数据库

- 1、数据库概念与设计
  - 关系型数据库、非关系型数据库?
    - 关系型数据库：二维表格形式，可实现关联查询（通过相同属性）
    - 非关系型数据库NoSQL：不具备关联查询等，但查询速度快
      - 键值型数据库redis(内存缓存)
      - 文档型数据库mongoDB：特别的键值型数据库，键是文档
      - 列式数据库hbase：减少系统IO（行式存储冗余字段多，IO多，列式可以只选取想要的字段），适合分布式文件系统
      - 搜索引擎数据库elasticsearch：倒排索引
  - 什么是数据、DB、DBMS?
    - 数据：描述事务的符号记录
    - 数据库：长期存储在计算机内、有组织、可共享的大量数据的集合
    - 数据库管理系统：一种系统软件，可实现数据定义、操纵、存储管理；数据库建立、维护功能、事务管理
  - 数据库系统DBS的组成?
    - 硬件平台及数据库（计算机+DB）
    - 数据库管理系统DBMS（及其应用开发工具）

- 用户（DBA，应用程序员）
- **数据库系统的数据独立性?**
  - 物理独立性：应用程序与数据物理存储独立
  - 逻辑独立性：应用程序与数据库逻辑结构独立
  - 不会因为数据存储结构和逻辑结构的变化影响应用程序，由二层映像实现
- **数据库系统的特点?**
  - 1、数据共享性高、冗余度低、易扩充 容易
  - 2、数据独立性高
  - 3、数据结构化（与文件系统的本质区别）共解读
  - 4、由DBMS管理控制
- **文件系统特点?**
  - 优点：数据长期保存；由文件系统管理数据 尚雯婕
  - 缺点：共享性差、冗余度大；独立性差
- **DBA职责?**
  - 1、决定数据库信息结构和内容
  - 决定数据库存储结构和策略
  - 定义安全性、完整性要求
  - 2、监控数据库使用运行
  - 3、数据库改进
- **关系的性质?**

- 1、每个分量不可再分（最基本）
- 列的顺序无所谓、行的顺序无所谓
- 列是同质的
- 2、两个元组的候选码不能取相同的值
- 不同列可能出自同一域

- 逻辑数据模型的组成?
  - 数据结构、数据操作、完整性约束
  - 数据库类型的划分依据数据模型
- 数据库设计流程?
  - 需求分析
  - 概念结构设计
    - 将需求分析得到的用户需求抽象成概念模型
    - 独立于机器，E-R模型是描述概念模型的工具
  - 逻辑结构设计：将概念结构设计阶段设计好的ER图转化为DBMS产品支持的逻辑数据模型
  - 物理结构设计：
    - 1、确定数据库的物理结构。
      - 存取方法：B+树索引（聚簇索引否？）、哈希索引
      - 存储结构（数据存放位置、系统配置）
        - 存取位置：易变和稳定部分，经常存取和存取频率较低应当分开存放

- 2、评价物理结构：估算各种方案时间、空间效率、维护代价，权衡选择
- 数据库实施（数据载入、应用程序调试）、运行和维护

- 什么是概念模型?
  - 用一组概念描述一个系统，反映真实世界，独立于机器
  - ER图（实体联系模型）是概念模型的一种表示方法
- 什么是逻辑模型?
  - 反映数据的逻辑结构。用于DBMS实现，包括层次、网状、关系、面向对象模型等
  - 概念转逻辑，一般就是ER图转关系模式
- 有哪些逻辑数据模型?
  - 层次模型、网状模型、关系模型、面向对象数据模型...
- 概念结构设计的方法?
  - 自顶向下：首先定义全局概念结构的框架，然后逐步细化
  - 自底向上：首先定义各局部应用的概念结构，然后将它们集成起来，得到全局概念结构
  - 逐步扩张：首先定义核心概念结构，然后向外扩充，逐步生成其他概念结构，直至总体概念结构

