金融科技序列考试（初级）

1. 信息安全专项（3%）

散列算法

也称单向散列函数、哈希算法、HASH算法、或者消息摘要算法。

好的散列函数的特点：

1. 计算单向性。
2. 弱碰撞自由。给定M，寻找$M^{'}$使得$H(M)=H(M^{'})$在计算上不可行。
3. 强碰撞自由。寻找$M,M^{'}$使得$H(M)=H(M^{'})$在计算上不可行。

MD5

SHA-1

数字签名

即Digital Signature。

数字签名过程：

1. 发送者利用发送者的私钥对摘要信息加密，发送给接受者。
2. 接受者用公钥解密，和原文的摘要信息进行比对。

数字证书

数字信封

即对密钥进行非对称加密，利用密钥对称加密实际内容。

SSL协议

MAC

即Message Authentication Code，消息鉴别码。

MAC过程：

1. 参与通讯的双方间通过秘密信道共享一个密钥和特定算法。
2. 发送方生成MAC并随报文发送。
3. 接收方生成MAC并比对。

2. 程序语言专项（7%）（java）

略。

3. 数据库专项（8%）

数据模型的基本概念

三个数据领域

现实世界

如报表、图表等。

信息世界

如实体、实体集、属性、码。

机器世界

如字段、记录、文件、记录码。

信息世界与机器世界术语的对应关系

属性与字段

属性，即attribute。
字段，即field。

在数据库里面，表的“列”称为“字段”。

实体与记录

实体，即entity。客观存在并可以相互区分的事物。
记录，即record。在关系数据库中，也被称为元组（tuple）。
在数据库里面，表的“行”称为“记录”。

码

即key，键。唯一能区分实体的属性或者属性集。

实体型

具有相同属性的实体具有由相同的特征和性质，用实体名及其属性名集合来抽象和刻画同类实体，称为实体型。

实体集

同型实体的集合称为实体集。

概念数据模型

又称信息模型。即现实世界到机器世界的抽象。

E-R模型

即Entity-relationship model，实体联系模型。
（目前没看出来有啥用，有啥用以后再说吧）

基本数据模型

按计算机系统的观点对数据建模。

层次模型

网状模型

关系模型

面向对象模型

数据模型三要素

1. 数据结构
2. 数据操作
3. 数据的约束条件

DBMS

即Database Management System，数据库管理系统。

DDL

即Data Definition Language，数据定义语言。用于对数据库的结构进行描述。

外模式

即External Schema。

用户与数据库系统的接口，是概念模型的逻辑子集。外模式面向具体的应用程序，定义在逻辑模式之上，但独立于存储模式和存储设备。

概念模式

数据库的逻辑表示，包括每个数据的逻辑定义以及数据间的逻辑联系。

内模式

内模式又称存储模式，对应于物理级，它是数据库中全体数据的内部表示或底层描述，是数据库最低一级的逻辑描述。

完整性定义（）**

安全保密定义（）**

数据字典（）**

DML

即Data Manipulation Language，数据操纵语言。实现检索、插入等功能。

数据库运行管理

如并发控制、安全检查、存取控制、完整性检查、执行和运行日志的组织管理和事务管理、自动回复等。

数据组织、存储和管理

如分类组织、存储、管理各种数据等。

数据库的建立和维护

如数据库的建立、转换、转储、恢复等。

其他功能

如通信、数据转换等。

DBMS的特征

数据结构化且统一管理

有较高的数据独立性

数据控制功能

1. 安全性。即权限和防止数据的泄露、更改和破坏。
2. 完整性。数据库的完整性是指数据库的正确性与相容性。保证数据库中的数据是正确的，避免非法更新。
3. 并发控制。
4. 故障恢复。

