保证机制：
      * 原子性和持久性由DBMS系统的恢复机制保证
      * 隔离性由DBMS系统的并发控制机制实现
      * 一致性由事务管理机制的综合机制包括并发控制机制和恢复机制共同保证。
* 并发调度：
  + 会带来数据的不一致性：
    - 丢失修改：后提交的事务结果破坏了先提交的事务的提交结果
    - 不可重复读：
      * 事务1读取某一数据后，事务2对其做了修改，当事务1再次读该数据时，得到与前一次不同的值
      * 出现幻行现象（事务1从数据库中读取某些数据记录后，事务2删除或插入了部分记录，导致再次读取时记录消失/多出）
    - 读取“脏”数据：事务1修改某数据并写回，在事务2读取该数据后，事务1被撤销，数据恢复原值，此时事务2里的数据与数据库中的数据不一致
  + 并发调度具有可串行性：
    - 可串行化：多个事务的并发执行结果必须与按某一次序串行地执行这些事务时的结果相同
    - 为了保证并发调度的正确性，DBMS的并发控制机制必须提供一定的手段来保证调度是可串行化的
  + 目前普遍采用的并发控制方法：
    - 封锁
    - 时标方法（timestamp）
    - 乐观方法
  + 基于封锁的并发控制方法
    - 在事务对数据对象进行操作时加锁，然后才可以进行操作，操作完成后再在某个时刻释放锁
    - 在事务释放它的锁之前，其他事务不能更新此数据对象
    - 封锁技术有S锁和X锁
      * **S锁（**共享锁**/读锁）**：若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。
        + 保证了其它事务可以读A，但在T释放A上的S锁之前不能对A做任何修改
      * **X锁（排他锁/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，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁。
        + 保证了其它事务在T释放A上的锁之前不能读取和修改A
    - 一级封锁协议：
      * 事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放（事务结束包括正常结束COMMIT和非正常结束ROLLBACK）。
      * 可防止丢失修改，并保证事务T是可恢复的
      * 不能保证可重复读和不读“脏”数 据
    - 二级封锁协议：
      * 一级封锁协议 + 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。
      * 进一步防止读“脏”数据
      * 由于读完数据后即可释放S锁，不能保证可重复读
    - 三级封锁协议：
      * 一级封锁协议 + 事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放。
      * 除了防止了丢失修 改和不读“脏”数据外，还进一步防止了不可重复读
    - 两段封锁协议：
      * 每个事务的执行可以分为两个阶段：生长阶段（加锁阶段）和衰退阶段（解锁阶段）
      * **加锁阶段**：在该阶段可以进行加锁操作。在对任何数据进行读操作之前要申请并获得S锁，在进行写操作之前要申请并获得X锁。加锁不成功，则事务进入等待状态，直到 加锁成功才继续执行
      * **解锁阶段**：当事务释放了一个封锁以后，事务进入解锁阶段，在该阶段只能进行解锁操作不能再进行加锁操作
      * 一个事务开始后就处于加锁阶段，一直到执行ROLLBACK和COMMIT之前都是加锁阶段
      * 具体内容
        + 在事务T的R(A)操作之前，先对A加S锁，如果加锁成功，则执行操作R(A)，否则，将R(A)加入A的等待队列
        + 在事务T的W(A)操作之前，先对A加X锁，如果加锁成功 则执行操作W(A)，否则，将W(A)加入A的等待队列
        + 在收到事务的Abort或Commit请求后，释放T在每个数据上所加的锁，如果在数据A的等待队列中不空，即有其它的事务等待对A进行操作，则从队列中取出第一个操作，完成加锁，然后执行该操作
        + 执行Abort和Commit请求后，不再接收该事务的读写操作
    - 事务隔离级别（Isolation Levels）
      * 事务的隔离级别有：Read Uncommitted，Read Committed，Repeatable Read，Serializable四种，级别依次递增。关系型数据库（除MySQL以外）默认的隔离级别为Read Committed

|  |  |  |  |
| --- | --- | --- | --- |
| 事务隔离级别 | 脏读 | 不可重复读 | 幻读 |
| Read-Uncommitted（读未提交） | 有 | 有 | 有 |
| Read-Committed（读已提交） | 无 | 有 | 有 |
| Repeatable-Read（可重复读） | 无 | 无 | 有 |
| Serializable（可串行化） | 无 | 无 | 无 |