- 混合策略：将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，自底向上策略设计各局部概念结构

- 属性的划分准则?
  - 每个属性不可再分
  - 属性不能与其他实体有联系
- 如何把ER图转换成关系模式?
  - 每个实体，m：n的属性 独立成一个关系模式
  - n端属性：自己的+联系的+1端主键
  - m:n属性：自己的+两端主键
- 模式分解两条基本原则?
  - 保持函数依赖性
  - 分解的无损连接性：分解后的关系通过自然连接可以恢复成原来的关系
- 三类ER图冲突？署名杰
  - 属性冲突：属性域冲突（类型、范围）、取值单位冲突
  - 命名冲突：同名异义，异名同义
  - 结构冲突：
    - 实体间联系在不同ER图为不同类型
    - 同一实体在不同ER图中属性个数、排列顺序不同
    - 同一对象在一ER图中被当做实体、另一ER图中当做属性

- 什么是SQL注入？原因？
  - 攻击数据库的手段
  - 未对用户提交数据进行合法性检查导致
  - 用户提交数据库查询代码，获得想要的数据
- 关系的三种类型？
  - 基本表、查询表、视图表
- 关系和关系模式的区别？
  - 关系是元组的集合，是关系模式某一时刻的状态/内容
  - 关系的描述和抽象是关系模式
  - 关系：动态、变化（关系操作不断更新数据库的数据）
  - 关系模式：静态、稳定
- 什么是数据字典？
  - 数据库中数据的描述
  - 1、数据项：不可再分的数据单位
  - 2、数据结构：反映数据之间的组合关系
  - 3、数据流：数据结构在系统内的传输路径
  - 4、数据存储：数据结构停留保存的地方，数据流的来源和去向之一
  - 5、处理过程：描述处理过程的功能及处理要求
- 三级模式？

- 模式：概念/逻辑模式，对全体数据逻辑结构和特征的描述，一个数据库一个模式（内、外模式之间，是关系的） 卡卡罗特
- 外模式/子模式：用户看到的数据视图，一个数据库多个外模式（关系的）
- 内模式：存储模式，数据物理结构和存储方式的描述，一个数据库一个内模式（无关系的）

- 三级模式的作用
  - 外模式：保证数据库安全性（每个用户只能看到访问对应外模式的数据）；方便用户取用数据，模式是所有用户的公共数据视图
  - 内模式：优化内模式，可以提高存取效率
  - 模式：减少数据冗余，实现数据共享
- 两层映射?
  - 外模式、模式映射
    - 外模式是模式的一部分
    - 模式改变时，外模式不用变
    - 逻辑独立性
  - 模式、内模式映射
    - 数据存储结构改变，应用程序不受影响
    - 内模式改变，模式不用变
    - 物理独立性
- 视图作用?
  - 简化用户操作

- 用户可以以不同视角看待同一数据
- 对机密数据提供安全性保护

- 视图、基本表?
  - 视图是一个或多个基本表导出的表，是虚表
  - 数据库只存视图定义，不存数据，数据在基表中，基表变化，视图变化
  - 对视图的增删改有限制
- 什么时候指出视图所有列名?
  - 目标列是聚集函数或表达式
  - 多表连接选出了几个同名列
  - 要在视图中为某列启用更合适名字
- 视图消解?
  - 查询视图时，先检查查询涉及的基本表、视图是否存在
  - 从数据字典中取出视图定义，把定义中子查询与用户定义结合，转化为对基本表的查询
  - 执行修正后的查询（更新也需要视图消解）

- 2、数据库运算
- SQL语言分类?
  - 数据查询语言DQL: SELECT　　data query
  - 数据定义语言DDL: CREATE/DROP/ALTER（从无到有，改变结构）　　data definition
  - 数据操纵语言DML: INSERT/DELETE/UPDATE（增删改）　　data

manipulation

- **数据控制语言DCL: GRANT/REVOKE/COMMIT/ROLLBACK（安全性控制）** data control

- **SQL特点?**
  - **综合统一。集DCL，DML，DDL功能于一体；**
  - **高度非过程化。只需要提出“做什么”，而不需要指明怎么做；**
  - **面向集合的操作方式。**
  - **提供多种使用方式。既可以作为独立的语言进行交互，又可以作为嵌入式语言嵌入到更高级的语言程序中进行操作；**
- **MYSQL四大数据类型?**
  - **整数、小数、字符串、日期**
- **关系代数运算符有哪些**
  - **传统集合运算符：并、差、笛卡尔积、交**
  - **专门的关系运算符：选择、连接、投影、除**
  - **其中基本运算为：并、差、积、选择、投影**
- **除运算?**
  - **R÷S：在R中选包含S的行，再去除S列**
- **DROP和DELETE的区别**
  - **DROP把数据内容和结构都删除**
  - **DELETE删除数据，可搭配WHERE删除表中数据的一部分**