DBMS的主要分类

RDBS

即Relation Database Systems，关系型数据库系统。

在关系模型中，实体以及实体之间的联系用关系来表示。

关系型数据库通常包含以下组件：Client Server SQL。常见的RDBS有：Oracle、SQL等。

OODBS

即Object-Oriented Database System，面向对象数据库系统。

ORDBS

即Object-Oriented Relation Database System，对象关系数据库系统。

在RDBS中增加了元组、数组、集合等数据类型以及操作的能力。

SQL

SQL的特点

1. 综合统一。SQL集数据定义、数据操纵和数据控制功能于一体，风格统一。
2. 高度非过程化。即只要指出“做什么”，无须指出“怎么做”。
3. 面向集合的操作方式。SQL语言面向集合，其操作对象、查找结果可以是元组的集合。
4. 自含式语言和嵌入式语言。

超键

即Super Key。在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。

候选键

即Candidate key。最小超键。

主键

即Primary Key。也被称为关键字。用户选作元组标识的一个候选键。

作为主键最好是完全业务无关的字段，我们一般把这个字段命名为id。常见的可作为id字段的类型有：

1. 自增整数类型：数据库会在插入数据时自动为每一条记录分配一个自增整数，这样我们就完全不用担心主键重复，也不用自己预先生成主键；
2. 全局唯一GUID类型：使用一种全局唯一的字符串作为主键，类似8f55d96b-8acc-4636-8cb8-76bf8abc2f57。GUID算法通过网卡MAC地址、时间戳和随机数保证任意计算机在任意时间生成的字符串都是不同的，大部分编程语言都内置了GUID算法，可以自己预算出主键。

外键

即Foreign Key。用于将两个表连接在一起，让两个表的数据保持同步。

一个表的外键用来指向另一个表的主键。包含外键的表称为从表，被指向的表称为主表。

SQL的核心操作

数据查询

SELECT

数据定义

CREATE、DROP、ALTER

数据操纵

INSERT、UPDATE、DELETE

数据控制

GRANT、REVOKE

事务处理

指针控制

SQL DDL的常用命令

创建表（ ）**

修改表（ ）**

删除表（ ）**

定义索引（）**

创建索引（）**

删除索引（）**

SQL查询

数据查询（）**

简单查询（）**

连接查询（）**

子查询（）**

聚集函数（）**

分组查询（）**

使用别名（）**

字符串操作（）**

视图查询（）**

数据更新和访问控制

插入语句（）**

删除语句（）**

修改语句（）**

授权语句（）**

收回权限语句（）**

数据库事务管理

事务开始（）**

Begin TRANSATION

事务提交（）**

COMMIT

事务回滚（）**

ROLLBACK

数据库并发控制

并发操作带来的数据不一致问题：

事务的并发操作破坏了事务的隔离性。

1. 丢失更新
2. 不可重复读
3. 读脏数据

并发控制技术

排他锁

又称X锁或者写锁。

事务T对数据对象A加上X锁，则只允许T读取和修改A，直到T释放A。

共享锁

又称S锁或者读锁。

事务T对数据对象A加上S锁，则只允许T读取A。在此期间，其他事务只能再对A加上S锁，直到释放S锁。

三级封锁协议

一级封锁协议

修改数据要加X锁。这解决了丢失更新问题。

二级封锁协议

在一级封锁协议的基础上，增加了读数据要加S锁，读完释放。这解决了读脏数据问题。

三级封锁协议

在一级封锁协议的基础上，增加了读数据要加S锁，事务结束释放。这解决了可重复度问题。

活锁与死锁

活锁

可能永远等待。

死锁

分别请求封锁对方已经封锁的数据。导致无法继续运行。

最大限度降低死锁的方法（）**

可串行准则

结果与串行相同=并发事务正确。

两端封锁协议

分两个阶段，第一个阶段只封锁，第二个阶段只解锁。

数据库的备份与恢复（中级）

数据仓库简介（中级）

NoSQL非关系型数据库简介（中级）

数据库应用系统的生命周期（中级）

数据库设计的方法（中级）

需求分析（中级）

概念结构设计（中级）

逻辑结构设计（中级）

数据库物理设计（中级）

数据库实施（中级）

数据库维护（中级）

4. 分布式专项（6%）

分布式服务

背景

1. 推动我行架构向分布式（系统解耦）、服务化转型（提升业务快速组合创新）。
2. 满足业务高容量，高并发，高增长。

适用场景

包括但不限于：

1. 并发量暴增。
2. 随着业务发展，用户并发量会大规模增长的应用，如客户信息查询。
3. 支撑上述应用的基础应用。
4. 超过32K的大报文应用，或者大数据类等互联网对客应用。建议使用分布式—大数据框架（DDS）。

