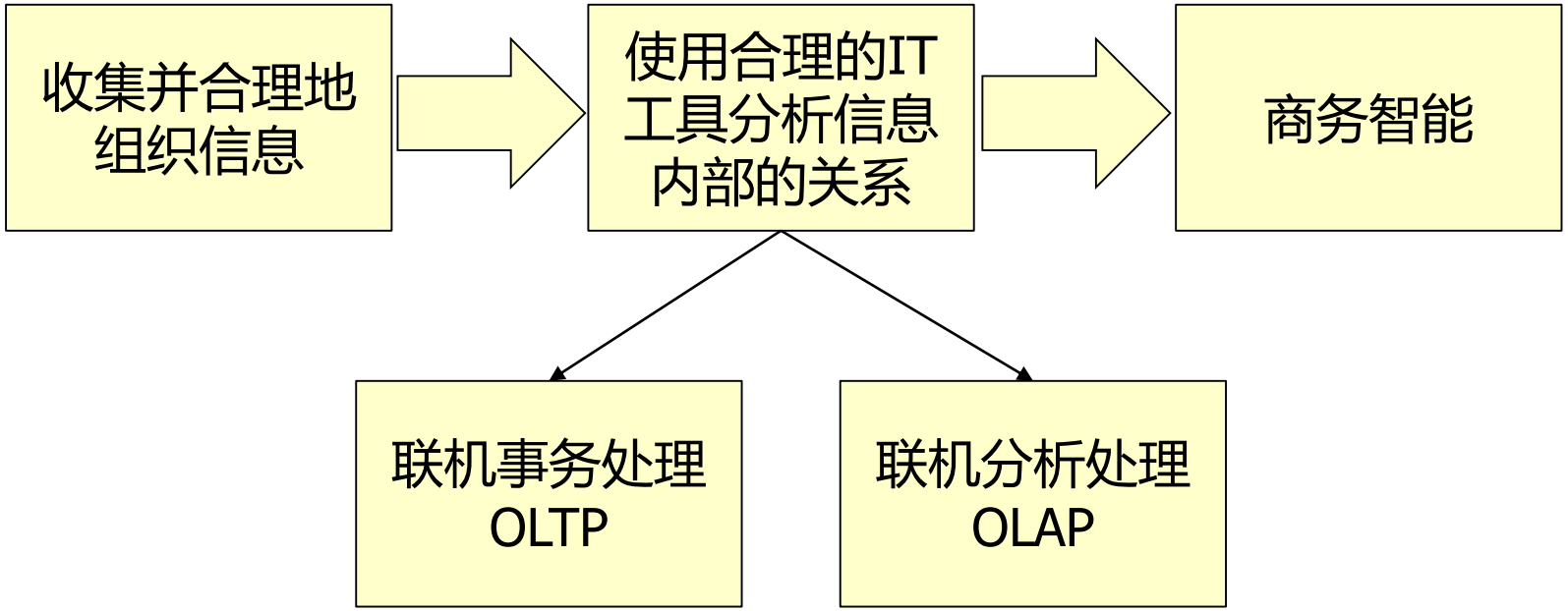
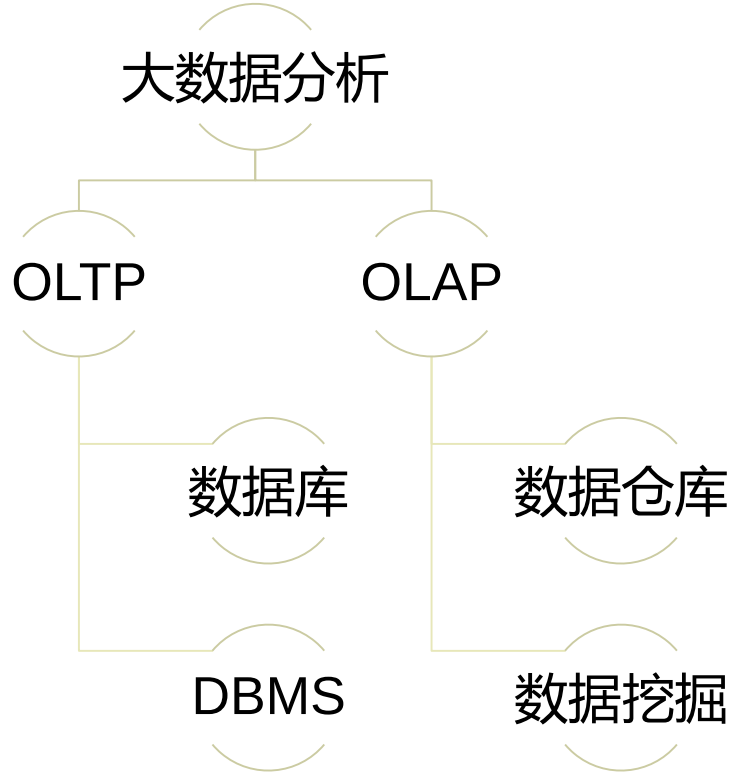
大数据分析基础

数据库基础及应用

- 数据库基础——关系型数据库
- MySQL
- 利用Python操作数据库
- 数据库在大数据时代的发展

- 大数据分析无法凭空出现





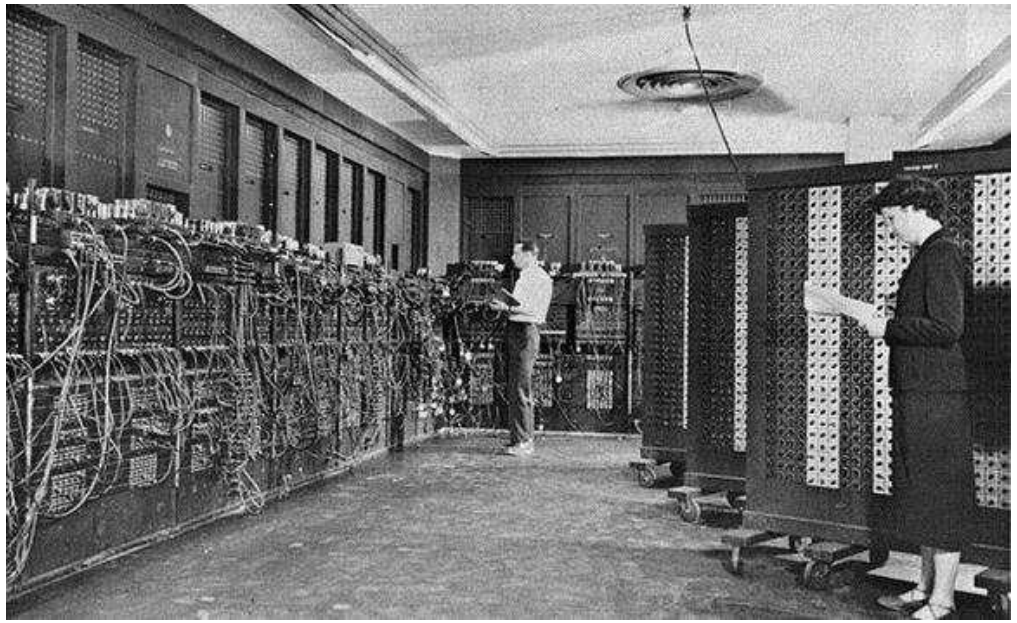
- 联机事务处理（Online transaction processing，OLTP）
  - 输入信息收集、处理、更新
  - 支持业务处理
  - 销售订单、应收账款等
  - 由业务数据库和数据库管理系统来支持
- 联机分析处理（Online analytical processing，OLAP）
  - 一种提供决策支持的信息处理方式
  - 帮助进行数据分析
  - 由数据仓库和数据挖掘工具来支持

- 数据管理技术的三个阶段
  - 人工管理阶段（20世纪50年代中期以前）
  - 文件系统阶段（20世纪50年代后期—60年代中期）
  - 数据库系统阶段（20世纪60年代后期—今）

背景

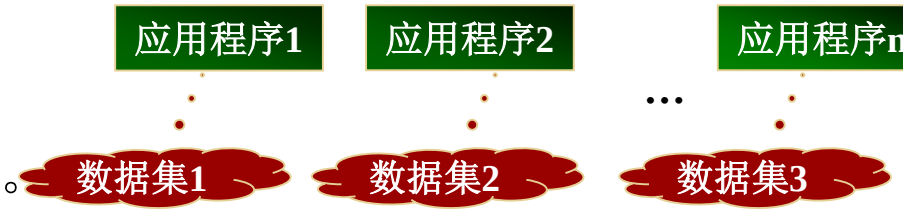
计算机主要用于科学计算

- 数据量小、结构简单，如高阶方程、曲线拟和等
  - 1946年ENIAC是美国奥伯丁武器试验场为了满足计算弹道需要而研制成的，这台计算器使用了17840支电子管，大小为80英尺×8英尺，重达28t（吨），功耗为170kW，运算速度为每秒5000次的加法运算



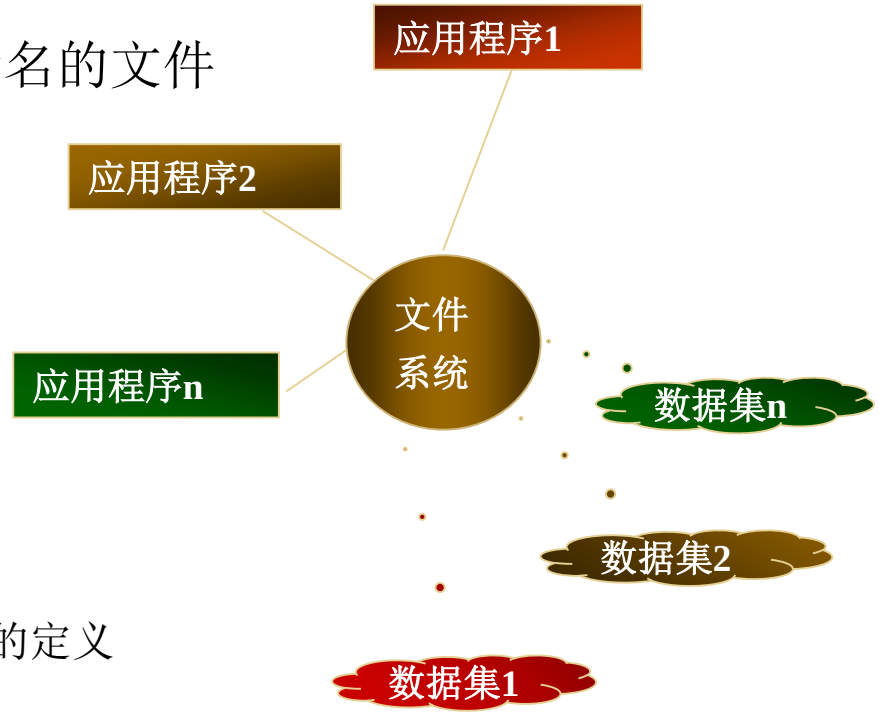
人工管理阶段（~1950中期）

- 背景
  - 外存为顺序存取设备
    - 磁带、卡片、纸带，没有磁盘等直接存取设备。
  - 没有操作系统及数据管理软件
    - 用户用机器指令编码，通过纸带机输入程序和数据，程序运行完毕后，由用户取走纸带和运算结果，再让下一用户操作。
  - 特点
    - **数据不保存**：每个用户使用自己的数据，用完撤走不保存。
    - **应用程序管理数据**：用户通过特定的应用程序完全负责数据管理工作，包括数据的组织、存储结构、存取方法、输入输出等。
    - **数据不共享**：一组数据只能对应一组程序。
    - **数据不具有独立性**：程序中存取数据的子程序随着存储结构的改变而改变。

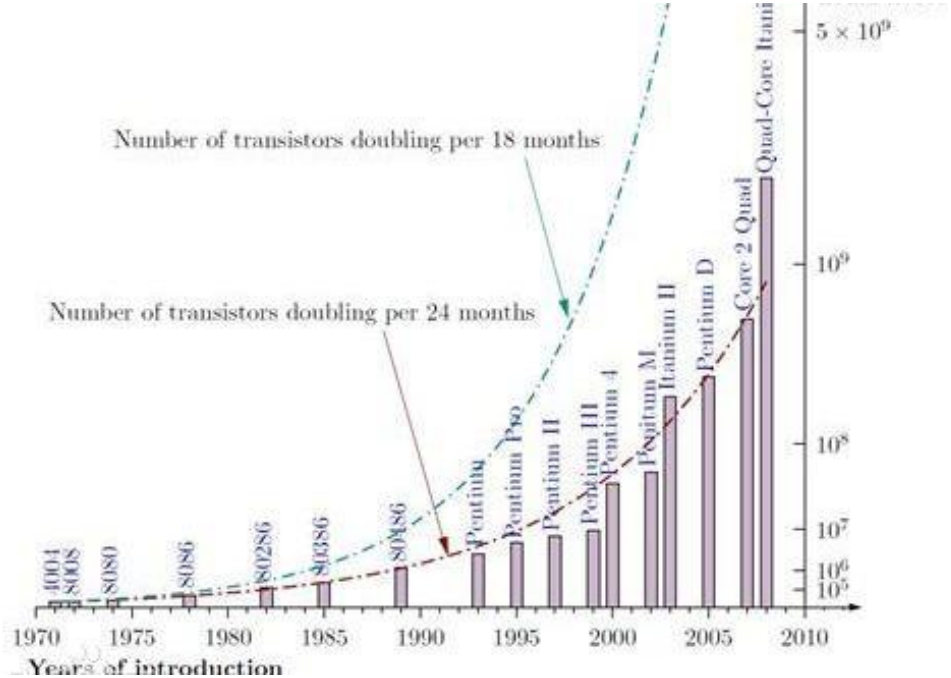


文件系统阶段（1950中期~1960中期）

- 背景
  - 计算机不但用于科学计算，还用于管理
  - 外存有了磁盘、磁鼓等直接存取设备
  - 有了专门管理数据的软件，一般称为文件系统
    - 在文件系统中，数据按其内容、结构和用途组成若干命名的文件
  - 特点
    - 数据可以长期保存
    - 共享性差 （一个文件对应一个程序）
      - 冗余、不一致
    - 数据与程序有一定的独立性
      - 文件的逻辑结构与存储结构由系统进行转换
      - 数据在存储上的改变不一定反映在程序上
      - 文件的逻辑结构改变时，应用程序必须改变，同时修改文件结构的定义



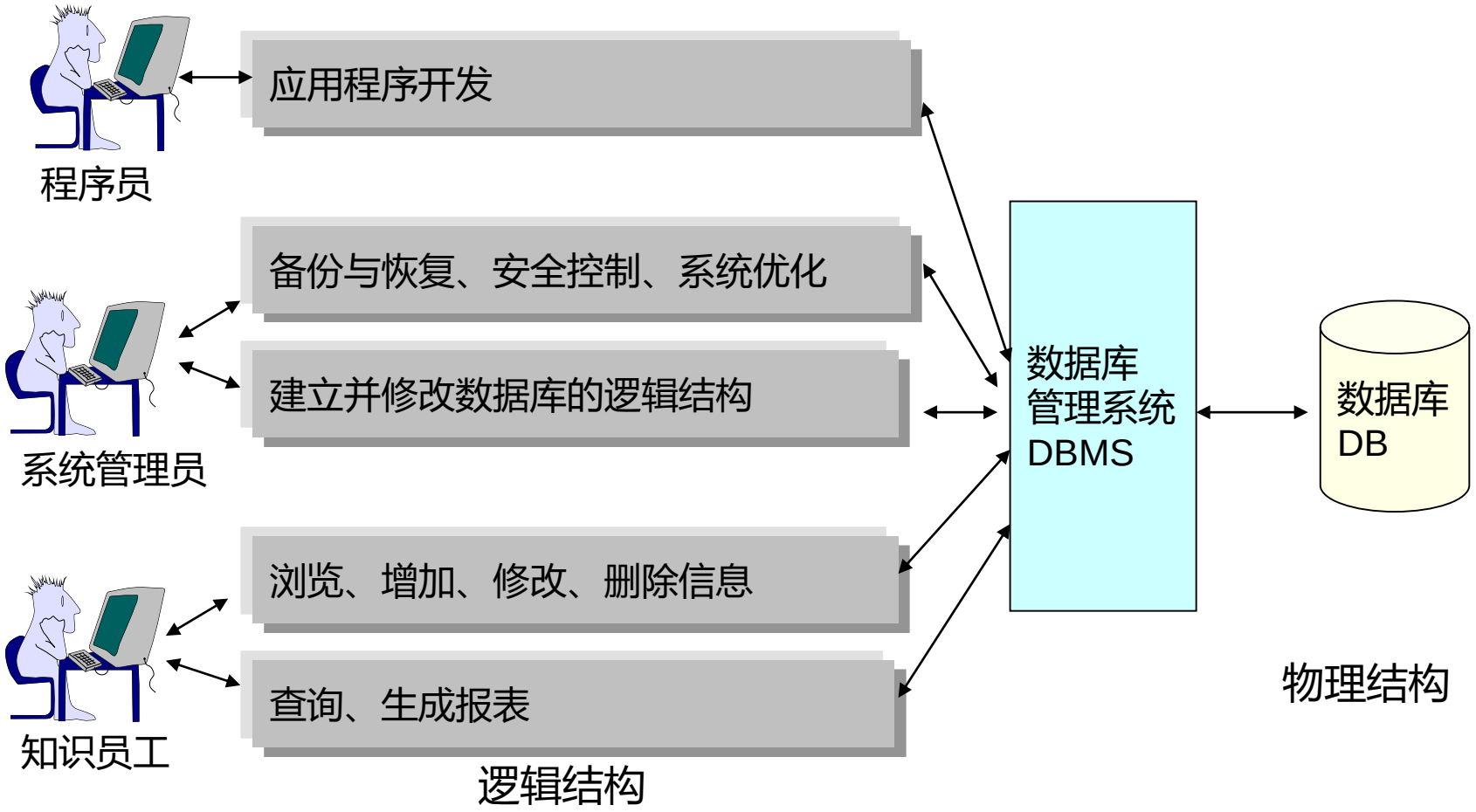
- 背景
  - 计算机管理的数据量大，关系复杂，共享性要求强（多种应用、不同语言共享数据）
  - 外存有了大容量磁盘，光盘
  - 软件价格上升，硬件价格下降
    - 摩尔定律