- sum和count区别?
  - sum用于求和，count用于行数统计
- 笛卡尔积、连接、等值连接、自然连接?
  - 笛卡尔积：关系模式A与B，A中有k1个元组，每个元组有m列；B中有k2个元组，每个元组有n列，那么A与B的笛卡尔乘积则为拥有k1*k2个元组，且每个元组有m+n列的集合。
  - 连接：连接是特殊的笛卡尔乘积，即从两个关系的笛卡尔乘积中选择符合特定条件的元组
  - 等值连接：等值连接是特殊的连接，即从两个关系的笛卡尔乘积中选择某些属性值相等的元组。
  - 自然连接：自然连接是特殊的等值连接，要求有同名属性列才可连接，且去重

3、数据库完整性、安全性

- 数据库完整性？DBMS应提供怎样的功能实现?
  - 数据库数据的正确性和相容性，正确性：符合现实世界语义，相容性：同一对象在不同关系表中的数据符合逻辑
  - 1、定义完整性约束的机制
  - 2、提供完整性检查
  - 3、提供违约处理
- 数据库保护问题包括哪些方面?
  - 安全性、完整性、故障恢复、并发控制

- 数据库完整性、安全性区别?
  - 完整性：保证数据库数据正确性、相容性；防范对象是不合语义的数据
  - 安全性：保护数据库防止恶意破坏和非法存取；防范对象是非法用户和非法操作
- 关系模型完整性约束（关系的约束条件）
  - 实体完整性：主键不为空
  - 参照完整性：外键必须在另一关系模式中存在，或为NULL
  - 用户定义完整性
  - 完整性指正确性和相容性
- 实现参照完整性要注意什么?
  - 1、定义时外键必须在另一关系模式中存在，或为NULL
  - 2、参照完整性检查和违约处理
    - 参照表：插入元组、修改外码值——拒绝
    - 被参照表：删除元组、修改主码值——拒绝/级联修改/设置为空值 P161
- 触发器?
  - 通过事件触发执行的特殊存储过程，是保证数据库完整性的一种方法。
  - 与其他完整性约束的区别？动作体可以很复杂，通常是一段存储过程
  - CREATE TRIGGER 触发器名

- BEFORE/AFTER 触发事件 ON 表名
- REFERENCING NEW/OLD ROW AS 变量
- FOR EACH ROW/STATEMENT
- （WHEN 条件）触发动作体

- 行级触发器FOR EACH ROW、语句级触发器STATEMENT?
  - 行级：对表中每行语句执行一次
  - 语句级：触发动作体之前、后执行一次
- 实现数据库安全性的措施?
  - 1、用户身份鉴别
  - 2、存取控制：（定义用户权限，合法权限检查)
    - 自主存取控制：用户自主决定将数据的存取权限授予何人，决定是否将授权的权限也授予他人，通过GRANT、REVOKE实现
    - 强制存取控制：对数据本身进行密级标记，只有符合密级标记要求的用户才可操纵数据
  - 3、视图机制：为不同用户定义不同视图，把对象限制在一定范围内
  - 4、审计：将用户对数据库操作记录到审计日志中，方便事后检查
  - 5、数据加密：存储加密、传输加密

- 4、关系数据理论
- 什么是数据依赖?