服务定义

1. 从技术视角来看，服务是可发现的、可调用的，可复用的，经过封装的软件资源，是应用功能之间进行访问的能力。
2. 从业务视角来看，服务是经过标准化定义的，具有独立完整业务含义的业务活动。

分布式服务

服务集群

服务网关

注解

面向服务架构（SOA）：

即SOA（Service-Oriented Architecture）。SOA定义了一种可通过服务接口复用软件组件的方法。 此类接口会使用通用的通信标准，这些标准能够快速合并到新应用程序中，而不必每次都执行深度集成。

SOA具有如下特点：

1. 每项服务都包含执行完整的独立业务功能所需的代码和数据集成。
2. 这些服务接口的耦合比较松散，也无需知道底层逻辑，可以视为黑箱。
3. 使用 SOAP（简单对象访问协议）/HTTP 或 JSON/HTTP 等标准网络协议来公开服务，以便发送有关读取或更改数据的请求。

SOA和微服务架构的差别：

微服务是对SOA的升华，其主要区别如下：

1. 微服务去中心化，去掉ESB企业总线。
2. Docker容器的出现。
3. 微服务完全分离。

DDS：

Data Distribution Service 数据分发服务。

Restful：

Reastful是一种网络应用程序的设计风格和开发方式，基于HTTP，可以使用XML格式定义或JSON格式定义。

HTTP协议可以理解为Restful的一种表现。但HTTP协议不完全遵循Restful的原则，如大多数Web服务器都是用cookie和会话存储状态。

Restful有四个特点：

1. 资源，URI
2. 状态转化，PUT，GET，POST，DELETE
3. 表现层，Representation。资源具体表现出来的形式，如HTML、JSON、XML等。
4. 无状态性。客户端与服务端之间的交互在请求之间是无状态的，从客户端到服务端的每个请求都必须包含理解请求所必需的信息。

分布式事务

术语和定义

事务准入适配

准入规避原则：

1. 若通过业务功能设计、利用本地数据库可以实现业务一致性，则禁止引入分布式。
   解读：关系数据库比分布式事务更加成熟。
2. 晚间批量、联机小批量禁止引入分布式。

SAGA模型适配

1. 隔离性强，无并发的业务，宜用SAGA模型。
2. 可能造成银行账务风险场景，禁止使用SAGA。

TCC模型适配

1. 可能造成银行账务风险的业务场景，应使用TCC模型。
2. 对同一业务数据修改有并发的业务，应使用TCC模型。

消息模型适配

事务设计基本规范

事务开发测试

1. 事务容量：参与者应不大于10个，每个事务的输入/输出参数大小在8KB以内。
2. 事务测试：关注服务相应时间、tps（Transaction per Second，即事物数/秒）、cpu、内存、磁盘IO。TCC模型还要关注并发度。

事务部署

注解

ACID

是指数据库管理系统在写入或更新资料的过程中，为保证事务是正确可靠的，所必须具备的四个特性。

原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。

隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。
持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