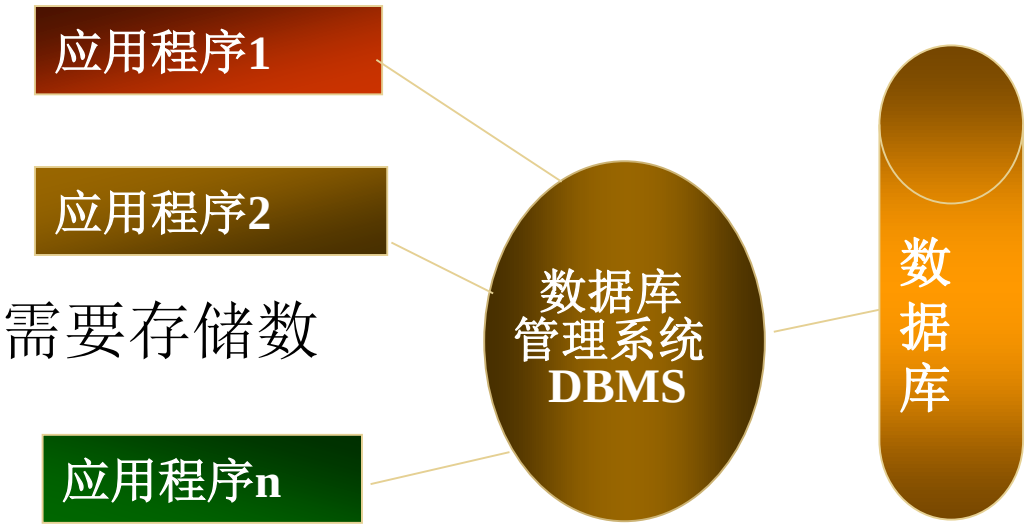
- 数据库（DataBase，DB）
  - 长期储存在计算机内、有组织的、可共享的相关信息的集合。
- 数据库的特点
  - 永久存储
  - 有组织
  - 可共享

- 数据库管理系统（DataBase Management System，DBMS）
  - 是介于用户与操作系统之间的一层数据管理软件。
  - 是基于某种数据模型的。
- 数据库管理系统的功能
  - 数据定义
  - 数据组织、存储和管理
  - 数据操纵
  - 数据库的事务管理和运行管理
  - 数据库的建立和维护
  - 其他

- 数据库和DBMS的关系



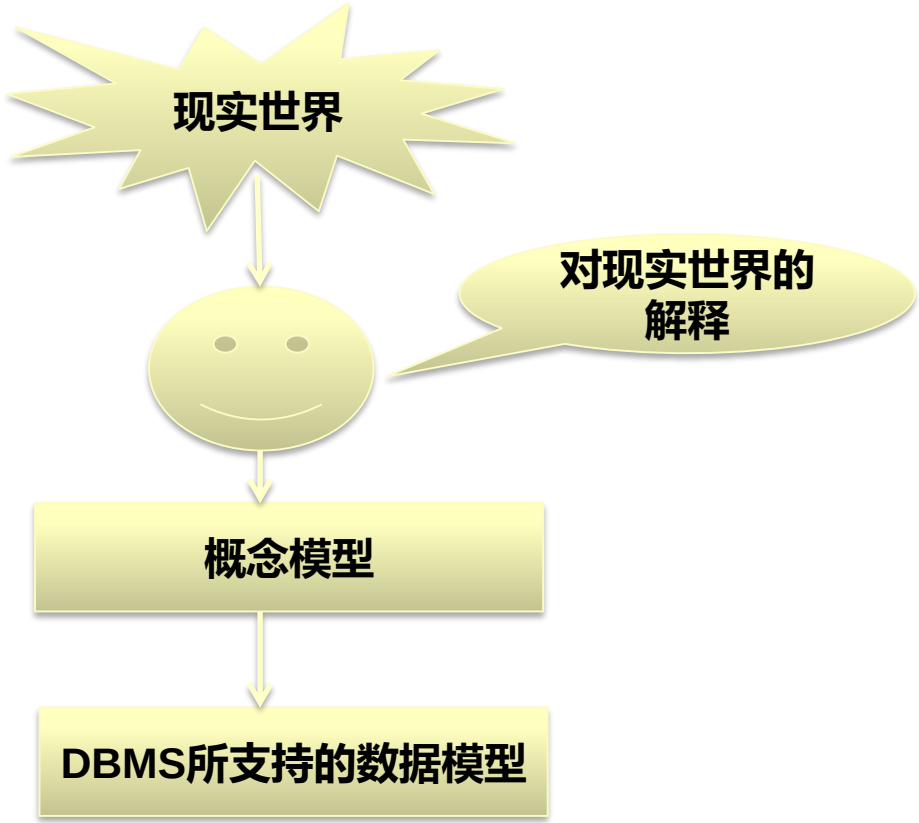
- 优点
  - 数据有整体的结构性，面向全组织，面向现实世界
  - 由数据库管理系统DBMS统一存取，维护数据语义及结构
  - 数据共享性好
  - 数据与程序完全相互独立
- 缺点
  - 数据库系统的设计非常复杂、困难并且耗时，需要存储数据间的关系
  - 需要对所有的程序员和用户进行训练
  - 需要为软硬件支付一定的成本
    - DB2、Oracle数十万美元起，MySQL数十万RMB起
  - 单个程序所需运行时间较长



数据管理技术的比较

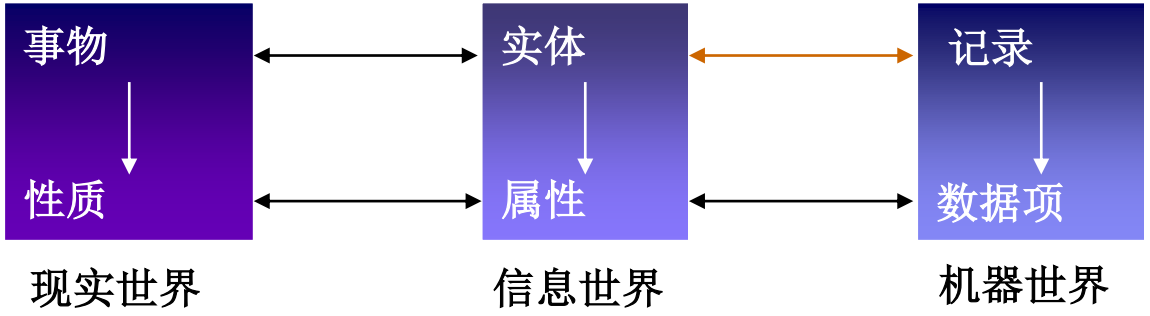
	人工	文件系统	数据库
硬件	无存储设备	硬盘	大容量硬盘
软件	无操作系统	文件系统	**DBMS**
数据管理者	用户	文件系统	**DBMS**
面向	应用程序	应用程序	组织
数据共享	**No**	低	高
数据独立性	**No**	低	高
结构化	**No**	半结构化	结构化
控制	应用程序	应用程序	**DBMS**

- 数据模型（Data Model）是对现实世界各种事物特征的数字化的模拟和抽象
- 数据模型应能满足三个方面的要求：
  - 能比较真实地模拟现实世界
  - 容易为人所理解
  - 便于在计算机上实现



- 现实世界
  - 即客观世界，产生最原始的数据
- 信息世界
  - 是现实世界在人们头脑中反映并用文字或符号记载下来，是人对现实世界的认识抽象过程，经选择、命名、分类等抽象工作后进入信息世界
- 机器世界
  - 用数据模型描述现实世界中的事物及其联系

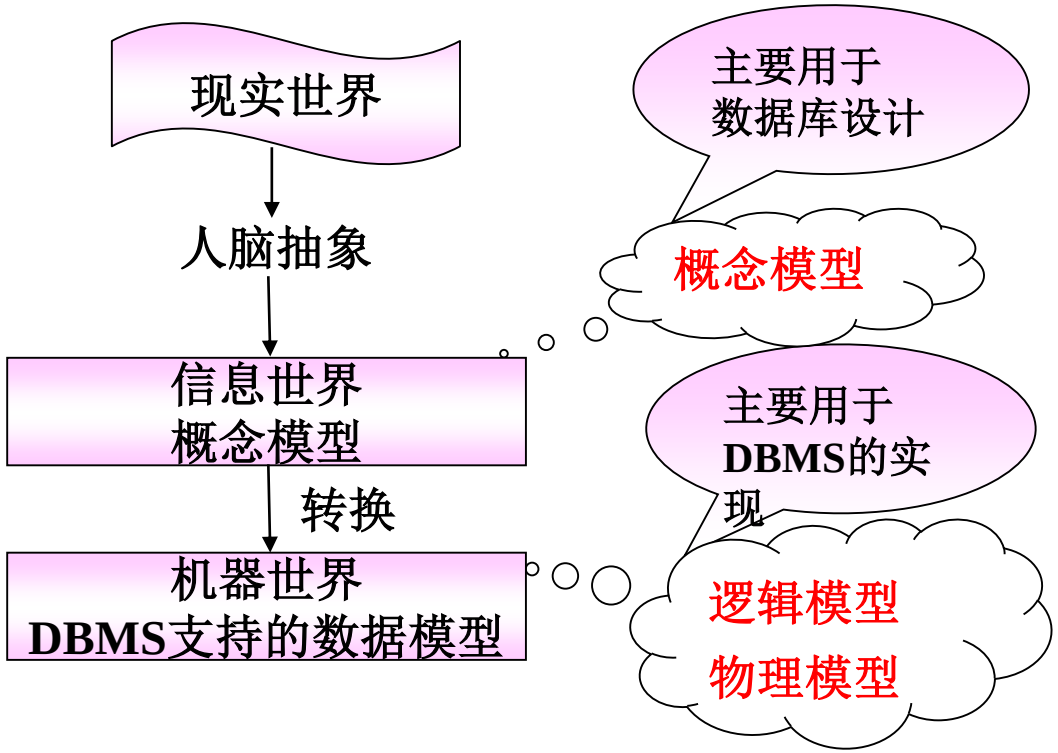
从现实世界到机器世界



黑板

颜色是黑色材料是木材形状是矩形尺寸是长3.2米， 高1.4米

黑色、木质、矩形、3.2m * 1.4m



- 概念模型（Conceptual Model）
  - 又称信息模型、语义模型，是按用户的观点来对数据和信息建模。
  - 利用实体、联系和约束描述现实世界的静态、动态和时态特征。
  - 不注重数据的物理组织结构，着重表示数据模拟的语义，使数据模型更接近现实世界，便于用户理解。
  - 常用的概念模型包括：
    - 实体-联系模型（E-R模型）
    - 面向对象数据模型
    - 函数数据模型
    - 扩展关系模型
    - 语义关联模型

- 逻辑模型
  - 按计算机系统的观点对数据建模
  - 主要包括：
    - 层次模型（Hierarchical Model）
    - 网状模型（Network Model）
    - 关系模型（Relational Model）
    - 面向对象模型（Object Oriented Model）
    - 对象关系模型（Object Relational Model）
- 物理模型
  - 对数据最低层的抽象
  - 描述数据在系统内部的表示方式和存取方法，在磁盘上的存储方式和存取方法
  - 面向计算机系统，由DBMS实现

数据模型的组成要素

- 数据结构 —— 描述系统的静态特性
  - 所研究的对象类型（Object type）的集合，包括：数据的类型、内容和性质的对象（事物）；数据之间联系的对象（联系）。
- 数据操作 —— 描述系统的动态特性
  - 对数据库中的各种对象的实例（值）允许执行的操作的集合。主要有检索和更新（插入、删除、修改）两类操作。
- 完整性约束 —— 完整性规则（条件）的集合
  - 给出数据及其联系所具有的制约、依赖和存储规则，用于限定数据库的状态和状态变化，保证数据库中的数据的正确、有效、完全和相容。

- 概念模型最常用的表示方法
  - 实体-联系方法（Entity-Relationship Approach，ER模型）
  - P.P.S. Chen（陈平山）于1976年提出
  - 该模型直接从现实世界中抽象出实体类型及实体间联系，然后用E-R图表示现实世界的概念模型

基本概念

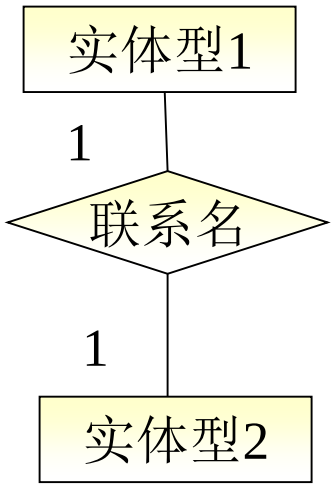
- 实体（entity）：客观存在，可以相互区别的东西称为实体。可以是具体的对象，也可以是抽象的事件。
- 实体集（entity set）：性质相同的同类实体的集合。
- 属性（attribute）：实体的某一方面的特征
- 属性域（domain）：属性的取值范围；含值的类型
- 码（key）：唯一标识每个实体的属性或属性集
- 实体型（entity type）：某一实体属性的集合
- 联系（relationship）：现实世界中事物内部以及事物之间的联系在信息世界中反映为实体内部的联系和实体之间的联系

- 实体间的联系

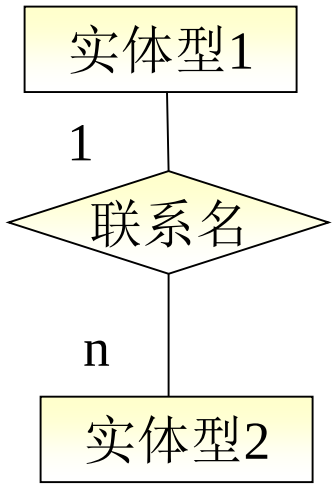
两个实体型
三个实体型
一个实体型

一对一联系（1:1）
一对多联系（1:n）
多对多联系（m:n）

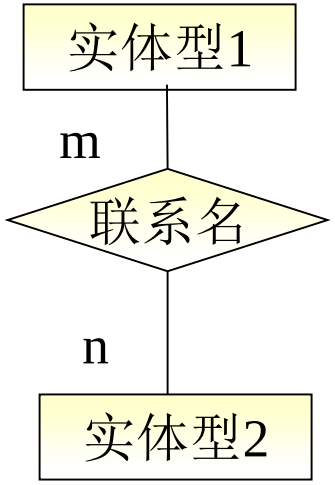
- 两个实体型之间的联系
  - 一对一的联系，记为 1：1
  - 一对多的联系，记为 1：n
  - 多对多的联系，记为 m：n



1:1联系



1:n联系



m:n联系

- 一对一联系
  - 实体集E_1中每个实体至多和实体集E_2中一个实体有联系，反之亦然，记为1：1
- 一对多联系
  - 实体集E_1中的每个实体与实体集E_2中任意个实体有联系，而E_2中每个实体至多和E_1中的一个实体有联系，记为1：n
- 多对多联系
  - 实体集E_1中的每个实体与实体集E_2中任意个实体有联系，反之亦然，记为m：n

