# 一 数据仓库

## 1.数据仓库分层

### 1.2 主题层

FDM主题层:  
团体

事件

协议

产品

财务

渠道

资产

公共

汇总

报表

## 2.遇到的困难

## 3.平时工作中如优化重复性的工作

## 4.给一个需求场景 简述如何设计开发

# 二 大数据

# 三 数据治理

# 四 Java基础

# 五 版本控制工具

$ git merge dev-20191119

# 六、面试记录

## PAJF面试

## 1. 挑选项目经历中最有价值的一个讲出来，如果让你重构，你会怎么样做才能做得更好？

### 2.数据库基础知识

文章参考地址：

<https://wenku.baidu.com/view/c73b15b17e21af45b207a850.html>

<https://www.jianshu.com/p/c65c4ec0fb3f>

<https://blog.csdn.net/dosthing/article/details/87954213>

#### （1） 数据库设计的三范式 原则及为了解决什么问题

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(英文Normal Form,缩写NF)。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式

在实际开发中最为常见的设计范式有三个：

1）.第一范式（确保每列保持原子性）

第一范式是最基本的范式。如果数据库表中的所有字段都是不可分解的原子值，就说明了该数据库表满足了第一范式。数据表中的每一列（字段），必须是不可拆分的最小单元，也就是确保每一列的原子性，而不是集合。

实例：如以下订单表，买家地址列并不符合第一范式，需要继续拆分



上表所示的订单遵循了第一范式的要求，这样对用户使用城市进行分类的时候就非常方便，也提高了数据的性能

2）.第二范式（确保表中的每列都和主键相关）

第二范式在第一范式的基础之上更进一步。第二范式需要确保数据库中的每一列和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，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。

满足1NF的基础上，要求：表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系（一个表只描述一件事情）。第二范式消除表的无关数据。

将上述订单表拆分成多张表，拆分后如下图



3）.第三范式（确保每列都和主键列直接相关，而不是间接相关）

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关，列和列之间不存在相互依赖关系。要求一个关系中不包含已在其它关系已包含的非主关键字信息。

满足2NF的基础上，任何非主属性不依赖于其他非主属性（在2NF的基础上消除依赖传递）（也表明不允许数据存在冗余的现象）。

上述表中，虽然单价和订购数量可以计算出总价，但是单价是直接和商品相关的，并不是和订单直接相关，所以不满足第三范式，需要继续拆分



**（2）数据库的事务性**

除了数据库设计的三大范式之外，事务处理也是保证数据完整性的重要手段。事务是单独的工作单元，该单元可以包含多个操作以完成一个完整的任务。锁是在多用户环境中对数据访问的限制。事务和锁确保了数据的完整性。

**事务处理**

提交commit，当所有的操作步骤都被完整执行后，称该事务被提交。

回滚rollback，由于某一操作步骤执行失败，导致所有都没有被提交，则事务必须回滚，即回到事务执行前的状态。

**事务ACID属性**

事务处理的特性，每一个事务都有他们所共有的特性，叫做ACID特性，分别是原子性atomicity、一致性consistence、隔离性Isolation、持久性Durability

1）原子性，事务的原子性表示事务执行过程中，把事务作为一个工作单元处理，一个工作单元可能包含若干个操作步骤，每个操作步骤都必须完成才算完成，若因任何原因导致其中的一个步骤操作失败，则所有操作步骤失败，前面的步骤必须回滚。

2）一致性，事务的一致性保证数据处于一致状态。如果事务开始时处于一致状态，则事务结束时也应处于一致状态，不管事务成功还是失败。

3）隔离性，事务的隔离性保证事务访问的任何数据都不会受到其他事务所做的任何改变的影响，直到该事务完成。

4）持久性，事务的持久性保证假如事务执行，则它在系统中产生的结果应该是持久的。

（3）数据库五大约束

1、主键约束(Primay Key)

唯一性，非空性

2、唯一约束(Unique)

唯一性，可以空，但只能有一个

3、检查约束(Check)

对该列数据的范围、格式的限制(如：年龄、性别等)

4、默认约束(Defaut)

该数据的默认值

5、外键约束（Foreign Key）

需要建立两表间的关系

#### （2） 数据库设计的拉链表 必须字段

本文链接：<https://blog.csdn.net/weixin_45399233/article/details/100601551>

本文链接：<https://blog.csdn.net/u012965373/article/details/81515463>

本文链接：https://blog.csdn.net/fhy36897/article/details/89553784

增量数据与历史库做成拉链表

首先介绍几个表概念：

**全量表**：每天的所有的最新状态的数据

**增量表**：每天的新增数据

**拉链表**：维护历史状态，以及最新状态数据

**流水表**：对于表中的每一个记录都会修改，可以用于反映实际记录的变更

拉链表VS流水表：

拉链表：通常是对账户信息的历史变动进行处理保留的结果，用于统计业务相关情况

流水表：每天的交易形成的历史，用户统计账户及客户的情况

在数据仓库的数据模型设计过程中，经常会遇到这样的需求：

1）. 数据量比较大；

2）.表中的部分字段会被update，如用户的地址，产品的描述信息，订单的状态等；

3）.需要查看一个时间点或者时间段的历史快照信息，比如，查看一个订单早历史某一个时间点的状态，比如，查看某一个用户在过去某一个时间内，更新过几次等等；

4）变化的频率和比例不是很大，比如，总共有1000万的会员，每天新增和发生变化的有10万左右；

5）如果对这个表每天都保留一份全量，那么每次全量中会保存很多不变的信息，对存储是极大的浪费。

拉链历史表，技能满足反应数据的历史状态，又可以最大程度的节省存储。.