幂等性

接口的幂等性实际上就是：
接口可重复调用，在调用方多次调用的情况下，接口最终得到的结果是一致的。

幂等性的实现方法：（待学习）

1. 全局唯一ID。通过全局唯一ID判断该操作是否已经执行。
2. 去重表
3. 插入或更新
4. 多版本控制
5. 状态机控制

SAGA模型

SAGA的组成

1. 每个Saga由一系列Ti 组成
2. 每个Ti 都有对应的补偿动作Ci，补偿动作用于撤销Ti造成的结果

SAGA的执行顺序

1. T1, T2, T3, ..., Tn
2. T1, T2, ..., Tj, Cj,..., C2, C1，其中0 < j < n

SAGA的恢复策略

1. backward recovery，即第二种执行顺序
2. forward recovery， 即第一种执行顺序，用于必须成功的场景。

SAGA的使用条件

1. 只允许两个层次的嵌套
2. 在外层，全原子性不能得到满足。也就是说，sagas可能会看到其他sagas的部分结果
3. 每个子事务应该是独立的原子行为

对ACID的保证（对一致性和隔离性不保证）

1. 原子性（Atomicity）：正常情况下保证。
2. 一致性（Consistency），在某个时间点，会出现A库和B库的数据违反一致性要求的情况，但是最终是一致的。
3. 隔离性（Isolation），在某个时间点，A事务能够读到B事务部分提交的结果。
4. 持久性（Durability），和本地事务一样，只要commit则数据被持久。

TCC模型

即Try-Confirm-Cancel。

空回滚与防悬挂

TCC服务中，未收到Try请求的情况下，收到Cancel请求，这被称为空回滚。TCC模型应允许空回滚。
TCC服务中，如果在执行Cancel请求后才收到Try请求，应拒绝Try请求。这被称为防悬挂。

DSF服务

即Distributed Service Framework，分布式服务框架。

5. 程序设计基础与程序设计专项（1%+8%+1%）

程序设计基础知识

用户界面设计（和教材不一致）

1. 结构设计
2. 交互设计
3. 视觉设计

程序流程图

1. 顺序（加工）
2. 选择判断（逻辑）
3. 循环（控制）

内聚

一个模块内部各个元素间彼此结合的紧密程度的度量。

内聚可以划分为以下7个种类：

1. 偶然内聚
2. 逻辑内聚
3. 时间内聚
4. 过程内聚
5. 通信内聚
6. 顺序内聚
7. 功能内聚

耦合

模块之间依赖程度的度量。

耦合可分为4个种类：

1. 内容耦合
2. 控制耦合
3. 公共环境耦合
4. 数据耦合

高内聚

1. 功能模块化、单一化。
2. 复杂功能进行分解。

低耦合

1. 分层，层之间用设计模式实现分离。

大部分Web应用可分为：

A. 表现层：MVC模式

B. 业务层：工厂模式，IoC模式，AOP模式

C. 持久层：DAO模式，ORM模式

2. 使用框架代替模式。

程序设计复用技术

程序设计复用技术可分为：

1. OOD，面向对象设计。
2. POD，面向过程设计。
3. SOD，面向服务设计。
4. IOD，面向接口设计。
5. AOD，面向方面设计。

程序的可维护性（中级）

程序的可维护性定义

可分析性、可改变性、稳定性、易测性、可理解性。

提高稳定性的方法

1. 类型不匹配，强制转换
2. 给予类成员最小的可见性
3. 讲组件和服务尽可能单独实现
4. 变量声明限制在作用域等

提高可识别性的方法

有效命名

提高可分析性

提高可改变性

提高可维护性

提高可分离性

数据库设计

数据库范式

第一范式

数据库表中的字段都是单一属性、不可再分的。

第二范式

在第一范式的基础上，不存在非关键字段对任一候选关键字段的部分函数依赖。

第三范式

在第二范式的基础上，不存在非关键字段对任一候选关键字段的传递函数依赖。

范式的优点

结构清晰、避免数据冗余和操作异常。

范式的缺点

数据之间关系过多，连接操作频繁，不利于查询操作。

数据库反范式

常用技术

冗余列、派生列、重新组表、分割表。

优点

查询方便。

缺点

数据的完整性问题，表的修改速度问题。

数据库反范式实例：冗余表设计

垂直切分

将一个库中的数个表，分到若干个库中。

解决的问题：降低单节点数据库的负载。

未解决的问题：MySQL单表记录超过2000万，读写性能会下降的很快，垂直切分并不能起到缩表的效果。

水平切分

按照某个字段的某种规则，把数据切分到多张数据表。一张数据表化整为零，拆分成多张数据表，这样就可以起到缩表的效果了。

先做水平切分，后做垂直切分

冗余表设计的需求

水平切分会有一个patition key，通过patition key的查询能够直接定位到库，但是非patition key上的查询可能就需要扫描多个库了。

例如订单表，业务上对用户和商家都有订单查询需求：

Order(oid, info_detail)