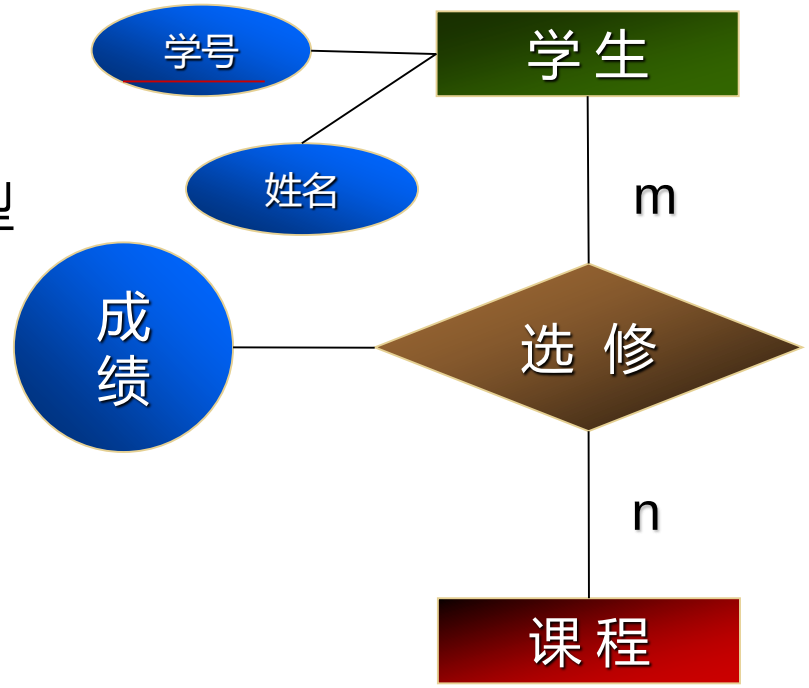
ER图的四个基本成分

实体名　　矩形框表示实体型

属性名　　椭圆形表示属性

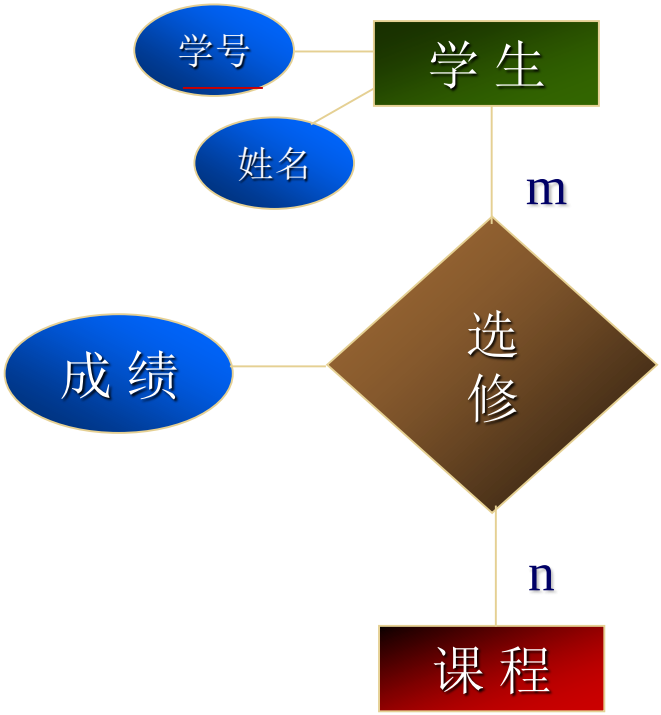
联系名　　菱形表示联系

 　　连接实体型与联系类型，也可用于表示实体与属性的联系并注明种类；对构成码的属性，在属性名下画一横线表示。

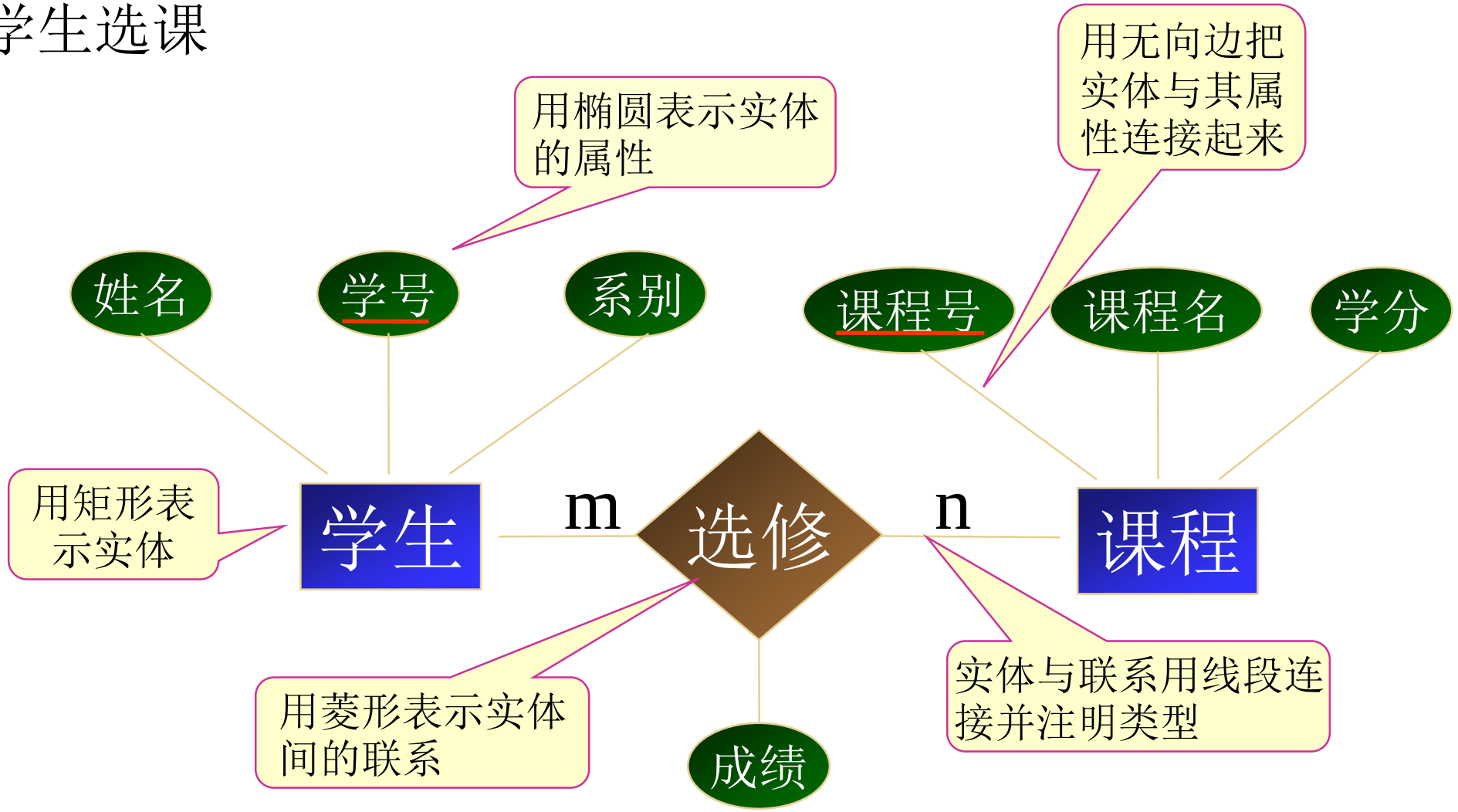


ER图设计过程

- 首先确定实体类型：几个实体类型及相应的实体名
- 确定联系类型：各实体类型之间是否有联系，是何种联系类型及相应的联系名
- 连接实体类型和联系类型，组合成E-R图
- 确定实体类型和联系类型的属性
- 确定实体类型的码
- 实体名和联系名均不能重复!



- 例：学生选课

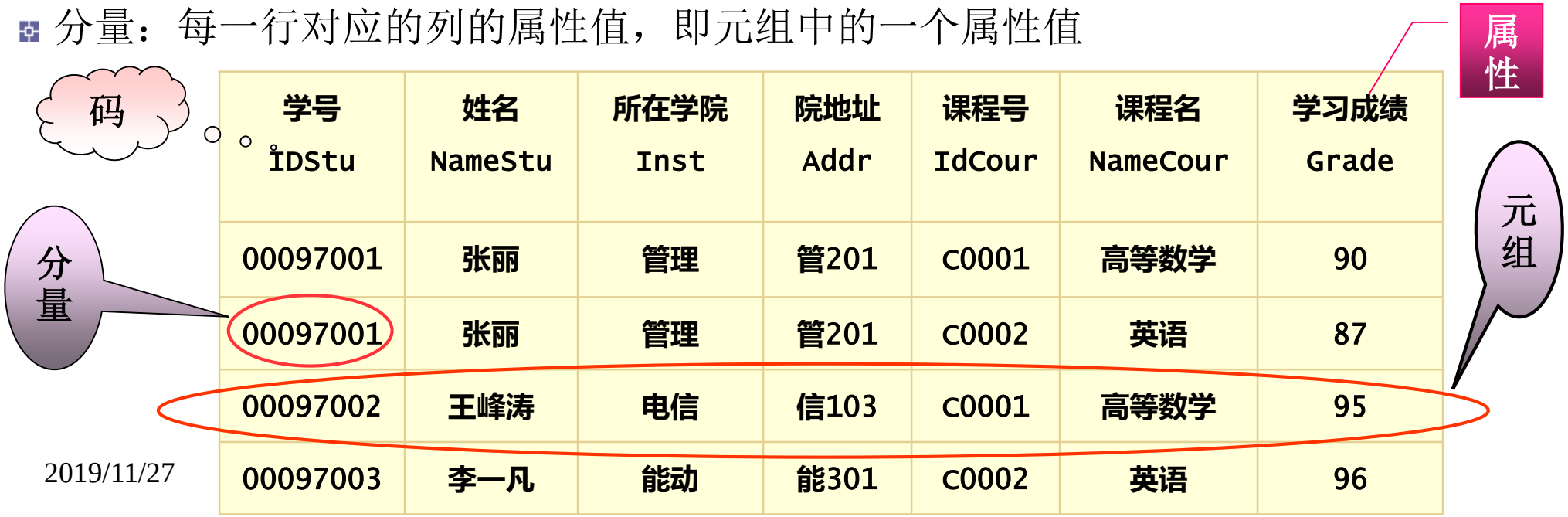

用椭圆表示实体的属性
用无向边把实体与其属性连接起来
姓名
学号
系别
课程号
课程名
学分
用矩形表示实体
学生
m
选修
n
课程
用菱形表示实体间的联系
成绩
实体与联系用线段连接并注明类型

- 实体、实体之间的联系均由二维表来表示
- 二维表称为**关系**

SC

学号 IDStu	姓名 NameStu	所在学院 Inst	院地址 Addr	课程号 IdCour	课程名 NameCour	学习成绩 Grade
00097001	张丽	管理	管201	C0001	高等数学	90
00097001	张丽	管理	管201	C0002	英语	87
00097002	王峰涛	电信	信103	C0001	高等数学	95
00097003	李一凡	能动	能301	C0002	英语	96

- 二维表，由行与列构成。
  - 元组(Tuple)：在二维表中的一行，称为一个元组
  - 属性(Attribute)：在二维表中的列，称为属性
  - 码(Key)：如果在一个关系中存在唯一标识一个实体的一个属性或属性集称为实体的键，即使得在该关系的任何一个关系状态中的两个元组，在该属性上的值的组合都不同
  - 域(Domain)：属性值的取值范围为值域
  - 分量：每一行对应的列的属性值，即元组中的一个属性值



学号 IDStu	姓名 NameStu	所在学院 Inst	院地址 Addr	课程号 IdCour	课程名 NameCour	学习成绩 Grade
00097001	张丽	管理	管201	C0001	高等数学	90
00097001	张丽	管理	管201	C0002	英语	87
00097002	王峰涛	电信	信103	C0001	高等数学	95
00097003	李一凡	能动	能301	C0002	英语	96

码

- 主键（Primary Key，PK）
  - 用来区别一张关系表内的不同元组
- 外键（Foreign Key，FK）
  - 用于与另一张表的关联，是能确定另一张表记录的字段，用于保持数据的一致性

	主键	外键
定义：	唯一标识一条记录，不能有重复的，不允许为空	表的外键是另一表的主键, 外键可以有重复的, 可以是空值
作用：	用来保证数据完整性	用来和其他表建立联系用的
个数：	主键只能有一个	一个表可以有多个外键

- 对关系的描述称为**关系模式**，一般表示为：
  - 关系名（属性1，属性2，…，属性n）

学号 IDStu	姓名 NameStu	所在学院 Inst	院地址 Addr	课程号 IdCour	课程名 NameCour	学习成绩 Grade
00097001	张丽	管理	管201	C0001	高等数学	90
00097001	张丽	管理	管201	C0002	英语	87
00097002	王峰涛	电信	信103	C0001	高等数学	95
00097003	李一凡	能动	能301	C0002	英语	96

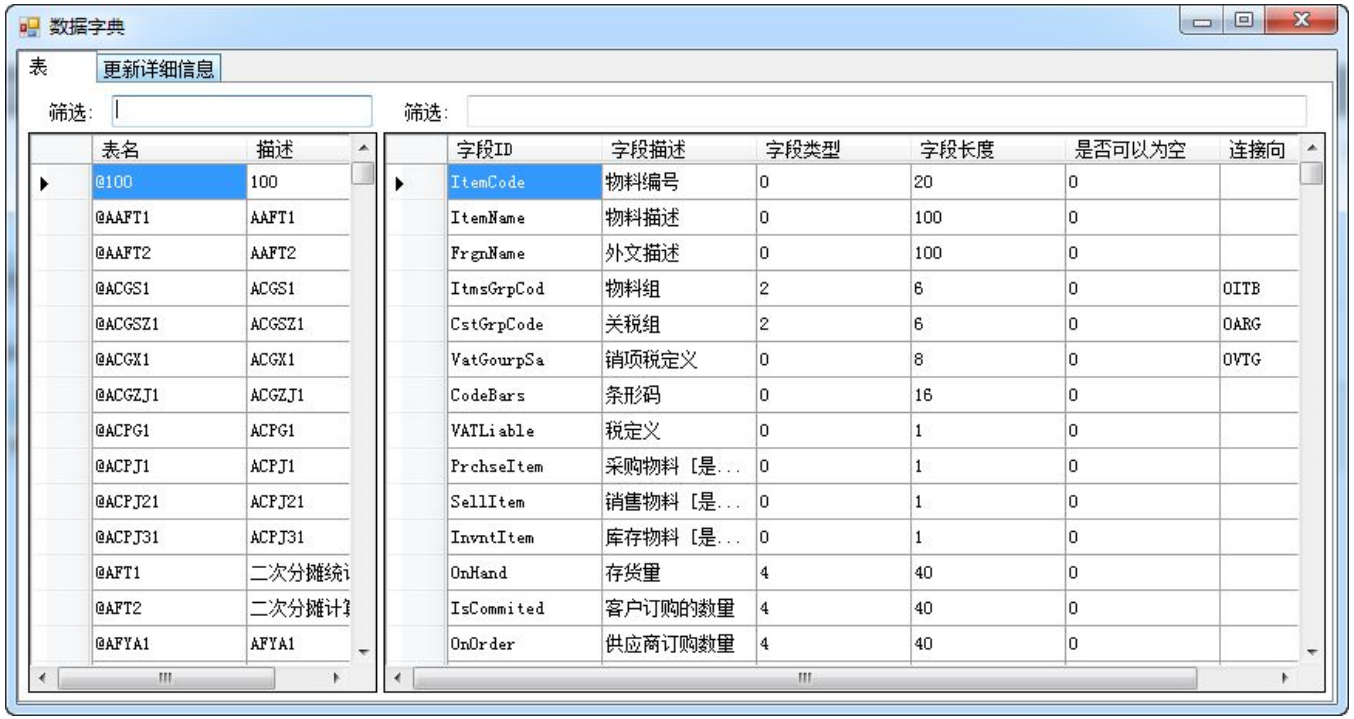
学生课程（学号，姓名，所在学院，院地址，课程号，课程名，学习成绩）

- 为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。
  - **第一范式(确保每列保持原子性)**
    - 数据库表中的所有字段值都是不可分解的原子值
  - **第二范式(确保表中的每列都和主键相关)**
    - 第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）
    - 在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中
      - 比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。
  - **第三范式(确保每列都和主键列直接相关,而不是间接相关)**
    - 每一列数据都和主键直接相关，而不能间接相关
      - 比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段

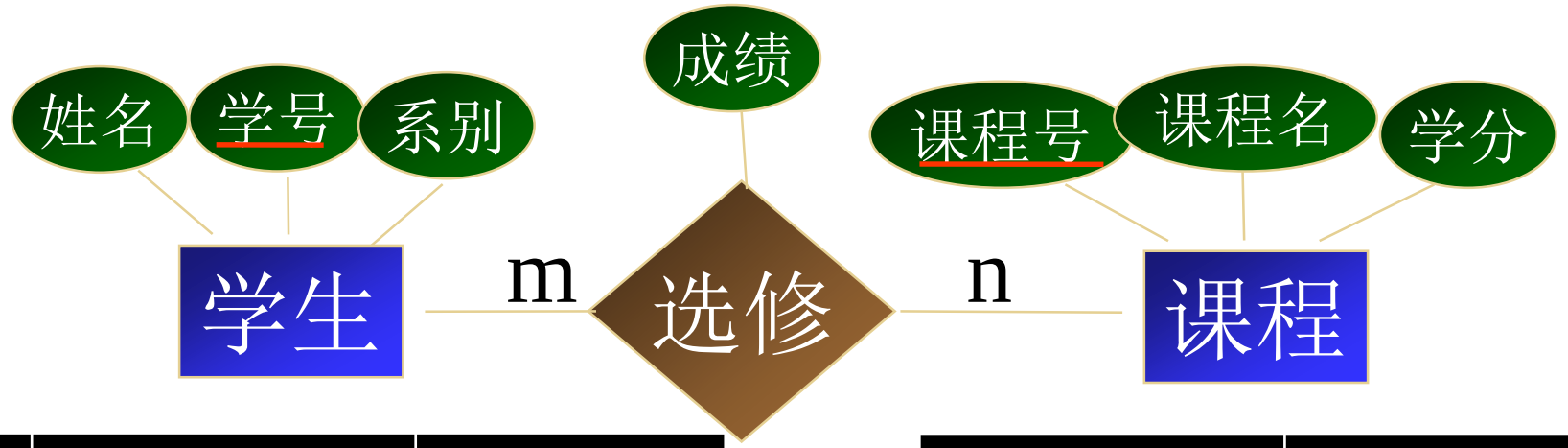
- 数据操作
  - 关系模型的数据操作主要包括：查询、插入、删除和修改数据
  - 数据操作是集合操作，操作对象和操作结果都是关系，即若干元组的集合
  - 存取路径对用户隐蔽，用户只要指出“干什么”或“找什么”，不必详细说明“怎么干”或“怎么找”
- 完整性约束
  - 实体完整性
    - 实体完整性(Entity integrity)是指关系的主关键字不能重复也不能取“空值"
  - 参照完整性
    - 参照完整性(Referential Integrity)是定义建立关系之间联系的主关键字与外部关键字引用的约束条件。
  - 用户定义完整性
    - 用户定义完整性(user defined integrity)则是根据应用环境的要求和实际的需要，对某一具体应用所涉及的数据提出约束性条件
- 存储结构
  - 在DB的物理组织中，表以文件形式存储