- 关系内部、属性与属性之间的约束关系，通过值相等体现，包括函数依赖、多值依赖
  - 函数依赖：关系中属性间的对应关系（给定一个x对应唯一一个y，且y的取值由x决定）
    - 完全函数依赖：X的真子集 无法确定Y，Y完全函数依赖于X
    - 部分函数依赖：X的一部分 可以确定Y，Y部分函数依赖于X
    - 传递函数依赖： X —> Y, Y 不决定 X 且 Y —>Z，则有 X —>Z
  - 多值依赖：一个主码决定一组值
- 只满足1NF关系模式会产生的问题/关系规范化为解决什么问题?
  - 插入异常：元组码值为空，无法插入表中
  - 删除异常：删除不应删除的信息
  - 修改复杂
  - 数据冗余：数据重复存储
- 码、候选码、主码、全码、主属性、非主属性?
  - 码：=超码，能唯一标识一个元组的属性集（一个或多个属性）
  - 候选码：能唯一标识一个元组的最小属性集（少一个都不行）
  - 主码：候选码中选一个
  - 全码：所有属性共同构成一个候选码

- 主属性：包含在任一候选码中的属性
- 非主属性：不在候选码中的属性

- 什么是范式?
  - 满足一定要求的关系
- 关系规范化?
  - 通过分解关系模式，逐步消除数据依赖中不合适的部分
  - 目的：解决关系模式存在插入异常、删除异常、修改复杂、数据冗余的问题
  - 过程：1NF到4NF
- 1NF/2NF/3NF/BCNF/4NF判断?
  - 1NF：每个属性不可再分
  - 2NF：不存在非主属性对码的部分函数依赖，主码的一部分可以确定非主属性
  - 3NF：不存在非主属性对码的传递函数依赖
  - BCNF：每个决定因素（箭头左边）都是候选码，只考虑函数依赖，BCNF规范化程度最高
  - 4NF：消除多值依赖（一个主码决定一组值）
- 什么是平凡、不平凡的函数依赖?
  - 平凡函数依赖（X包含Y，且X确定Y）
  - 不平凡的函数依赖（X不包含Y，且X确定Y）
  - 平凡函数依赖必然成立，因此只讨论非平凡函数依赖

- 5、事务、故障恢复

- 什么是事务?
  - 事务是满足ACID特性的一组操作
- 事务的四大特性?
  - 原子性Atomicity：事务是不可分割的最小单元，要么全部成功，要么全部失败回滚
  - 一致性Consistency：多个事务对同一个数据读取的结果是相同的
  - 隔离性Isolation：事务所做的修改，提交之前，对其他事务不可见
  - 持久性Durability：事务提交，修改会永远保存到数据库中，数据库崩溃也不会消失
- 如何实现事务的ACID特性?
  - 日志：事务的原子性、一致性、持久性
  - 锁机制：隔离性（多事务更新相同数据，只允许有锁的事务更新，前一个事务释放锁，其它事务才能更新数据）I lock
- redo log和undo log的区别?
  - 重做日志redo log：数据写磁盘之前把所有操作先记录下来，数据库崩溃重做操作，效率低
  - 回滚日志undo log：重做日志没写完先写一点数据，记录在什么地方写了什么数据（如果事务提交之前数据库崩溃，先写的数据就变成了脏数据，利用undo恢复数据）
- ROLLBACK 和 UNDO 的区别

- 回滚：事务运行过程中发生了某种故障，撤销全部操作（在事务未完成时发生）
- 撤销：撤销对数据库的任何修改（事务执行完发生）

- 数据库有哪些故障?
  - 事务内部的故障：
  - 系统故障：软故障；某事件使得系统停止运转，系统要重新启动
  - 介质故障：硬故障，外存故障
  - 计算机病毒
- 事务故障恢复?
  - 反向扫描日志文件undo log，查找更新操作
  - 把更新前的值写回数据库 撤销更新
  - 重复此步骤，到事务的开始标记
- 系统故障恢复?
  - 正向扫描日志，找到故障前提交的事务，标记重做（重做操作）
  - 故障时尚未完成，标记撤销（撤销数据修改）
  - 撤销、重做
  - 注：数据库恢复利用数据转储或登记日志文件（数据冗余思想）
- 介质故障恢复?
  - 重装数据库，重做已完成的事务