* + - * Read-Uncommitted（[@疯之墒](https://blog.csdn.net/oyw5201314ck/article/details/79621347)）
        + 事务所作的修改在未提交前，其他并发事务是可以读到的
        + 存在脏读——在查询过程中进行的修改不会更新到查询结果中
      * Read-Committed（[@疯之墒](https://blog.csdn.net/oyw5201314ck/article/details/79621954)）
        + 一个事务只能看到其他并发的已提交事务所作的修改
        + 解决了Read-Uncommitted中的脏读问题
        + 存在不可重复读——两次查询之间如果有修改那么结果会不同
      * Repeatable-Read（[@疯之墒](https://blog.csdn.net/oyw5201314ck/article/details/79622649)）
        + 解决了Read-Committed中的不可重复读问题：

即使两次查询间有UPDATE，两次查询人会返回一样的结果（UPDATE前）

但是如果两次查询间有INSERT，两次查询结果不一样——会有幻读

* + - * Serializable（[@疯之墒](https://blog.csdn.net/oyw5201314ck/article/details/79623253)）
        + 执行事务时回阻塞其他并发事务（通过锁来实现）
        + 脏读、不可重复读、幻读统统不会有
        + 但是处理事务的吞吐量低，严重浪费数据库的性能
* 数据库故障：
  + 事务故障
  + 系统故障
  + 介质故障
  + 计算机病毒
* 数据库恢复
  + 转储
    - 静态转储：系统停止对外服务，不允许用户运行事务，只进行转储操作
      * 优点：实现简单
      * 缺点：降低了数据库的可用性：
        + 转储必须等待用户事务结束才能进行；
        + 新的事务必须等待转储结束才能执行
    - 动态转储：转储操作与用户事务并发进行，转储期间允许对数据库进行存取或修改：
      * 优点：
        + 不用等待正在运行的用户事务结束 
        + 不会影响新事务的运行
      * 缺点：不能保证副本中的数据正确有效
  + 日志文件：
    - 基于记录的日志文件
      * 每条日志记录的内容
      * 事务标识（标明是那个事务）
      * 操作类型（插入、删除或修改）
      * 操作对象
      * 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）
    - 基于数据块的日志文件
      * 每条日志记录的内容
      * 事务标识（标明是哪个事务）
      * 更新前数据所在的整个数据块的值（对插入操作而言，此项为空值）
      * 更新后整个数据块的值（对删除操作而言, 此项为空值）
    - 登记日志文件的两条原则：
      * 登记的次序严格按并行事务执行的时间次序
      * 必须先写日志文件，后写数据库