- 优点
  - 严格的数学概念
  - 结构单一、简单
  - 用户易懂易用存取路径透明
  - 数据独立性强
  - 安全性高
  - 易于开发
- 缺点
  - 查询效率低

- 数据字典是指对数据的数据项、数据结构、数据流、数据存储、处理逻辑、外部实体等进行定义和描述，其目的是对数据流程图中的各个元素做出详细的说明，使用数据字典为简单的建模项目。



- 对每个实体（Entity）均建立一张关系表
- 对每个关系（Relationship）也同样建立一张关系表
  - 但是，有时候为了在性能和规范中取得平衡，并不一定要这么做

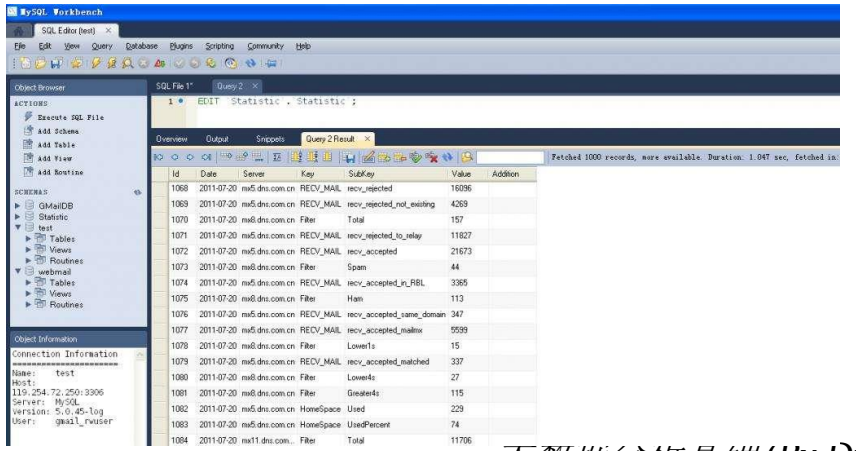


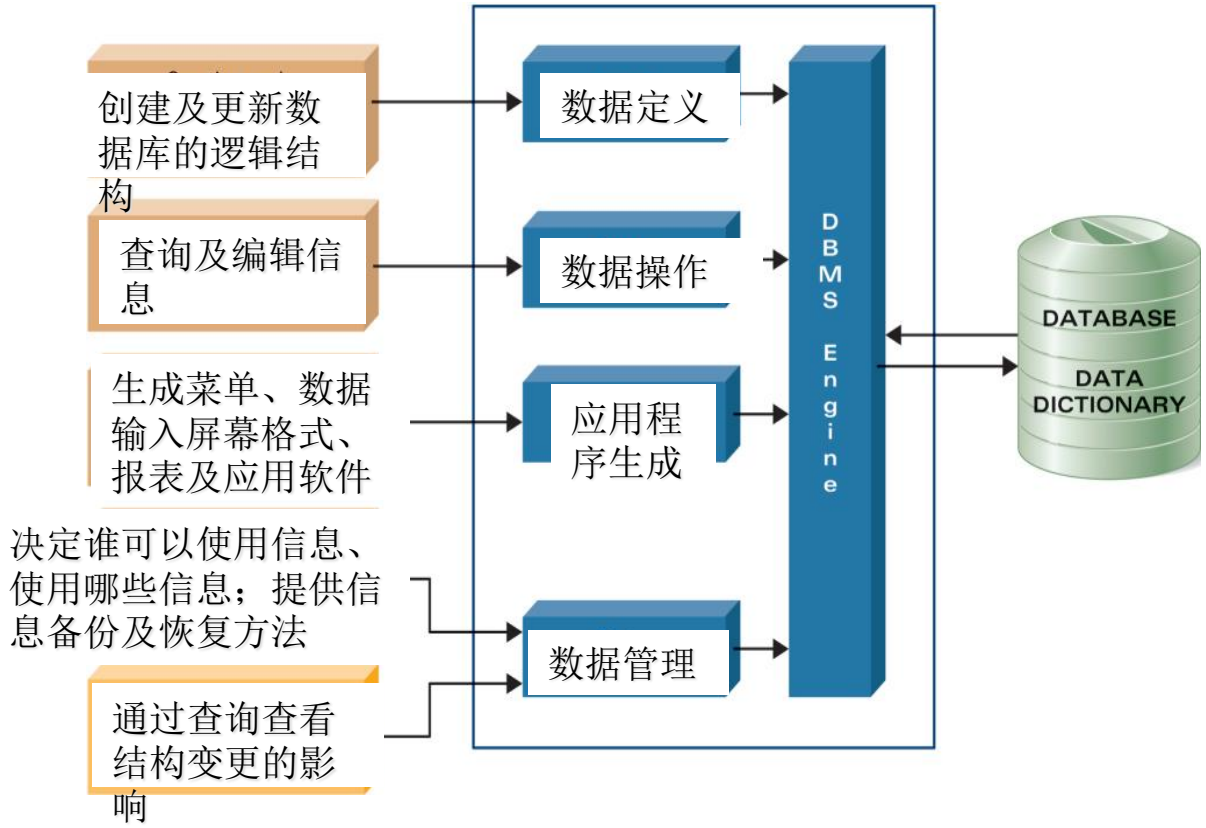
姓名	学号（PK）	系别
张三	35620191150001	物流工程

课程名	课程号（PK）	学分
大数据分析基础	10010	2

课程号（PK/FK）	学号（PK/FK）	成绩
10010	35620191150001	99

- 数据库管理系统（Database Management Systems，DBMS）工具帮助指定数据库逻辑需求，在数据库中访问和使用信息
- DBMS有5个重要的组成部分
  - DBMS引擎
  - 数据定义子系统
  - 数据操作子系统
  - 应用程序生成子系统
  - 数据管理子系统



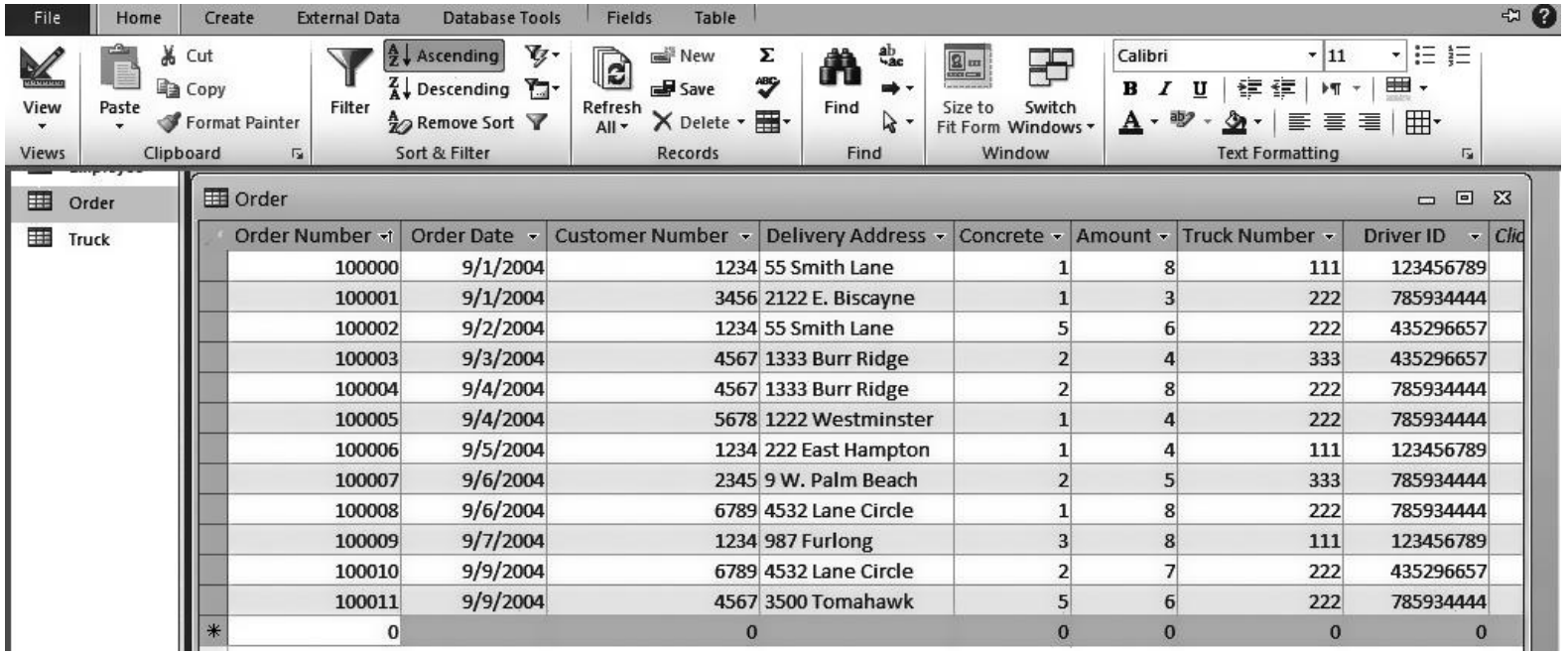


- 数据库引擎是用于存储、处理和保护数据的核心服务
  - 接受来自其他各DBMS子系统道的逻辑查询请求，并将逻辑查询请求转换成其对应的物理形式
  - 数据库的引擎决定了数据库系统的性能
  - 不同的引擎有着不同的特点和适用环境
    - MySQL引擎：MyISAM、InnoDB、MERGE、MEMORY(HEAP)、BDB(BerkeleyDB)、EXAMPLE、FEDERATED、ARCHIVE、CSV、BLACKHOLE
- 物理视图：信息在存储设备上怎样进行物理排列、存储和读取。
- 逻辑视图：关注知识工作者要如何排列和存取信息，以满足其特定的业务需求。
- DBMS引擎将这两者进行了隔离

- 数据定义子系统—帮助人们在数据库中建立并维护数据字典，以及定义数据库中的文件结构
- 数据字典帮助定义
  - 字段名称
  - 数据类型（数字型等）
  - 格式（电话号码前是否有区号）
  - 缺省值
  - 有效范围
  - 输入约束等

- 数据操作子系统—帮助用户对数据库中的信息进行增加、修改和删除，并帮助用户在数据库中查询有价值的信息
- 是最主要的交互界面
- 数据操作工具包括：
  - 视图 (view)
  - 报表生成器 (report generators)
  - 范例查询工具 (Query-by-example tools)
  - 结构化查询语言（SQL）等

- 视图允许用户查看数据库文件的内容，对其进行必要的修改，完成简单的分类，并通过查找操作得到具体信息的位置
  - 视图实际上是基于查询的一个虚拟表
  - 它并不会存储在数据库中，只会在需要的时候查询并显示

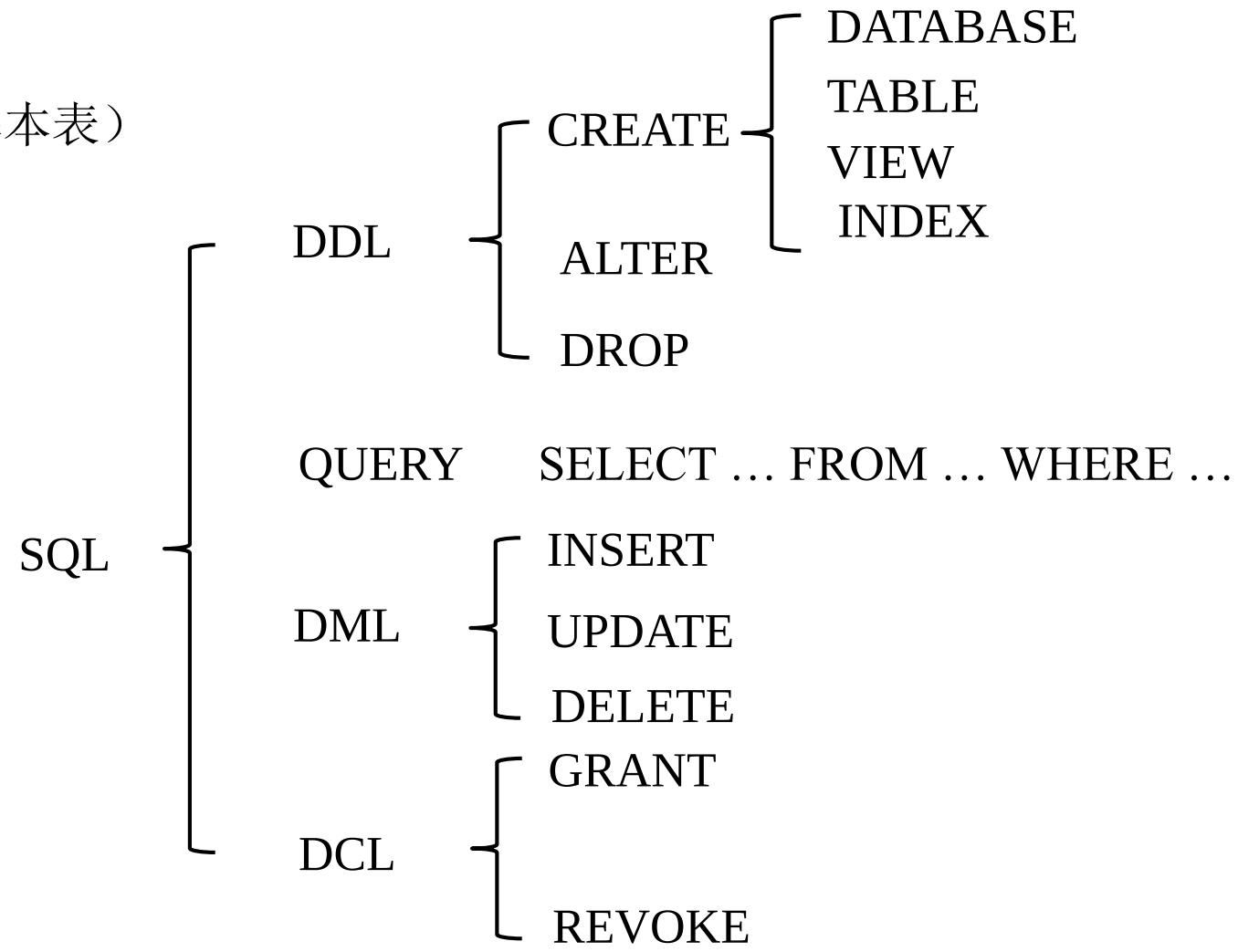


Order

Order Number	Order Date	Customer Number	Delivery Address	Concrete	Amount	Truck Number	Driver ID
100000	9/1/2004	1234	55 Smith Lane	1	8	111	123456789
100001	9/1/2004	3456	2122 E. Biscayne	1	3	222	785934444
100002	9/2/2004	1234	55 Smith Lane	5	6	222	435296657
100003	9/3/2004	4567	1333 Burr Ridge	2	4	333	435296657
100004	9/4/2004	4567	1333 Burr Ridge	2	8	222	785934444
100005	9/4/2004	5678	1222 Westminster	1	4	222	785934444
100006	9/5/2004	1234	222 East Hampton	1	4	111	123456789
100007	9/6/2004	2345	9 W. Palm Beach	2	5	333	785934444
100008	9/6/2004	6789	4532 Lane Circle	1	8	222	785934444
100009	9/7/2004	1234	987 Furlong	3	8	111	123456789
100010	9/9/2004	6789	4532 Lane Circle	2	7	222	435296657
100011	9/9/2004	4567	3500 Tomahawk	5	6	222	785934444
0		0		0	0	0	0

- 结构化查询语言(Structured Query Language，SQL)，是一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统
- 不过各种通行的数据库系统在其实践过程中都对SQL规范作了某些编改和扩充。所以，实际上不同数据库系统之间的SQL不能完全相互通用

- 数据定义（DDL）
  - 定义、删除、修改关系模式（基本表）
  - 定义、删除视图（View）
  - 定义、删除索引（Index）
- 数据操纵（DML）
  - 数据查询
  - 数据增、删、改
- 数据控制（DCL）
  - 用户访问权限的授予、收回



SQL特点

- 高度非过程化的语言
  - 用户只需提出“干什么”，至于“怎么干”由DBMS解决。
- 面向集合的语言
  - 每一个SQL的操作对象是一个或多个关系，操作的结果也是一个关系。
- 一种语法结构，两种使用方式
  - 即可独立使用，又可嵌入到宿主语言中使用，具有自主型和宿主型两种特点。
- 具有查询、操作、定义和控制四种语言一体化的特点。
- 语言简洁、易学易用
  - 核心功能只有9个动词：SELECT、CREATE、DROP、ALTER、INSERT、UPDATE、DELETE、GRANT、REVOKE，语法简单，接近英语。

- 应用程序生成子系统—包含帮助用户建立面向事务处理的应用程序
  - 建立数据输入屏幕功能
  - 为特定的DBMS选定程序设计语言
  - 建立公共的操作交互界面
- 数据管理子系统—帮助管理整个数据库环境通过提供
  - 备份与恢复
  - 安全管理
  - 查询优化
  - 重新组织
  - 并发控制功能
  - 变动管理

- 事务机制可以确保数据一致性
  - 事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。
    - 原子性（atomicity）：一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
    - 一致性（consistency）：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
    - 隔离性（isolation）：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
    - 持久性（durability）：持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