- 数据库中为什么要有恢复子系统？它的功能是什么？
  - 计算机系统中软硬件错误、操作员失误、恶意破坏不可避免，这些故障轻则造成运行事务中断，影响数据正确性，重则破坏数据库，造成数据丢失，因此...
  - 功能：把数据库从错误状态恢复成故障前某个一致性状态
- 什么是日志？
  - 记录事务对数据库更新操作的文件
- 登记日志的原则？
  - 严格按照并发事务执行次序
  - 先写日志文件、再写数据库
- 数据库镜像？
  - DBMS根据DBA的要求，自动把整个数据库或部分关键数据，复制到另一个磁盘上
  - 主数据库更新，DBMS自动把数据复制过去
- 数据库镜像/副本用途
  - 介质故障时，可用镜像恢复
  - 不故障，用于并发操作：一个用户加排他锁修改数据时，另一用户可读镜像数据库上的数据
- 数据库快照？
  - 某一时间点的静态视图，只读
  - 故障恢复

- 6、并发控制
  - 并发不一致?
    - 定义：事务并发，存取到不正确的数据
    - 破坏事务的隔离性和一致性
    - 解决方法：并发控制
  - 并发一致性问题有哪些?
    - （这些问题源于事务之间的相互影响）
    - 丢失修改：两事务都改，一前一后，后面的覆盖前面的修改
    - 脏读：事务读到了其它事务未提交的数据，事务读到的数据和数据库中数据不一致
    - 不可重复读：前后两次读取数据不同
    - 幻读：不可重复的一种，事务不独立执行时出现。一事务修改表中数据，涉及全部行，另一事务也修改表中数据，插入一行，那么，第一个事务发现表中有未修改的数据行，像幻觉一样
  - 脏读和不可重复读最根本的原因
    - 读到其他事务未提交的数据
  - 并发控制方法?
    - 封锁（读写锁、意向锁）
    - 事务隔离
    - 时间戳：记录每个事务开始的时间

- 乐观控制法：假设事务执行正确，事务提交前正确性检查，发现有冲突，回滚
- 多版本并发控制：维护数据库对象的多个版本

- 封锁粒度？
  - 封锁对象的大小
- 读写锁？
  - 互斥锁、写锁、X锁：只允许一个事务读取+更新
  - 共享锁、读锁、S锁：允许多事务读取
- 意向锁？
  - 意向读IX：想获得X，先获得IX；拟对后裔加写锁
  - 意向写IS：想获得S，先获得IS；拟对后裔加读锁
  - 意向锁之间都是兼容的
- 一级封锁协议？
  - 事务修改数据必须加X锁，事务结束释放
  - 解决丢失修改，使得不能有两事务同时对同一数据修改一改，长
- 二级封锁协议？
  - 满足一级
  - 事务读取数据必须加S锁，读完马上释放
  - 解决读脏数据二读，短
- 三级封锁协议？

    - **满足一级**
    - **事务读取数据必须加S锁，事务结束释放**
    - **解决不可重复读**三读，长
- **事务隔离级别?**
    - **读未提交read uncommitted——事务中的修改，未提交，也可读**
    - **读已提交read committed——只能读提交过的数据 脏读**
    - **可重复读repeatable read——多次读数据结果一致 脏读+不可重复读+丢失修改（除了幻读)**
    - **可串行化serializable——事务串行执行 完全服从ACID原则，牺牲并发性**
- **两段锁协议?**
    - **事务必须分两个阶段对数据项加锁和解锁。**
    - **第一个阶段获得封锁。事务可以获得任何数据项上的任何类型的锁，但是不能释放**
    - **第二阶段释放封锁，事务可以释放任何数据项上的任何类型的锁，但不能申请。**
- **冲突操作、冲突可串行化?**
    - **冲突：不同事务对同一数据读写、写写操作**
    - **不可交换次序的冲突操作：同一事务两个操作、不同事务冲突操作**

- 冲突可串行化：保证冲突操作顺序不变，交换不冲突操作得到另一调度，新调度串行，原调度是冲突可串行化的调度
- 可串行化调度?
  - 多个事务并发执行的结果与按某一次序串行执行的结果相同
  - 可串行化调度的充分条件：1.冲突可串行化 2.事务遵守两段锁协议
- 活锁和死锁?
  - 死锁：多事务循环等待对方的资源，无外力，无法向前推进——死锁预防、检测和解除
  - 活锁：拿到资源，互相释放，不执行——先来先服务
- 死锁预防?
  - 一次封锁法：每个事务必须一次性把要用的数据全部加锁
  - 顺序封锁法：预先规定事务对数据对象封锁顺序
- 死锁检测和解除?
  - 超时法：事务等待时间超过规定时限，发生死锁
  - 等待图法：事务等待图出现回路
  - ——>解除：选择一个处理死锁代价最小的事务，撤销，释放所有锁