**Lecture 11**

* 图：
  + 一个图*G=(V, E)*是由一个有限顶点集V和顶点之间的边集E组成的。顶点和边有时也成为结点和连接
  + 有向图/无向图：由构成边集的个顶点对是否有序决定
  + 结点不一定是空间中的点：以河流网为例，图的结点表示的是河流，连接表示的是河流之间的流入关系
  + 两条边共享一个结点称为邻接，一系列邻接边组成一条路径，闭合的路径形成环
* 图的物理数据模型
  + 邻接矩阵：可以快速回答对边的查询，例如：边(u,v)是否在图G中
  + 邻接表：指针数组，可以快速的查询临近顶点
* 传递闭包：
  + 图G(V, E)的传递闭包G\*是瞒住下列条件的图：
    - G\*与G有相同的顶点集V
    - G\*的边集由G的所有路径组成
  + 对于一个关系R，其传递闭包X是递归方程X=X∪(R JOIN X)的最小点集解
* SQL CONNECT子句：
  + 用于遍历一个有向无环图（DAG）
  + 使用广度优先搜索（BFS）来遍历DAG
  + 输入：
    - DAG的边
    - 开始顶点
    - 遍历方向
  + 输出：DAG的传递闭包
* 路径计算：
  + 单对
  + 单源：部分传递闭包问题
  + 所有对
* 图遍历算法是所有路径查询的计算基础：
  + 广度优先(BFS)
  + 深度优先(DFS)
  + Dijkstra
  + A\*：在建立了有效的评估函数时，效率要高于Dijkstra算法
* 连通性剩余率CRR=非分离边总数/边总数
* CCAM：连结性聚集的空间网络存取方法

**Lecture 13 - OLAP**

* OLTP & OLAP
  + OLTP：Online Transaction Processing（联机交易处理）
    - 短处理
    - 简单查询
    - 使用小部分数据
    - 频繁更新
  + OLAP：Online Analytical Processing（联机分析处理）
    - 长处理
    - 复杂查询
    - 使用大部分数据
    - 不频繁更新
* 维（Dimension）：观察数据的特定角度
  + 层次（Hierarchy）：就某个维的不同细节组成（e.g. 年、季、月、日等就是时间维的一种层次）
* 星型模式（Star Schema）：
  + 由一个中心表（事实表）和一组维表组成
  + 事实表（Fact table）：更新频繁（往往只是不断的添加新数据），数据量很大
  + 维表（Dimension tables）：更新不频繁、不大
  + 一个例子：事实表=销售记录表；维表=商店，商品，顾客
  + 星型模式的事实表与所有的维表相连（通过码），而每一个维表只与事实表相连
* 多维分析的基本操作：
  + 切片 (Slice)：使原本的维数-1，如：产品销售状况→1997年产品销售状况
  + 向上综合 (Roll-up)：也称为上钻操作，提供超立方体Cube上的聚集操作：
    - 在某个维的某一层次上由低到高的聚集操作，如在时间维上由日聚集到月，由月聚集到年
    - 通过减少维的个数进行聚集操作，如在二维Cube中包含有时间维和地区维，把地区维去掉，则得到一个按时间维对所有地区进行的聚集操作
  + 向下钻取(Drill-down)：也称为下钻操作，是Roll-up的逆操作：
    - 在某个维的某一层次上由高到低的钻取操作，找到更详细的数据：如在时间维上，先看年，在看月
    - 通过增加新的维来获取更加细节的数据
* With Cube:
* With Rollup

**Lecture 13 - NoSQL**

* NoSQL = Not Only SQL:
  + 不是所有的问题都能单纯地通过传统的DBM得到很好的解决
* NoSQL的特点：
  + Performance：性能出众
  + Scalability：可扩展性好
  + Schema flexibility：架构灵活性佳（是半结构化数据）
  + Analytics capabilities：分析能力强
* 传统的DBMS在需要ACID特性的数据中仍有广泛的作用，但是NoSQL更适用于：
  + 在现实中以无结构或半结构化的方式存储的数据
  + 处理数据的程序需要有一定的性能或扩展性时
  + 处理数据的程序本身已经能保证结果的最终一致性（因此不再需要ACID的约束）
* NoSQL以BASE来代替ACID：
  + Basic Availability: 基本可用
  + Soft-state: 软状态/柔性事务，“无连接”
  + Eventual consistency: 最终一致性
* 当前的NoSQL数据库系统的分类：
  + MapReduce framework：分布式计算框架
  + Key-value stores：通过key快速查询到其value，存储不用考虑value的格式
  + Document stores：一般用类似json的格式存储，存储的内容是文档型的，有机会对某些字段建立索引，实现关系数据库的某些功能
  + Graph database systems：图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便