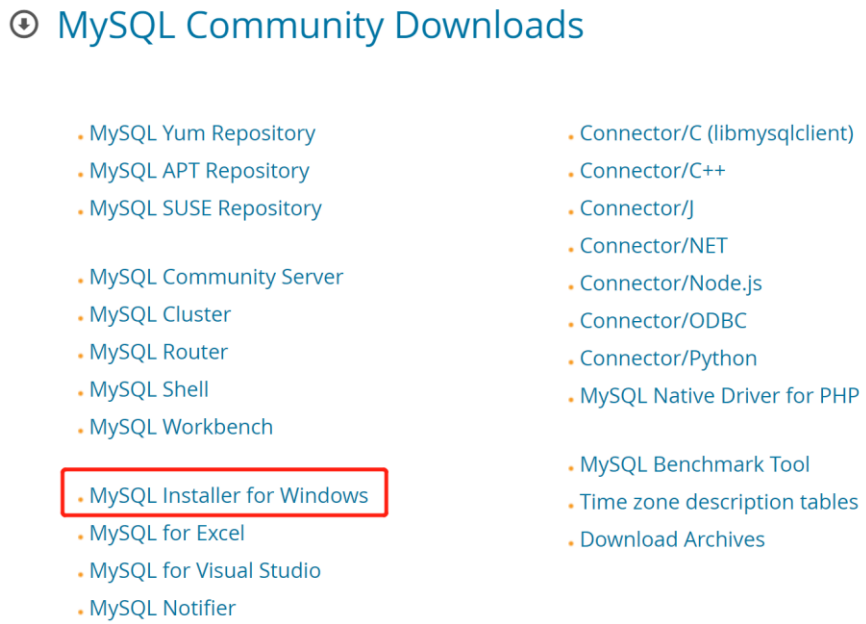
最新数据库流行度排名

357 systems in ranking, November 2019

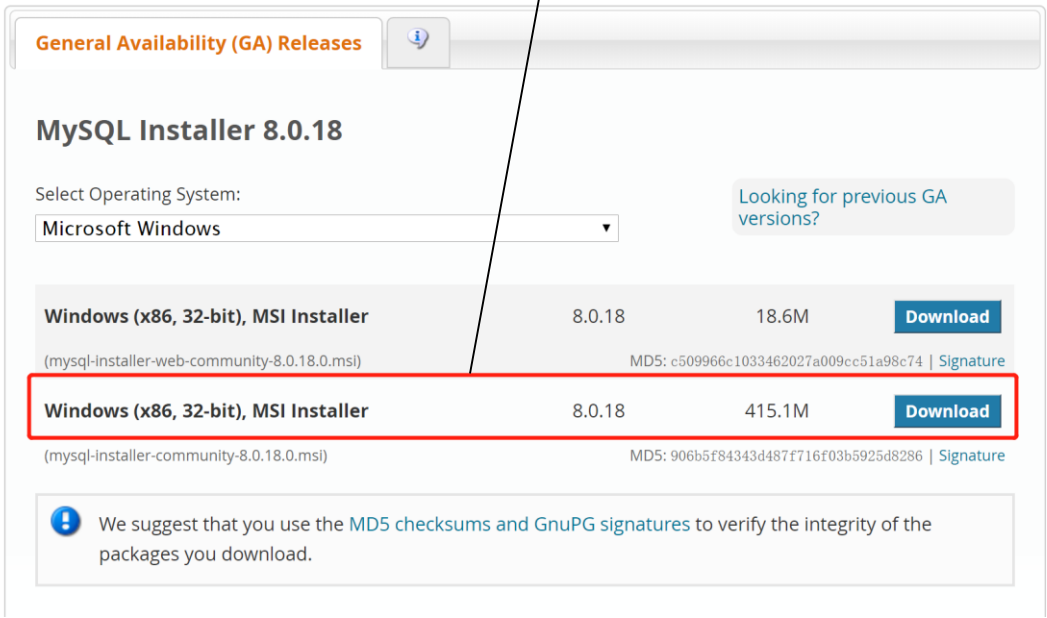
Rank Nov 2019	Rank Oct 2019	Rank Nov 2018	DBMS	Database Model	Score Nov 2019	Score Oct 2019	Score Nov 2018
1.	1.	1.	Oracle	Relational, Multi-model	1336.07	-19.81	+34.96
2.	2.	2.	MySQL	Relational, Multi-model	1266.28	-16.78	+106.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1081.91	-12.81	+30.36
4.	4.	4.	PostgreSQL	Relational, Multi-model	491.07	+7.16	+50.83
5.	5.	5.	MongoDB	Document, Multi-model	413.18	+1.09	+43.70
6.	6.	6.	IBM Db2	Relational, Multi-model	172.60	+1.83	-7.27
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	148.40	-1.77	+4.94
8.	8.	↓ 7.	Redis	Key-value, Multi-model	145.24	+2.32	+1.06
9.	9.	9.	Microsoft Access	Relational	130.07	-1.10	-8.36
10.	10.	↑ 11.	Cassandra	Wide column	123.23	+0.01	+1.48
11.	11.	↓ 10.	SQLite	Relational	121.03	-1.60	-1.68
12.	12.	12.	Splunk	Search engine	89.06	+2.23	+8.69
13.	13.	↑ 14.	MariaDB	Relational, Multi-model	85.57	-1.20	+12.32
14.	14.	↑ 15.	Hive	Relational	84.22	-0.52	+19.65
15.	15.	↓ 13.	Teradata	Relational, Multi-model	80.35	+1.61	+1.04
16.	16.	↑ 21.	Amazon DynamoDB	Multi-model	61.37	+1.19	+7.56
17.	17.	↓ 16.	Solr	Search engine	57.78	+0.21	-3.10
18.	18.	↑ 20.	FileMaker	Relational	55.73	-0.94	-0.02
19.	19.	↓ 18.	SAP Adaptive Server	Relational	55.29	-0.54	-1.27
20.	20.	↓ 19.	SAP HANA	Relational, Multi-model	55.11	-0.24	-0.76

下载

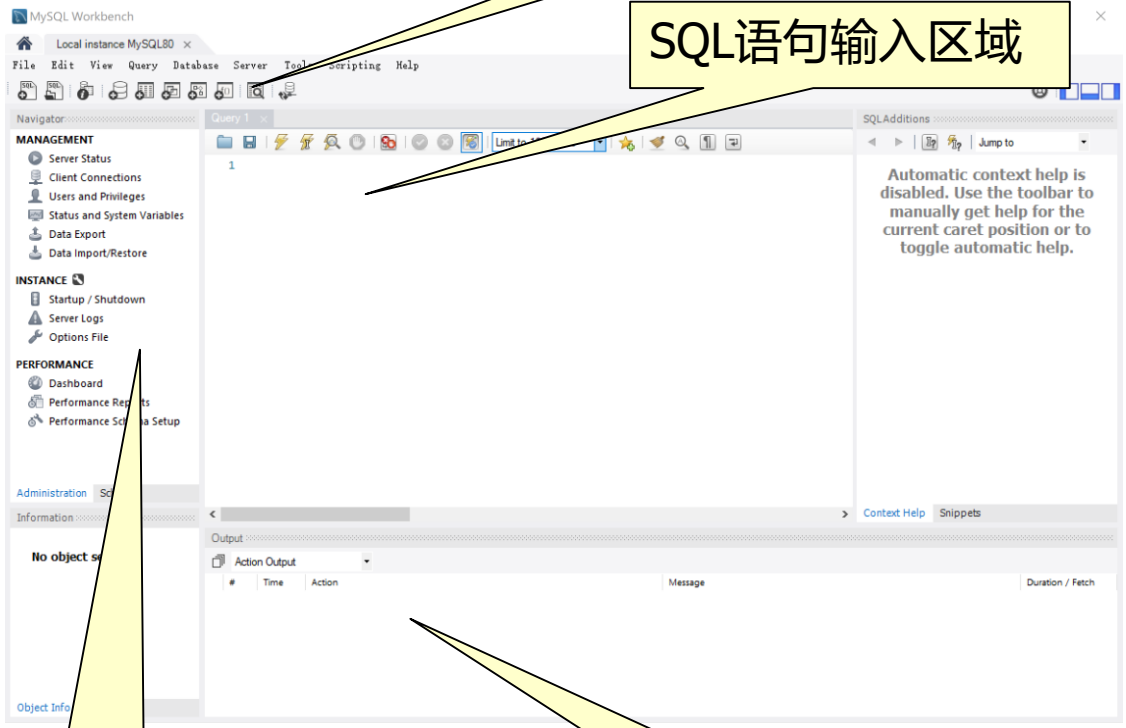
- https://dev.mysql.com/downloads/
- 安装过程基本上按照默认设置就可以

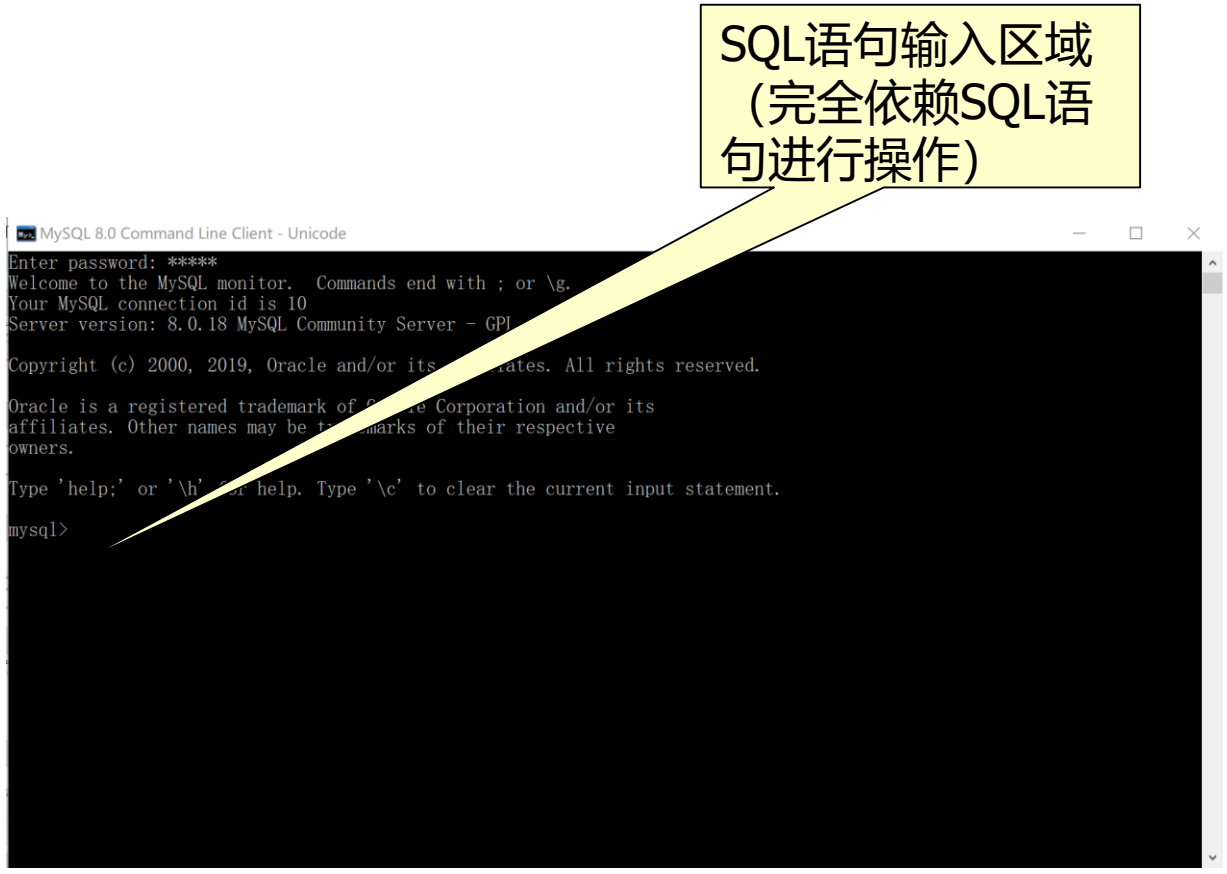


由于网络问题，建议选择直接下载离线安装包

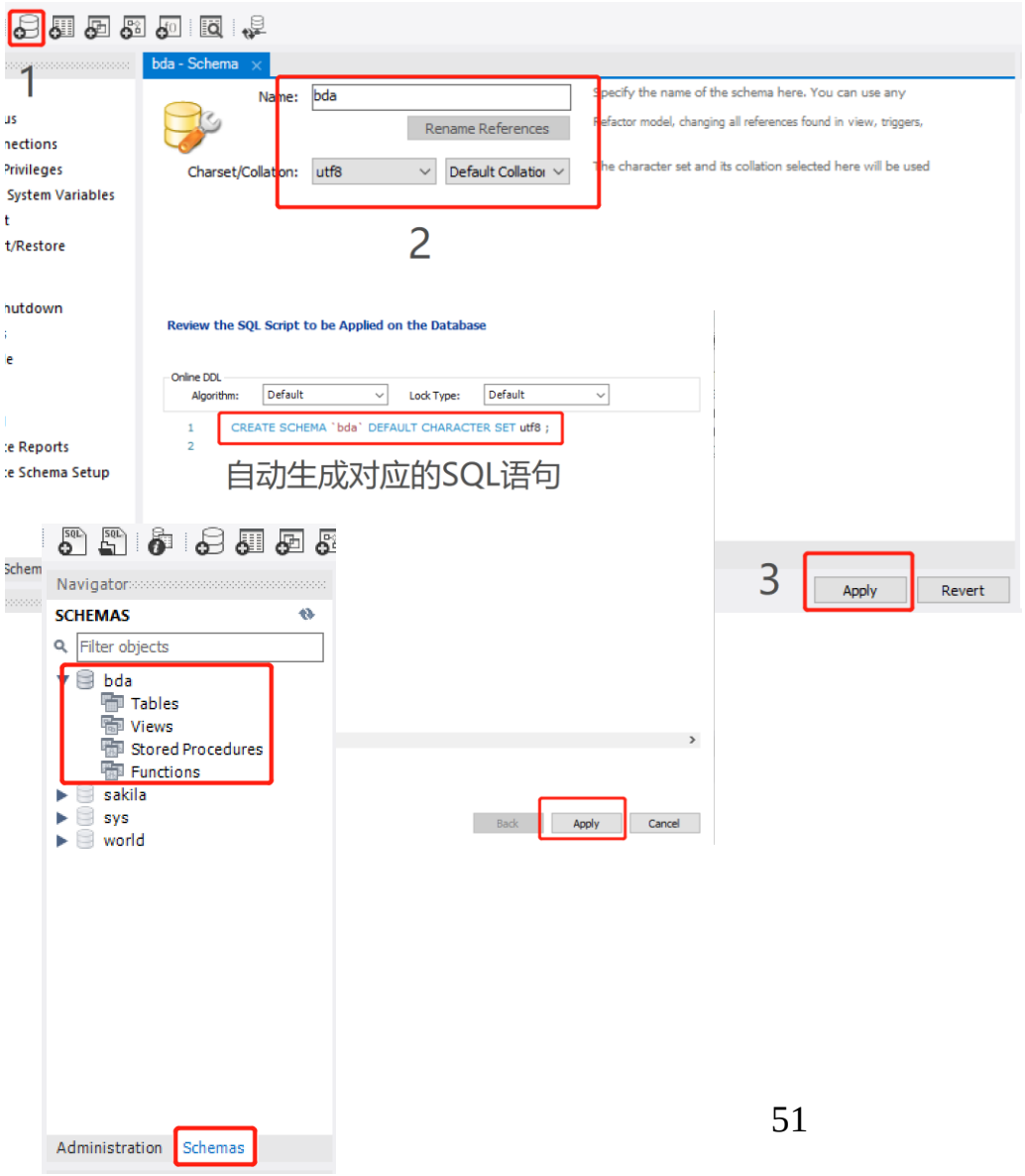


操作界面


工具栏
SQL语句输入区域
数据库展示、管理区域
结果输出区域


SQL语句输入区域（完全依赖SQL语句进行操作）
MySQL 8.0 Command Line Client - Unicode
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.18 MySQL Community Server - GPL
Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>

- 创建数据库
  - 图形化操作（Workbench中操作）
  - SQL（Workbench/Command Line Client）
    - 基本语法：CREATE [IF NOT EXISTS] SCHEMA <数据库名> [[DEFAULT] CHARACTER SET <字符集名>];
      - CREATE SCHEMA `bda` DEFAULT CHARACTER SET utf8 ;
        - []内可选
        - DEFAULT CHARACTER SET utf8用来设定数据库所用的编码集合，可省略
        - MySQL中的SCHEMA和之前所说的DATABASE是一样的，都是建立一个可以包含表、视图等内容的数据库
        - 数据库名中的`可以省略
          - 注意是`(键盘上和~在一起的那个字符)而不是’(单引号)



- 使用数据库
  - 选择某个数据库为当前的默认数据库
    - USE <数据库名>：USE bda;
    - 下一次使用这条命令就可以切换至另外一个数据库
- 查询数据库
  - 查看所有的数据库：SHOW DATABASES;
  - 查看当前使用的数据库：SELECT DATABASE();
  - 查看符合条件的数据库：SHOW DATABASES LIKE <字符串>;
    - 字符串需要用单引号或者双引号括起来
- 删除数据库
  - DROP DATABASE [IF EXISTS] <数据库名>;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| bda                |
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
7 rows in set (0.00 sec)
```

Result Grid | Filter Rov

	Database
▶	bda
	information_schema
	mysql
	performance_schema
	sakila
	sys
	world