- 什么是封锁?
  - 事务在操作数据前先对其加锁，释放锁之前其他事务不能对数据进行更新
- 7、性能（索引）
- 数据库索引？属于哪个模式?
  - 一种特殊文件，存放表中记录的引用指针，属于内模式的范畴
- 索引作用及优缺点?
  - 当表的数据量比较大时，查询操作比较耗时，建立索引可以加快查询速度。
  - 优点：加速查询速度；
  - 缺点：索引需要占一定的存储空间，且基本表更新时需要维护索引表。
- 索引分类（按列属性分）
  - 普通索引
  - 唯一索引：索引标记的属性列不能有重复数据，允许有空值
  - 主键索引：定义主键时自动创建，是一种唯一性索引
  - 全文索引：每个表只允许建立一个全文索引
  - 组合索引：指定多列作为索引列
- 索引分类（按结构分）
  - 顺序文件上索引：顺序项的索引与表中记录的物理顺序一致的索引

- **B+树索引：将索引组织成B+树形式；属性经常在查询条件、连接条件中出现、常作为聚集函数的参数**
- **hash索引：要给某张表某列增加索引时，将这张表的这一列进行哈希算法计算，得到哈希值，排列在哈希数组上**
  - **优点：时间空间复杂度低**
  - **缺点：哈希值可能存在碰撞；只能进行等值比较、不能范围查询；存储无序，不支持查询时排序**

- **聚簇索引、非聚簇索引?**
  - 二者都用B+树实现
  - **聚簇索引：为提高属性查询速度，索引和数据存在一起（索引项的顺序和表中记录的物理顺序保持一致），叶子结点就是数据节点；一个表只能有一个聚簇索引**
    - **优点：属性查询速度快**
    - **缺点：建立、维护聚簇开销大**
  - **非聚簇索引：将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行**
- **B树（多路平衡查找树）特性?**
  - **多路平衡查找树**
  - **1、节点：根节点可以只有一个关键字（两个孩子），除了根结点和叶子结点外，每个结点最少有 m/2（向上取整）个子结点**

- 2、排序：每个节点中的元素（关键字）从小到大排列；每个元素子左结点的值，都小于或等于该元素，右结点的值都大于或等于该元素。
- 3、所有叶子节点在同一层

- B+树与B树不同的地方？
  - 节点个数与关键字个数相等
  - 内部节点不包含数据，叶子节点包含所有数据
  - 叶子节点之间以指针相连
- 使用B+树作为索引数据结构的好处？
  - 内部节点不放数据，一次读取，一页内能存放更多的键
  - 叶节点由一条链相连，方便批量读取数据
- 数据库恢复技术实现？
  - 数据转储、登记日志文件
- 数据库并发技术实现？
  - 封锁
- select语句执行顺序
  - from 从哪里 on join 联接表 where 从表中筛选
  - group by 分组
  - 聚集函数
  - having 筛选分组
  - select选择

- order by排序

8、数据库编程

- 存储过程、函数?
  - 封装一组操作一组操作，编译后保存在数据库中，运行效率高
  - 函数和存储过程类似，但必须指定返回类型
- 嵌入式SQL语言与主语言通信?
  - SQL操作数据库，高级语言控制逻辑流程
  - 1、SQL通信区：存放SQL执行的状态信息
  - 2、主变量：主语言程序变量
  - 3、游标：数据缓冲区，存放SQL语句执行结果
- 数据仓库?
  - 为支持决策，面向主题、集成、不可更新（主要用于查询）、不断变化（增新删旧，不同综合方式）的数据集合
  - 体系结构：源数据、数据仓库、数据应用
    - 源数据—数仓需要ETL工具：抽取、转换、装载数据 eg: Kafka(处理结构化数据)
  - 与数据库区别?
    - DB：主要用于操作型处理，对数据查询修改，也称OLTP（联机事务处理）
    - 数仓：面向主题，分析数据，支持决策，也称OLAP（联机分析处理）

- **数据挖掘数据可以来自DB或数仓**

以上内容整理于 幕布文档