T(buyer_id, seller_id, oid)

如果用buyer_id来分库，seller_id的查询就需要扫描多库。

如果用seller_id来分库，buyer_id的查询就需要扫描多库。

索引规则

1. 创建索引的语句要指定表空间。
2. 当查询的行数占整个表总行数的比例≤5%的时候，建立B*树索引效果比较好。否则，需要慎重考虑是否建立B*索引。
3. 一个表的索引不能建的过多，因为INSERT、DELETE、UPDATE均会重算。
4. 索引列包含的不同值很多时，应建立B*树索引。B*索引对AND/OR等条件逻辑组合查询的效率很低。

5. 索引列包含的不同值很少时，应建立位图索引。位图索引对AND/OR等条件逻辑组合查询的效率很低。

联机程序接口设计

联机接口设计原则

1. 兼容性
2. 正确性
3. 易用性
4. 可读性
5. 易学性
6. 文档有范例
7. 高效性
8. 内存小

接口通信区设计原则

1. 公用通信区：具有普遍性，且和接口具体功能无关的信息内容，如运行状态、交互控制方便的内容
2. 专用通信区：某一类接口的公用信息
3. 私用通信区：某一接口的专用信息

接口版本使用规则

接口数据检查

接口错误处理

接口数据加密

接口API

API是某个应用对外提供的，用于访问其能力的程序接口。其调用基于标准的网络通信和交互，与开发语言无关。

1. WEB API
2. URL API

批量程序设计

时间划分

1. 日间执行
2. 日终执行
3. 不定期

机制划分

1. 定期自动执行
2. 定期手工执行
3. 不定期半自动执行
4. 不定期手工执行

模块划分

自动化设计

健壮性原则

在批量处理中遇到数据出现错误时，首先要遵循不中断的原则，只有情节非常特殊的时候，才能使用中断的处理方法。

1. 批量的不中断设计：以下情况，建议避免中断处理

1）当天影响某类账户的处理，但是有补救措施。

2）只影响银行内部的账务处理。

2. 不中断错误信息的记载
3. 批量的中断设计
4. 断点再续及断点重做

性能设计

1. 避开联机高峰期，分多次批量
2. 避免资源竞争
3. 批量整合，一次扫描
4. 大SQL

批量设计的一般性原则

易用性原则

1. 无人工操作原则
2. 开放平台集中管理调度原则
3. 断点再续及断点重做原则
4. 容错处理原则
5. 数据检查原则
6. 重复批量的控制原则

系统的实施运行与维护

白盒测试

白盒测试的原则：

1. 保证一个模块中的所有独立路径至少被测试一次。
2. 对所有的逻辑判定均需测试取真和取假两种情况。
3. 在上下边界及可操作范围内运行所有循环。
4. 检查程序的内部数据结构，保证其结构的有效性。

白盒测试的分类：

1. 静态测试方法：不要求在计算机上实际执行所测试的程序，主要以一些人工的模拟技术对软件进行分析和测试，如代码检查法、静态结构分析法等。
2. 是通过输入一组预先按照一定的测试准则构造实际数据来动态运行程序，达到发现程序错误的过程。白盒测试中的动态分析技术主要有逻辑覆盖法和基本路径测试法。