```
mysql> show databases like 'bda';
+--------------+
| Database (bda) |
+--------------+
| bda          |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| bda        |
+------------+
1 row in set (0.00 sec)
```

MySQL权限设置

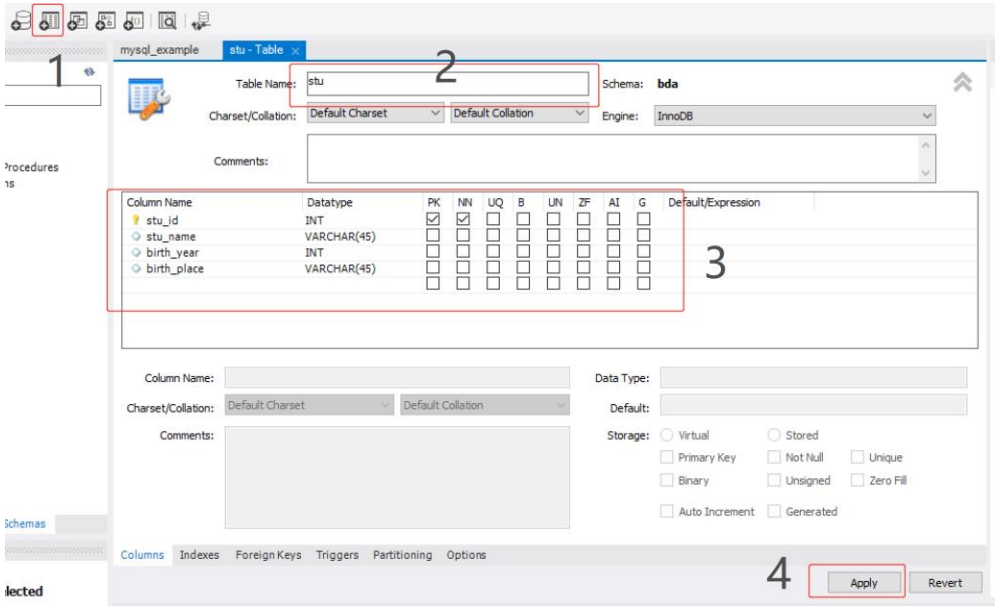
- 如何防止删库跑路？
  - 为不同的用户设置不同权限
  - 权限的等级
    - 整个服务器，使用 GRANT ALL 和REVOKE ALL；整个数据库，使用ON DATABASE.*；特定表，使用ON DATABASE.TABLE；特定的列；特定的存储过程
  - 权限授予
    - GRANT <权限> ON <数据库/表> TO <用户>[@<登录IP或域名> IDENTIFIED BY <登录密码> WITH GRANT OPTION];
      - 权限包括CREATE, ALTER, DROP, SELECT, INSERT, UPDATE, DELETE等27种权限（https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html）
      - %表示没有限制，在任何主机都可以登录
      - WITH GRANT OPTION表示允许用户将自己的权限授权给其它用户
      - GRANT SELECT ON BDA.* TO STU;
  - 权限回收
    - REVOKE <权限> ON <数据库/表> FROM <用户>

创建表

图形化操作

SQL

- CREATE TABLE <表名> ([表定义选项]) [表选项] [分区选项];
  - 其中表定义选项的格式为：<列名1> <类型1> [,…] <列名n> <类型n>
  - <表定义选项>：表创建定义，由列名、列的定义以及可能的空值说明、完整性约束或表索引组成。
  - 表选项用于对表指定存储引擎（ENGINE = <引擎>）、添加表的注释（COMMENT <注释>）等。
  - 分区选项（partition_options）用于将大数据表分区存储，以实现性能优化；分区仅在底层实现，具体对表进行操作时候不能对单个子表进行操作。

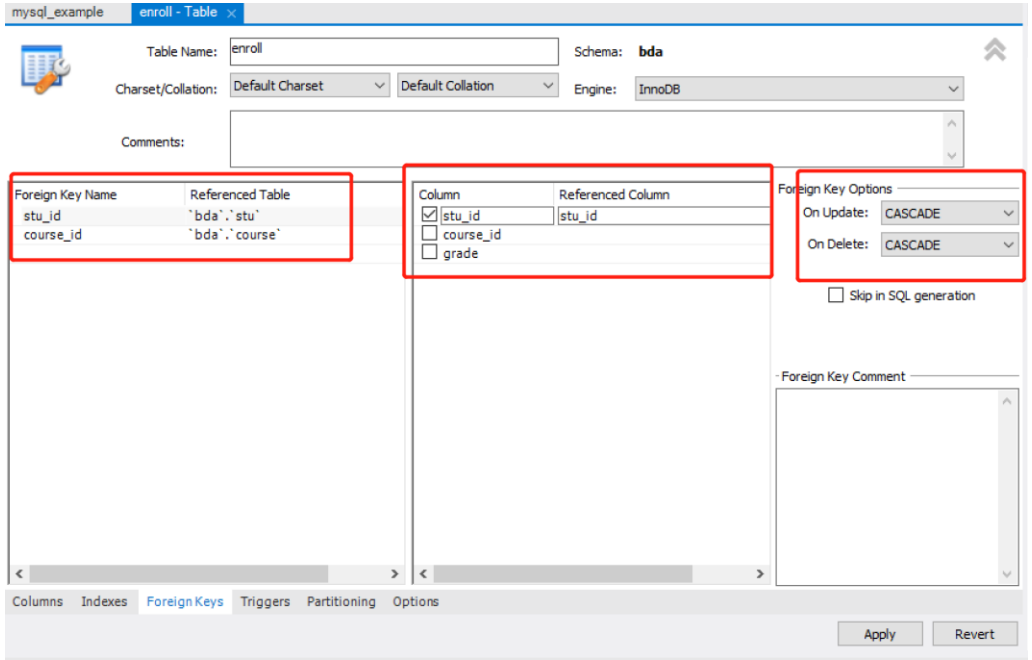



```
CREATE TABLE `bda`.`stu` ( `stu_id` INT NOT NULL, `stu_name` VARCHAR(45) NULL, `birth_year` INT NULL, `birth_place` VARCHAR(45) NULL, PRIMARY KEY (`stu_id`));
```

创建具有外键的表

外键选项

- CASCADE
  - 在父表上update/delete记录时，同步update/delete掉子表的匹配记录
- SET NULL
  - 在父表上update/delete记录时，将子表上匹配记录的列设为null (要注意子表的外键列不能为not null)
- NO ACTION
  - 如果子表中有匹配的记录，则不允许对父表对应候选键进行update/delete操作
- RESTRICT
  - 同no action，都是立即检查外键约束




```
CREATE TABLE `bda`.`enroll` ( `stu_id` INT NOT NULL,
`course_id` INT NOT NULL, `grade` INT NULL, PRIMARY
KEY (`stu_id`, `course_id`), INDEX `course_id_idx`
(`course_id` ASC) VISIBLE, CONSTRAINT `stu_id`
FOREIGN KEY (`stu_id`) REFERENCES `bda`.`stu`
(`stu_id`) ON DELETE CASCADE ON UPDATE
CASCADE, CONSTRAINT `course_id` FOREIGN KEY
(`course_id`) REFERENCES `bda`.`course` (`course_id`)
ON DELETE CASCADE ON UPDATE CASCADE);
```

- 修改表
  - ALTER TABLE <表名> [ADD <新列名> <数据类型> [完整性约束]] [DROP {<新列名> |<完整性约束名>}] [MODIFY <列名> <数据类型>];
    - ADD子句：增加新列和新的完整性约束条件
      - 不论基本表中原来是否已有数据，新增加的列一律为空值
    - DROP子句：删除指定的列或者完整性约束条件
    - MODIFY子句：用于修改列名和数据类型
      - 修改原有的列定义有可能会破坏已有数据
- 删除表
  - DROP TABLE <表名> [RESTRICT|CASCADE];
    - RESTRICT表示该表的删除是有条件的
      - 欲删除的基本表不能被其他表的约束所引用（如CHECK，FOREIGN KEY等），不能有视图，不能有存储过程或函数等。如果存在这些依赖该表的对象，则此表不能被删除
    - CASCADE表示该表的删除没有限制条件
      - 在删除基本表时，相关的依赖对象一起被删除

插入数据

- INSERT…VALUES语句
  - INSERT INTO <表名> [(<列名1> [, … <列名n>])] VALUES (值1[,… , 值n]);
- INSERT…SET语句
  - INSERT INTO <表名> SET <列名1> = <值1>, …, <列名n> = <值n>
- 从文本文件快速导入数据
  - LOAD DATA INFILE <数据文件> INTO TABLE <表名> [FIELDS TERMINATED BY <列分隔符>][ENCLOSED BY <数据封闭标记>] [IGNORE <需要忽略的行数或者列数> {LINES|ROWS}]
    - 在某个数据中有分隔符的情况下，那么就可以数据封闭标记来将数据进行封闭，比如加双引号作为字符串
  - LOAD DATA还有很多种用法
    - LOAD DATA INFILE ... [REPLACE|IGNORE] INTO TABLE： replace into 表示如果导入过程中有唯一性约束，直接覆盖；ignore into 则跳过。
    - LOAD DATA LOCAL INFILE： 在非服务端执行load data需要使用local。比如在 ipA 处登录 ipB 上的mysqld，就需要用到 local 。需要先设置set global local_infile=1。
    - 可以指定字段： LOAD DATA INFILE ... INTO TABLE xxx (col1,col2,...)
    - 可以设定值： LOAD DATA LOCAL INFILE '$tmpfile' REPLACE INTO TABLE db.tbname (a,b,c,d,e,f) set g=11,h='xxx';

查询

- SELECT <字段1>, …, <字段n> FROM <表1>,…, <表n> [WHERE <筛选条件表达式>] [GROUP BY <分组字段>] [HAVING <组的筛选条件>] [UNION SELECT … FROM…] [ORDER BY <排序字段>] [LIMIT <返回的记录数>]
- 通常的执行顺序

顺序	语句	作用
5	SELECT	选择需要输出的列
1	FROM	将表从硬盘读取到内存
2	WHERE	选取元组
3	GROUP	分组
4	HAVING	选择分组
6	UNION	查询结果的集合运算
7	ORDER BY	排序输出
8	LIMIT	限制输出的数量

- 选择列
  - SELECT <字段1>,<字段2>,…,<字段n>
    - 字段的顺序和表中字段的顺序可以不一致，最后结果按照SELECT中的顺序展示
    - <字段>后面加上AS <别名>则可以在结果中显示别名
    - 如要选取所有字段则用*
    - 还可以对字段进行简单的计算
- 去重
  - SELECT DISTINCT <字段1>,…,<字段n>
    - 根据<字段1>,…,<字段n>对结果进行筛选，展示不重复的结果
    - 结果中仅包含<字段1>,…,<字段n>
    - DISTINCT必须紧挨着SELECT，中间不能有字段

- 选择行
  - WHERE语句设置筛选条件
    - 比较：<、<=、>、>=、=、<>
    - 确定范围：BETWEEN A AND B、NOT BETWEEN A AND B
    - 确定集合：IN、NOT IN
    - 字符匹配：LIKE，NOT LIKE
    - 空值：IS NULL、IS NOT NULL
    - 多重条件：AND、OR、NOT
- LIKE字符串匹配
  - LIKE ‘匹配字符串’ [ESCAPE ‘换码字符’]
    - <匹配串>：可以是一个完整的字符串，也可以含有通配符 % 或 _ 。%表示任意长度（≥0的任意字符）；_表示单个的任意字符。
    - 当匹配串为固定字符串时，可以用 = 运算符取代 LIKE
    - ESCAPE‘换码字符’: 匹配串中‘换码字符’(转义符)之后的字(%,_)，被定义为普通字符(不作通配符用)

- 选择行
  - GROUP BY分组
    - 将查询结果集按某一列或多列的值分组，值相等的为一组，一个分组以一个元组的形式展现
    - 只有出现在GROUP BY子句中的属性，才可出现在Select子句中
    - 那些没有出现在GROUP BY中的属性，需要通过组函数计算才可以出现
    - 组函数的使用格式：
      - COUNT([DISTINCT|ALL] *|列名)
      - SUM([DISTINCT|ALL] 列名)
      - AVG([DISTINCT|ALL] 列名)
      - MAX([DISTINCT|ALL] 列名)
      - MIN([DISTINCT|ALL] 列名)
    - 组函数可用于SELECT子句中的目标列表中，或在HAVING子句的分组表达式中用作条件。
    - 对分出的每一组用HAVING进行筛选，筛选条件要用到组函数。
- HAVING 与 WHERE的区别
  - WHERE 决定哪些元组被选择参加运算，作用于关系中的元组
  - HAVING 决定哪些分组符合要求，作用于分组

选择行

ORDER BY排序

- 用ORDER BY子句对查询结果按照一个或多个列的值进行升/降排列输出
- 升序为ASC，降序为DESC，默认为升序
- 空值将作为最小值排序

LIMIT限制结果显示数量

- LIMIT [<位置偏移量>,] <行数>
- “位置偏移量”指 MySQL 从哪一行开始显示，是一个可选参数，如果不指定“位置偏移量”，将会从表中的第一条记录开始（第一条记录的位置偏移量是 0，第二条记录的位置偏移量是 1，以此类推）
- “行数”指示返回的记录条数
  - 如果要返回从偏移量到结束的所有行，行数参数可设置为-1

多表查询

表和表之间是连接的关系

- 内连接（INNER JOIN，典型的连接运算，使用像 = 或 <> 之类的比较运算符）
  - 包括相等连接和自然连接。
  - 内连接使用比较运算符根据每个表共有的列的值匹配两个表中的行。例如，检索 students和courses表中学生标识号相同的所有行
- 外连接（可以是左向外连接、右向外连接或完整外部连接）
  - LEFT JOIN或LEFT OUTER JOIN：左向外连接的结果集包括子句中指定的左表的所有行。如果左表的某行在右表中没有匹配行，则结果集行中右表对应列为空值。
  - RIGHT JOIN 或 RIGHT OUTER JOIN：右向外连接是左向外连接的反向连接
  - FULL JOIN 或 FULL OUTER JOIN：完整外部连接返回左表和右表中的所有行；当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值（MySQL不支持）
- 交叉连接 CROSS JOIN
  - 交叉连接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉连接也称作笛卡尔积
  - CROSS JOIN可写为JOIN，甚至可省略为逗号

单表也可以自己与自己连接

- SELECT * FROM `stu` A, `stu` B;
- 这里`stu` A表示为表`stu`起别名为A，在这之后的子句中都可以用A来代表`stu`

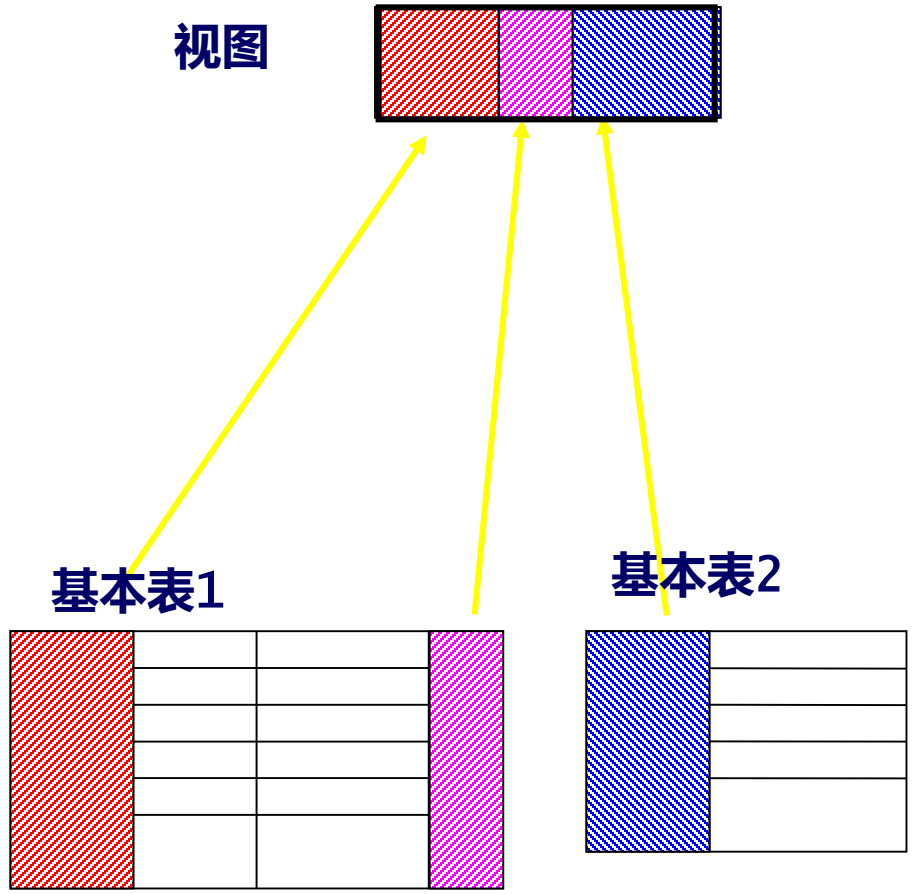
- 嵌套查询
  - 在SELECT … FROM … WHERE语句结构的WHERE子句中可嵌入一个SELECT语句块
    - 其上层查询称为外层查询或父查询
    - 其下层查询称为内层查询或子查询
      - SQL语言允许使用多重嵌套查询
      - 在子查询中不允许使用ORDER BY子句
      - 嵌套查询的实现一般是从里到外，即先进行子查询，再把其结果用于父查询作为条件
  - 返回单个值的子查询
  - 返回一组值的子查询
  - 带EXISTS的子查询
    - 不相关子查询：子查询的查询条件不依赖于父查询的称为不相关子查询。
    - 相关子查询：子查询的查询条件依赖于外层父查询的某个属性值的称为相关子查询(Correlated Subquery), 带EXISTS 的子查询就是相关子查询
    - EXISTS表示存在量词，带有EXISTS的子查询不返回任何记录的数据，只返回逻辑值‘True’或‘False’

- 更新数据
  - UPDATE
    - UPDATE <表名> [别名] SET <列名1> = <表达式1>,…,<列名n> = <表达式n> [WHERE <条件>]
    - UPDATE <表名> [别名] SET (<列名1>,…,<列名n>) = <子查询> [WHERE <条件>]
  - DBMS在执行修改语句时会检查所修改的元组是否破坏表上已定义的完整性规则
- 删除数据
  - DELETE
    - DELETE FROM <表名> [WHERE <条件>]

视图

视图的概念

- 视图是一个虚表
- 数据库中只存放视图的定义
- 视图对应的数据仍存放在原来的表中
- 随着表中数据的变化，视图的数据随之改变。
- 对视图的查询与基本表一样
- 对视图的更新将受到一定的限制



创建视图

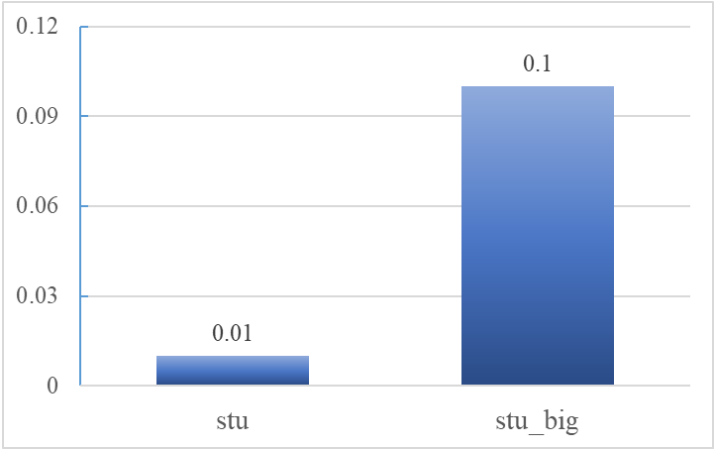
- CREATE VIEW <视图名> AS (<子查询>)

更新视图

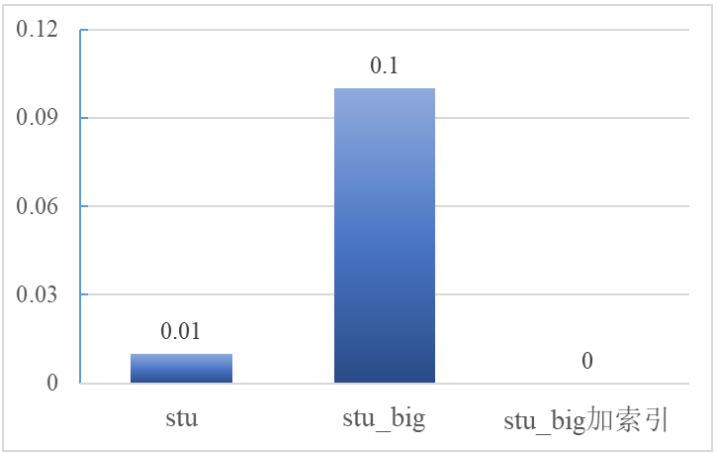
- 即通过视图插入、删除和修改数据，实质上转换为对基本表的更新

- 视图的优点
  - 提供数据的逻辑独立性
  - 提供数据的安全保护功能
  - 简化用户的操作
    - （对系统构成的视图，用户不必关心各表间的联系）
  - 同一数据多种用法

- 索引
  - 看一个简单的查询语句：
    - SELECT COUNT(*) FROM `stu` WHERE birth_year=1990;
    - 在两张表中分别运行，stu表含19条记录，stu_big表含190000条记录
  - 索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针
    - 更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度



- 创建索引
  - CREATE INDEX <索引名> ON <表名> (<列名1>[ASC | DESC]);
  - MySQL只对以下操作符才使用索引：<,<=,=,>,>=,between,in,以及某些时候的like(不以通配符%或_开头的情形)
  - 索引的缺点
    - 虽然索引大大提高了查询速度，同时却会降低更新表的速度
    - 因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件



- 使用pymysql包（适用于Python3）
  - 连接数据库
    - db = pymysql.connect(host=数据库地址,port=端口,user=用户名,passwd=密码,db=数据库,charset=字符集)
    - 本地数据库地址可以填localhost
    - 最好用try…except异常处理语句块包围
    - 结束后db.close()关闭连接，减少资源占用
  - 创建一个游标对象
    - cursor = db.cursor()，结束后用cursor.close()关闭
  - 执行语句
    - cursor.execute(SQL语句)
    - 更新、删除、插入操作还需要：db.commit()
  - 获取结果
    - fetchone(): 该方法获取下一个查询结果集，结果集是一个list对象
    - fetchall(): 接收全部的返回结果行
    - rowcount: 这是一个只读属性，并返回执行execute()方法后影响的行数
  - 回滚
    - db.rollback()

大数据时代的矛盾

大数据3V

- Volume容量：数据的容量越来越大
- Velocity速度：数据的增长速度和所需的数据处理速度越来越快
- Variety多样：数据的格式越来越多样化

对数据库的存储能力要求越来越高：大容量存储、高速读写

传统的关系型数据库特点：

- 计算耗时（查询效率低）