单元测试

单元测试的实现方式：

1. 人工静态检查：就是通常所说的“代码走读”，主要是保证代码逻辑的正确性
2. 动态执行跟踪：就是把程序代码运行起来，检查实际的运行结果和预期结果是否一致

单元测试和白盒测试的标准：

1. 语句覆盖
2. 分支覆盖
3. 条件覆盖
4. 分支-条件覆盖
5. 条件组合覆盖
6. 路径覆盖

软件调试

1. 试探法
2. 回溯法
3. 对分查找法
4. 归纳法

系统维护

根据维护目的的不同，软件维护分为：

1. 正确性维护：改正在系统开发阶段已发生而系统测试阶段尚未发现的错误。工作量：17%-21%。
2. 适应性维护：对软件加以改造，使之适应于新的环境，为使软件产品在新的环境下仍能使用而进行的维护，称为适应性维护。
3. 预防性维护：略。
4. 完善性维护：对已有的软件系统增加一些在系统分析和设计阶段中没有规定的功能和性能特征

维护的副作用：

1. 编码副作用
2. 数据副作用
3. 文档副作用

系统的可维护性

软件可维护性可用下面的7个质量特性来衡量：

1. 可理解性
2. 可测试性
3. 可修改性
4. 可靠性
5. 可移植性
6. 可实用性
7. 效率

注解

关系模式

即Relation Schema。

关系模式可以被形式化的表示为：R（U，D，dom，F）。其中：

1. U表示属性名的集合
2. D表示域的集合
3. dom表示U中的元素到D中的元素所构成的全体映射的集合
4. F表示U中元素的相互依赖关系

函数依赖

设X,Y是关系R的两个属性集合，当任何时刻R中的任意两个元组中的X属性值相同时，则它们的Y属性值也相同，则称X函数决定Y，或Y函数依赖于X。

记作X→Y。

部分函数依赖

设X,Y是关系R的两个属性集合，存在X→Y，若X'是X的真子集，存在X'→Y，则称Y部分函数依赖于X。

完全函数依赖

设X,Y是关系R的两个属性集合，X'是X的真子集，存在X→Y，但对每一个X'都有X'!→Y，则称Y完全函数依赖于X。

传递函数依赖

设X,Y,Z是关系R中互不相同的属性集合，存在X→Y,Y→Z，其中Y !→X，则称Z传递函数依赖于X。

树节点的高度与深度

1. 节点的深度：节点a到根节点的长度，被称为节点的深度。
2. 节点的高度：以节点a为根节点的子树的所有叶节点到节点a的最长简单路径的长度，被称为节点a的高度。
3. 树的高度：根节点的高度被称为树的高度。

二叉搜索树

一棵二叉树满足：

1. 非叶子结点的左指针指向小于其关键字的结点。
2. 非叶子结点的右指针指向大于其关键字的结点。

那么这棵树被称为二叉搜索树。

平衡二叉搜索树

又称AVL树。

AVL树的性质：

1. 左子树与右子树高度之差的绝对值不超过1。
2. 树的每个左子树和右子树都是AVL树。

平衡因子

某节点的左子树与右子树的高度差即为该节点的平衡因子。

平衡二叉树搜索树的所有节点取值均为$-1, 0, 1$。

所有节点取值均为$-1, 0, 1$的二叉搜索树是平衡二叉树搜索树。

最小失衡子树

新插入的结点向上查找，以第一个平衡因子的绝对值超过 1 的结点为根的子树称为最小不平衡子树。

平衡二叉搜索树的插入（未完）

平衡二叉搜索树的删除（未完）

红黑树

满足以下要求的二叉搜索树被称为红黑树：

1. 每个节点要么是黑色，要么是红色。
2. 根节点是黑色。
3. 叶节点是黑色。
4. 每个红色结点的两个子结点一定都是黑色。
5. 任意一结点到每个叶子结点的路径都包含数量相同的黑结点。

值得注意的是：

1. 约束4和5，保证了红黑树的大致平衡：根到叶子的所有路径中，最长路径不会超过最短路径的2倍。
2. 默认新插入的节点为红色。因为父节点为黑色的概率较大，插入新节点为红色，可以避免颜色冲突。