- 关系表以文件形式存储（一个文件一张表）
  - NTFS文件系统单个文件最大2TB
  - 磁盘单碟容量20TB（机械）或者100TB（固态硬盘）

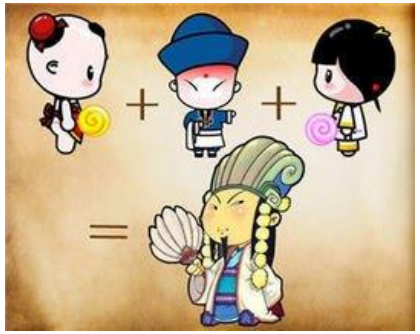
矛盾

- 动辄上PB甚至ZB的数据量

传统的关系型数据库越来越无法满足大数据时代的需求

解决方案

分布式

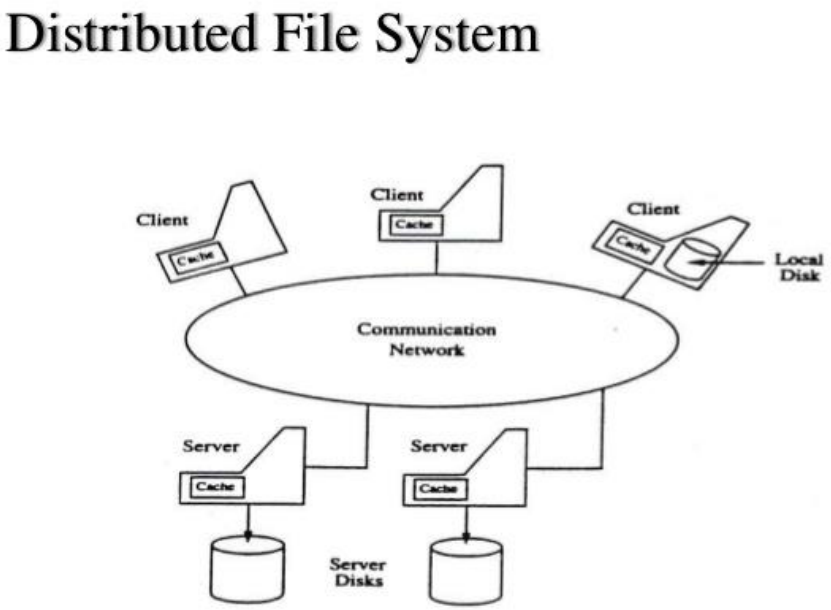


分布式文件系统

分布式文件系统（Distributed File System）

是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点(可简单的理解为一台计算机)相连

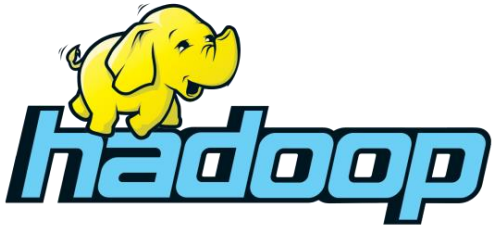
- 分布式文件系统有很多，比如DFS系统
  - DFS为文件系统提供了单个访问点和一个逻辑树结构
  - 通过DFS，可以将同一网络中的不同计算机上的共享文件夹组织起来，形成一个单独的、逻辑的、层次式的共享文件系统



分布式文件系统

GFS （Google 文件系统）

- GFS 是 Google公司为了存储海量搜索数据而设计的专用文件系统
  - GFS是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用
  - 它运行于廉价的普通硬件上，并提供容错功能
  - 它可以给大量的用户提供总体性能较高的服务
- 但是GFS是Google内部的文件系统，并不公开



HDFS（Hadoop DFS）

- HDFS是实现了GFS MapReduce框架模型的一个开源的分布式文件系统
  - Hadoop可以使用户在不了解分布式底层的情况下充分利用集群的威力，开发分布式程序，实现高速运算和存储
  - 理论上对集群中的节点（每一台计算机）硬件能力没有特别要求
- HDFS具有高容错高可靠性、高可扩展性、高获得性、高吞吐率等优点
  - 2008年4月依托拥有910个节点的集群，209秒内完成了对1TB数据的排序，击败了前年的冠军297秒
  - 2009年4月，在1400个节点的集群上对500G数据排序仅用了59秒

Hadoop/GFS设计目标

硬件错误是常态而非异常

- HDFS被设计为运行在普通硬件上，所以硬件故障很正常
- 一个集群可能存在着成百上千的节点，每个节点上都有文件系统的部分数据，每个组件都有可能出现故障
- 因此，错误检测并快速自动恢复是HDFS的最核心设计

流式数据访问

- HDFS上的应用主要是以流式数据读取为主，做批量处理而非用户交互处理
- 数据是一点点传输过来的，一边传输一边处理，而非全部传输后再统一处理，这样不需要占用大量的内存空间

大规模数据集

- 典型文件大小为GB甚至TB，因此支持大文件存储

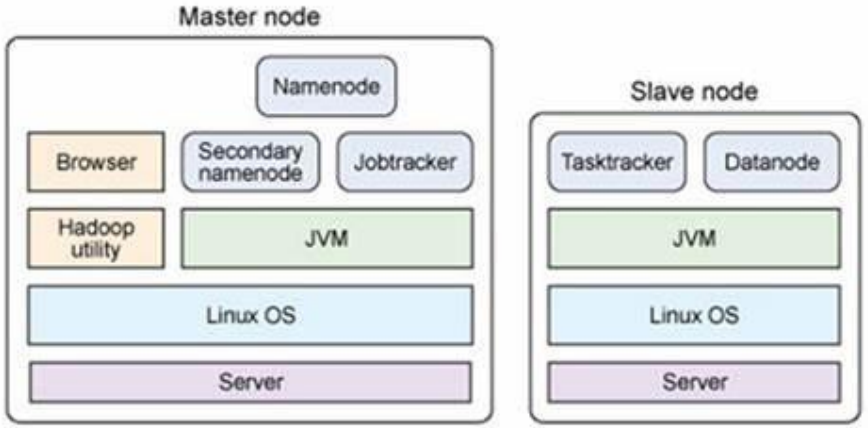
简单一致性模型

- 对文件一次性写入、多次读取，文件一经写入后就不需要再更改了，这样的假定简化了数据一致性问题

移动计算比移动数据更划算

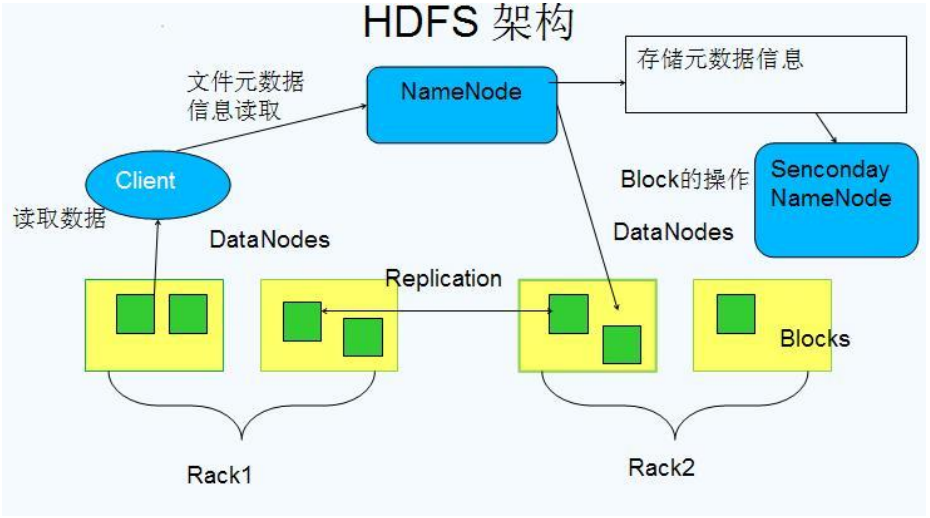
- 对于大文件而言，再数据旁执行操作会比将数据移动到计算应用附近更高效
- HDFS提供了接口，以便让程序将自己移动到数据存储的地方执行

- 主/从（Master/Slave）体系结构
  - 一个主节点、多个从节点
  - MapReduce能够将一个应用程序分割成许多小的工作单元以便在各个节点上执行
    - 其中这个应用程序叫“作业（job）”，分割出来的工作单元叫“任务（task）”
    - Hadoop集群中运行了一系列的后台程序
      - NameNode（仅有一个）：HDFS的守护程序，负责记录文件是如何背分割成数据块的，以及这些数据块分别存储在哪些数据节点上（元数据）；一旦NameNode宕机，整个系统将无法运行
      - DataNode：负责具体的读写操作
      - Secondary NameNode：用来监控NameNode的辅助程序，并定期保存元数据快照
      - JobTracker：决定那个文件被处理，并为不同的任务分配节点
      - TaskTracker：与负责存储数据的DataNode结合，管理节点中的任务；与JobTracker交互，一旦JobTracker无法获取TaskTracker的响应，则判定崩溃



HDFS体系结构

- HDFS的文件通常按照64MB被切分成不同的数据块（Block），每个数据块尽可能地分散存储于不同的DataNode中
- NameNode管理文件系统的元数据，DataNode存储实际数据
- 客户端通过同NameNode和DataNode的交互访问文件系统；联系NameNode以获取文件的元数据，而真正的文件I/O操作则是直接和DataNode交互
- NameNode使用事务日志（EditLog）来记录元数据的变化，使用映像文件（FsImage）存储文件系统的命名空间，包含文件的映射、文件的属性信息等；NameNode启动的时候合并映像文件和事务日志
- Secondary NameNode辅助NameNode处理映像文件和事务日志；周期性从NameNode复制映像文件和事务日志到临时目录，合并后上传到NameNode，NameNode更新映像文件并清理事务日志，使得日志大小始终控制在可配置的限度下



保障HDFS可靠性的措施

冗余备份

- 文件的所有数据块都会有副本（副本数量即复制因子，可配置）
- HDFS的文件都是一次性写入，并且严格限制为任何时候都只能有一个用户进行写操作
- DataNode使用本地文件系统来存储HDFS数据，但是对HDFS的文件一无所知
- DataNode启动时遍历产生一份数据块和本地文件对应关系列表（块报告，Blockreport），并发给NameNode

副本存放

- HDFS一般运行在多个机架上，采用机架感知的策略来改进数据的可靠性、可用性和网络带宽的利用率
- 通过机架感知，NameNode可以确定每个DataNode所属机架的ID
- 一般情况下，复制因子为3时，HDFS部署策略为一个副本在本地节点，一个副本在同一机架的另一个节点，最后一个放在不同机架上的节点；由于机架错误远比节点错误少，因此可以保证数据的可靠性和可用性，同时保证性能

心跳检测

- NameNode周期性地从集群中每个DataNode接受心跳包和块报告，如果没收到则判断为宕机，不再发送新的I/O请求

安全模式

- 系统启动时NameNode会进入安全模式，此时不进行数据块的写操作。NameNode接受Blockreports来确认安全，达到足够比例就退出安全模式

保障HDFS可靠性的措施

数据完整性检测

- 文件创建时会计算每个数据块的校验和，并将校验和作为一个单独的隐藏文件保存在命名空间下
- 客户端获取文件后会比对校验和，如果不同则认为数据损坏，从其它DataNode获取副本

空间回收

- 文件被用户或者应用程序删除时不会立刻从HDFS移走，而是进入到/trash目录，可以随时恢复
- 一定时间后才会被删除

元数据磁盘失效

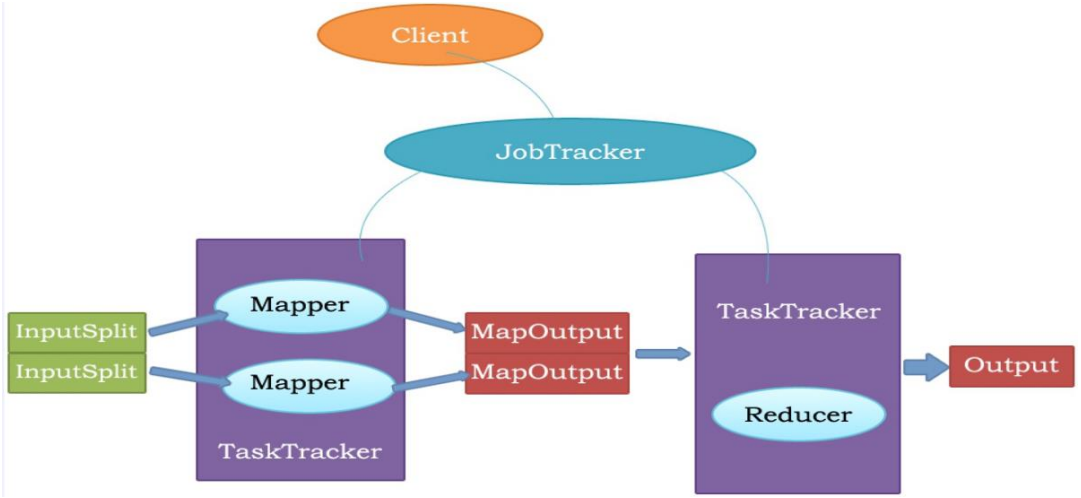
- NameNode可以配置为支持维护映像文件和事务日志的多个副本，任何修改都将同步到副本
- NameNode重启时总会选择最新的一致的映像文件和事务日志

快照

- HDFS目前还不支持快照

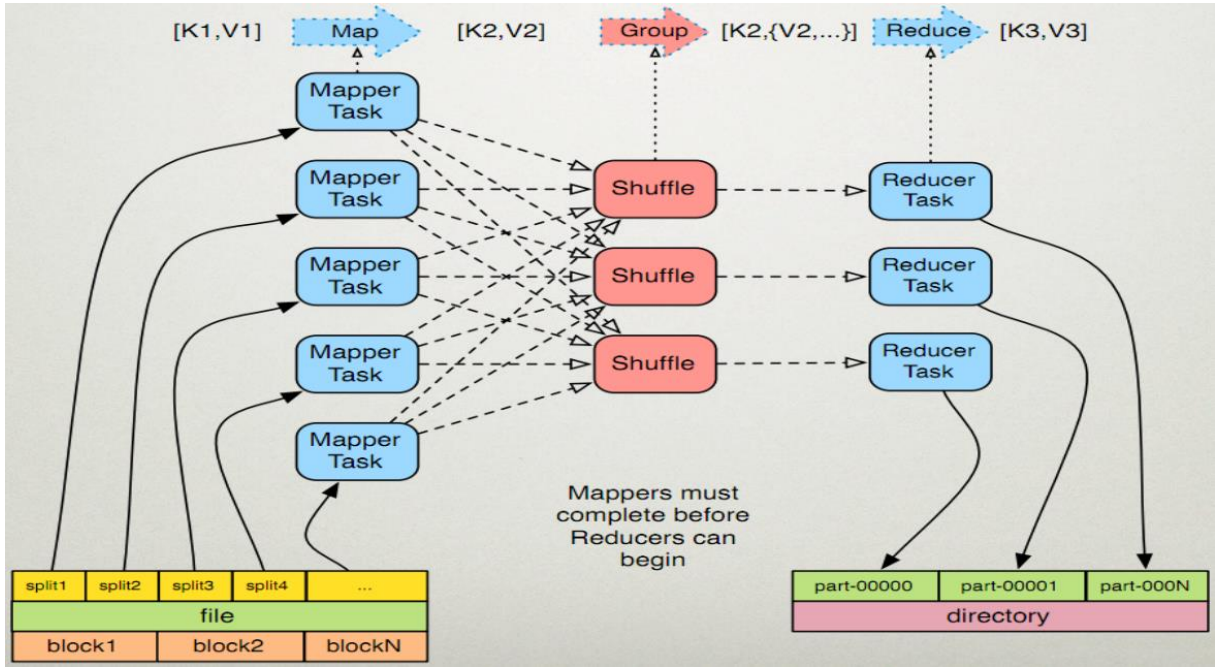
MapReduce——HDFS的核心

- MapReduce采用“分而治之”的思想，把对大规模数据集的操作分发给一个主节点管理下的各个分节点共同完成，然后通过整合各个分节点的中间结果得到最终结果
- MapReduce有两个阶段组成：Map（映射）和Reduce（归约），用户只需实现map()和reduce()两个函数，即可实现分布式计算
  - Map负责把任务分解成多个任务
  - Reduce负责把分解后的多任务处理结果汇总
  - 其它的比如分布式存储、调度、负载均衡等均由框架负责处理
  - 需要注意的是，使用MapReduce来处理的数据或者任务必须
    - 可以分解成许多小的数据集
    - 每个小数据集都可以完全并行地处理



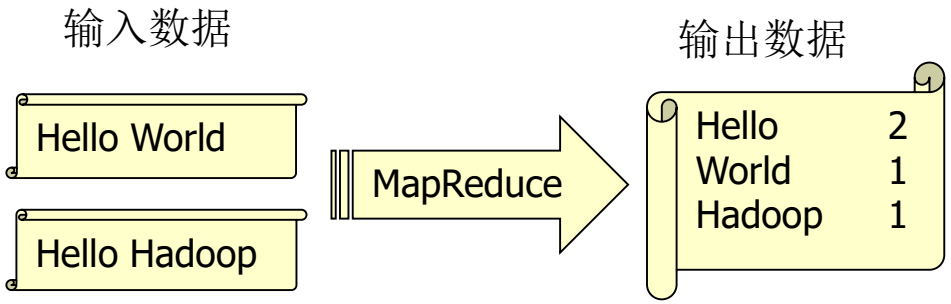
MapReduce基本流程

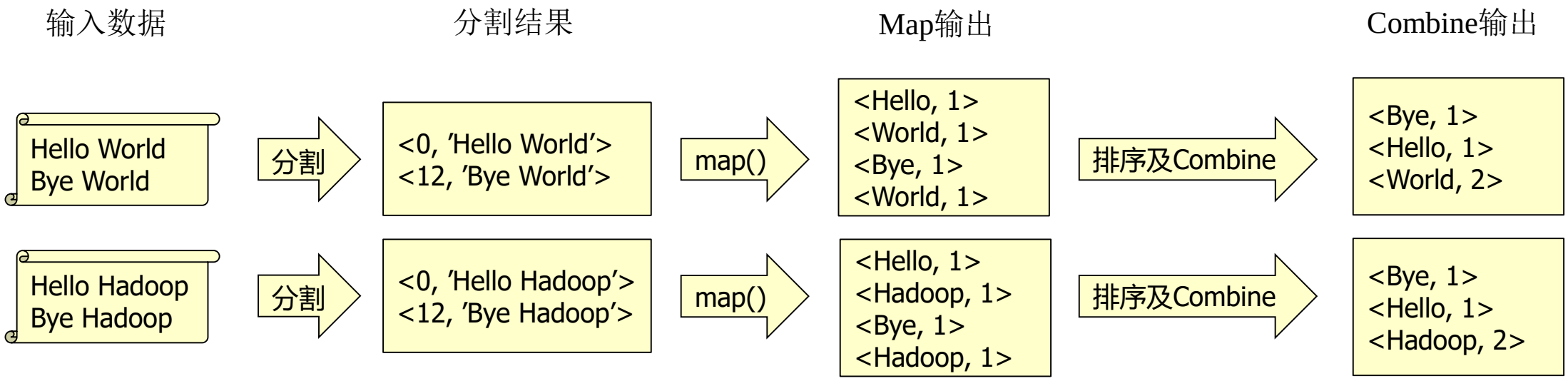
- 1. 输入数据分割为固定大小的片（split）
- 2. 每个split进一步分解成一批键值对[<K1,V1>]
- 3. 为每个split创建一个Map任务用于执行用户自定义的map函数，以[<K1, V1>]为输入，得到计算的中间结果[<K2, V2>]
- 4. （Shuffle过程）根据K2的范围进行分组对应到不同的Reduce任务（reducer）
- 5. Reducer把接收到的数据整合在一起排序，调用用户自定义reduce函数，对[<K2, V2>]进行处理，得到[<K3, V3>]并输出
- 注意：mapper数量由框架决定，但是reducer数量由用户自定义决定



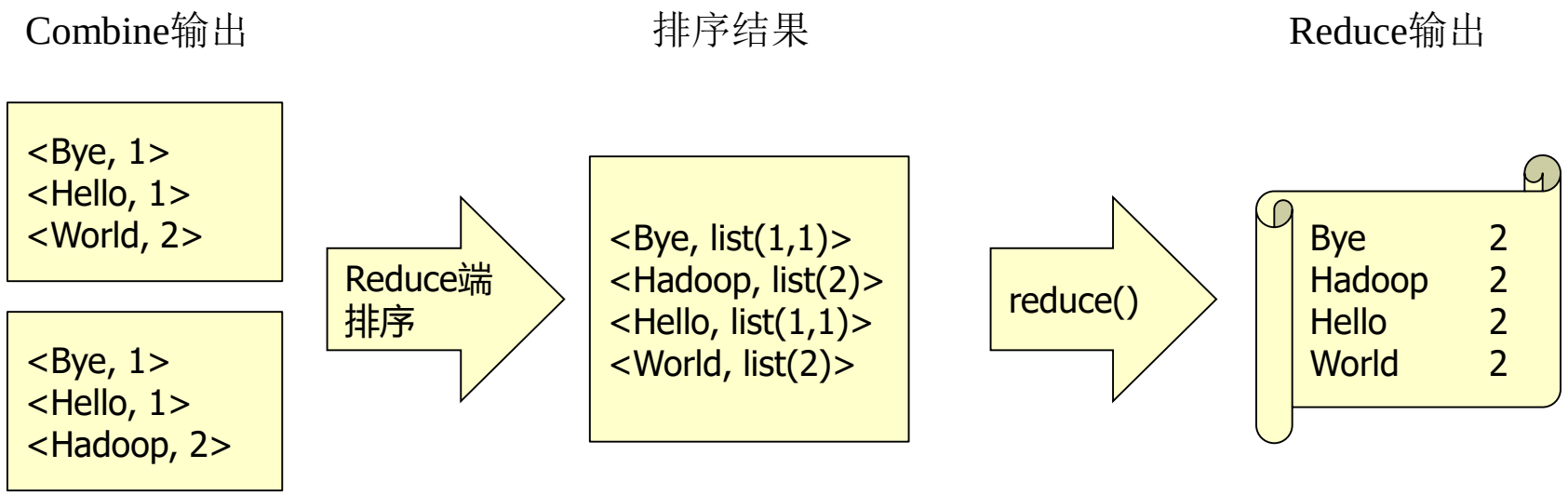
MapReduce经典算例

- WordCount
  - 主要功能是统计一系列文本文件中每个单词的出现次数
  - 是最简单也是最能体现MapReduce思想的程序之一





- WordCount
  - 主要功能是统计一系列文本文件中每个单词的出现次数
  - 是最简单也是最能体现MapReduce思想的程序之一



一个现实的问题：数据有时候是高维且稀疏的

稀疏数据的存储方案

- 1.把所有列的数据放在一个文件中（也就是传统的按行存储）
  - 那么当我们想要访问少数几个列的数据时，需要遍历每一行，读取整个表的数据，这样子是很低效的
- 2.把每个列的数据单独分开存在一个文件中（按列存储）
  - 那么当我们想要访问少数几个列的数据时，只需要读取对应的文件，不用读取整个表的数据，读取效率很高
  - 然而，由于稀疏表通常会有很多列，这会导致文件数量特别多，这本身会影响文件系统的效率

用户名	密码	邮箱	手机号	家庭住址	性别	真实姓名	年龄	身份证号	婚姻状况	爱好	职业	学历
user1	12345	user1@163.com		厦门					未婚		工人	
user2	54321	user2@163.com			男			34567898765				
user3	2341	user3@163.com	13999997781				19			篮球		
user4	11111	user4@163.com			女							
…	…	…	…	…	…	…	…	…	…	…	…	…
user_n	201999	user_n@163.com				张三						高中

HBase提供了第三种解决方案：两者相结合

HBase = Hadoop Database

逻辑模型

- HBase每个列属于一个特定的列族（Column Family）
- 表中由行和列确定的存储单元成为一个元素（cell）
- 每个元素保存了多个版本，由时间戳来标识
- 行键是行的唯一标识，并作为检索的主键
- HBase中三种访问方式：
  - 通过单个行键访问
  - 给定行键的范围访问
  - 全表扫描
- 行键可以是任意字符，并按照字典顺序排列
- 列的定义为：<列族>:<限定符>
- 时间戳对应每次操作的时间，元素由行键、列、时间戳唯一确定

HBase逻辑视图

行键	时间戳	列族info	列族pwd
1001	20190909		pwd:密码=WoShiZWJ@wudang5
	20160919	info:姓=曾 info:名=阿牛	
	20160719	info:姓=张 info:名=无忌 info:帮派=武当	
	20160711		pwd:密码=321
	20160101		pwd:密码=123

HBase的物理模型

- HBase是按照列来存储的矩阵
- 物理模型实际上就是把概念模型中的一行进行分割，并按照列族存储
  - 空值是不被存储的
  - 查询时间戳为20191001、行键1001、info:姓这一列返回的是null
  - 如果不指定时间戳则返回最新值
  - 列族需要建表时候事先确定，列可以随时增加

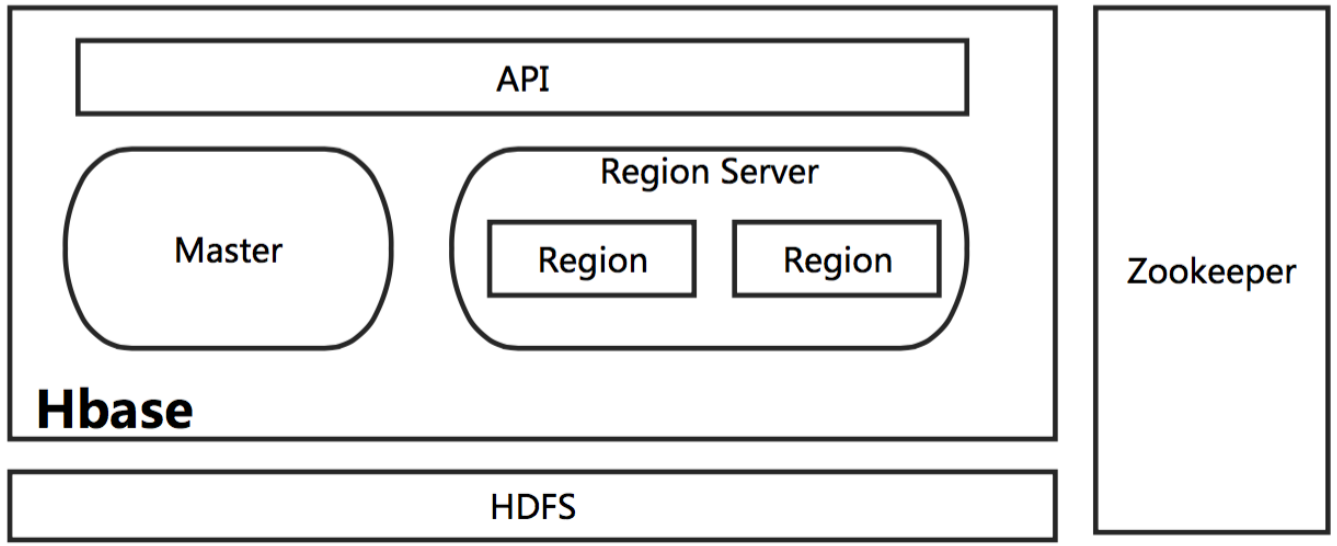
传统关系型数据库

ID	姓	名	密码	帮派	时间戳
1001	张	三	321	武当	20160719
1002	李	四	222		20160720

HBase物理模型

行键	时间戳	列族
1001	20160719	info{'姓': '张'，'名':'无忌', '帮派':'武当', }
	20160919	info{'姓': '曾'，'名':'阿牛'}
1001	20160101	pwd{'密码': '123'}
	20160711	pwd{'密码': '321'}
	20190909	pwd{'密码': 'WoShiZWJ@wudang5'}
1002	20160720	info{'姓': '李'，'名':'四'}
1002	20160720	pwd{'密码': '222'}

- HBase架构
  - HBase在行的方向上将表分成了多个Region，每个Region包含一定范围内（根据行键划分）的数据
  - 每个表最初只有一个Region，随着记录不断增加Region不断增加
  - 通常一台服务器只有一个Region
- HBase不支持SQL查询
  - 也就是目前正在逐渐流行的NoSQL
  - NoSQL = Not Only SQL



HBase与传统关系型数据库（RDBMS）的区别

	HBase	RDBMS
硬件架构	类似于 Hadoop 的分布式集群，硬件成本低廉	传统的多核系统，硬件成本昂贵
容错性	由软件架构实现，由于由多个节点组成，所以不担心一点或几点宕机	一般需要额外硬件设备实现 HA 机制
数据库大小	PB	GB、TB
数据排布方式	稀疏的、分布的、多维的 Map	以行和列组织
数据类型	Bytes	丰富的数据类型
事物支持	ACID 只支持单个 Row 级别	全面的 ACID 支持，对 Row 和表
查询语言	只支持 Java API （除非与其他框架一起使用，如 Phoenix、Hive）	SQL
索引	只支持 Row-key，除非与其他技术一起应用，如 Phoenix、Hive	支持
吞吐量	百万查询/每秒	数千查询/每秒