红黑树的插入（未完）

红黑树的删除（未完）

B树

B树，即B-树。是一种平衡的多路查找树。

大多数平衡树的操作（查找、插入、删除，最大值、最小值等等）需要$O(h)$次磁盘访问操作，其中h是树的高度，并且$h = log(n)$。

对于B树来说，树的高度将不再是$log(n)$（其中n是树中的结点个数），而是一个我们可控的高度h。

B树的性质：

1. 所有的叶子结点都出现在同一层上，并且不带信息。(可以看做是外部结点，实际上这些结点不存在，指向这些结点的指针为空)。
2. 每个结点包含的关键字个数有上界和下界。结点包含的关键字下界是$t-1$，上界是$2t-1$。其中，$t \geq 2$被称为B树的最小度数。
3. 一个包含x个关键字的结点有$x+1$个孩子。
4. 一个结点中的所有关键字升序排列，两个关键字k_1和k_2之间的孩子结点的所有关键字 key 在(k_1, k_2)的范围之内。

B+树

B+树是B树的变种。

B+树的性质与B树大体相同，具体性质如下：

1. 所有的叶子结点都出现在同一层上，并且包含所有的关键字。叶子结点之间用链指针相连。
2. 每个结点包含的关键字个数有上界和下界。结点包含的关键字下界是$t-1$，上界是$2t-1$。其中，$t \geq 2$被称为B树的最小度数。
3. 一个包含x个关键字的结点有x个孩子。
4. 一个结点中的所有关键字升序排列，两个关键字k_1和k_2之间的孩子结点的所有关键字 key 在$[k_1, k_2)$的范围之内。

B*树

B*树是B+树的变形。

B*树将索引层以指针连接起来，使搜索取值更加快捷。

索引列

位图索引

主要针对大量相同值的列而创建。

举例说明：

Name	Gender	Marital
张三	男	已婚
李四	女	已婚
王五	男	未婚
赵六	女	离婚
孙七	女	未婚

对于该表，可以生成如下位图索引：

RowId	1	2	3	4	5	...
男	1	0	1	0	0	
女	0	1	0	1	1	

RowId	1	2	3	4	5	...
已婚	1	1	0	0	0	
未婚	0	0	1	0	1	
离婚	0	0	0	1	0	

当我们使用查询语句“select * from table where Gender=‘男’ and Marital=“未婚”;”的时候 ，进行如下运算：

RowId	1	2	3	4	5
男	1	0	1	0	0
and					
未婚	0	0	1	0	1
结果	0	0	1	0	0

7. 各种前沿技术（6%）

云计算

大数据

5G

人工智能

区块链

物联网

8. 软件工程（3%）

9. 项目管理（2%+3%）

生命周期管理

需求分析与设计概述

10. 产品管理（7%）

产品管理概述

产品营销策略

导入期

特点：成本高，利润少。

营销目标：创造品牌和知名度。

成长期

特点：产品销售量快速并稳定增长。

营销目标：市场份额最大化。

成熟期

特点：趋于稳定，增长率低。

营销目标：保护市场份额的前提下利润最大化。

衰退期

特点：市场吸引力明显降低。

营销目标：减少费用，尽量获利。

产品经营策略

1. 维持策略
2. 调整策略
3. 退出策略
4. 新研发策略

产品基础管理

产品目录

1. 产品名称
2. 产品分级分类信息
3. 产品属性信息
4. 产品代码

产品研发与跟踪评价

产品市场分析、营销与销售支持

客户分析

1. 锁定目标客户
2. 确定产品核心价值
3. 确定产品销售和推广方式

客户分析方法

1. 用户研究
2. 市场细分
3. 分析型客户关系管理系统
4. Profiler工具等

用户体验工作

用户体验方法

用户研究

桌面研究

分析行业现状等，形成目标产品的初步设想。

用户行为记录

记录用户生活和工作，了解买用户的基本行为和习惯。

深度访谈

更加深入了解。

产品设计方法