拉链表案例：

1）有一张订单表，2019-06-20 这天里面有三条订单数据：



2）到了2019-06-21这天，表中有5条记录：



3）到了2019-06-22日，表中有6条记录：



这种存储方式有什么问题呢？

1.只保留一份全量，则数据和6月22日的记录一样，如果要查看6月21日订单001的状态，则无法满足。

2、每天都保留一份全量，则数据仓库中的该表共有14条记录，但好多记录都是重复保存，没有任何变化，如订单002,004，数据量大了会造成很大的存储浪费；

如果设计成历史拉链表，如下：



拉链表中字段解释：

1）、dw\_begin\_date 表示该条记录的生命周期的开始时间，dw\_end\_date表示该条记录的生命周期结束时间；

2）dw\_end\_date=’9999-12-31’表示该条记录目前处于有效状态；

3）如果查询当前所有有效的记录，则select \* from order\_his where dw\_end\_date =’9999-12--31’

4)如果查询2019-06-21的历史数据，则select \* from order\_his where dw\_begin\_date <= ‘2019-06-21’ and dw\_end\_date >= ‘2019-06-21’,这条语句会查询到以下记录：



可以看出，这样的历史拉链表，技能满足对历史数据的要求，又能很大程度的节省存储资源。

举例：













**需要参考博客，讲的很清楚，自己整理的由于篇幅限制截图不全。**



**增量抽取**

每天，从源系统订单表中，将前一天的增量数据抽取到ODS层的增量数据表。这里的增量需要通过订单表中的创建时间和修改时间来确定：



**增量刷新历史数据**

从2019-08-22开始，需要每天正常刷新前一天（2019-08-21）的增量数据到历史表。

**第一步，通过增量抽取，将2019-08-21的数据抽取到ODS：**



ODS增量表中2019-08-21的数据如下：



**第二步，通过DW历史数据(数据日期为2019-08-20)，和ODS增量数据(2019-08-21)，刷新历史表：**













#### （3） 数据库调优如何看调优的，怎么解决的，看什么字段

原文链接：<https://blog.csdn.net/u010520146/article/details/81161762>

下面介绍几种常见的SQL的调优方式,供借鉴.

**1).创建索引**

1.要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引

2.(1)在经常需要进行检索的字段上创建索引，比如要按照表字段username进行检索，那么就应该在姓名字段上创建索引，如果经常要按照员工部门和员工岗位级别进行检索，那么就应该在员工部门和员工岗位级别这两个字段上创建索引。

(2)创建索引给检索带来的性能提升往往是巨大的，因此在发现检索速度过慢的时候应该首先想到的就是创建索引。

(3)一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。

**2).避免在索引上使用计算**

在where字句中，如果索引列是计算或者函数的一部分，DBMS的优化器将不会使用索引而使用全表查询,函数

属于计算的一种,同时在in和exists中通常情况下使用EXISTS，因为in不走索引

效率低：

select \* from user where salary\*22>11000(salary是索引列)

1

效率高：

select \* from user where salary>11000/22(salary是索引列)

**3).使用预编译查询**

程序中通常是根据用户的输入来动态执行SQL，这时应该尽量使用参数化SQL,这样不仅可以避免SQL注入漏洞

攻击，最重要数据库会对这些参数化SQL进行预编译，这样第一次执行的时候DBMS会为这个SQL语句进行查询优化

并且执行预编译，这样以后再执行这个SQL的时候就直接使用预编译的结果，这样可以大大提高执行的速度。

**4).调整Where字句中的连接顺序**

DBMS一般采用自下而上的顺序解析where字句，根据这个原理表连接最好写在其他where条件之前，那些可以   
过滤掉最大数量记录。

**5)尽量将多条SQL语句压缩到一句SQL中**

每次执行SQL的时候都要建立网络连接、进行权限校验、进行SQL语句的查询优化、发送执行结果，这个过程   
是非常耗时的，因此应该尽量避免过多的执行SQL语句，能够压缩到一句SQL执行的语句就不要用多条来执行。

**6).用where字句替换HAVING字句**

避免使用HAVING字句，因为HAVING只会在检索出所有记录之后才对结果集进行过滤，而where则是在聚合前

刷选记录，如果能通过where字句限制记录的数目，那就能减少这方面的开销。HAVING中的条件一般用于聚合函数

的过滤，除此之外，应该将条件写在where字句中。

**7）使用表的别名**

当在SQL语句中连接多个表时，请使用表的别名并把别名前缀于每个列名上。这样就可以减少解析的时间并减   
少哪些友列名歧义引起的语法错误。

**8）.用union all替换union**

当SQL语句需要union两个查询结果集合时，即使检索结果中不会有重复的记录，如果使用union这两个结果集

同样会尝试进行合并，然后在输出最终结果前进行排序，因此如果可以判断检索结果中不会有重复的记录时候，应

该用union all，这样效率就会因此得到提高。

**9).考虑使用“临时表”暂存中间结果**

简化SQL语句的重要方法就是采用临时表暂存中间结果，但是，临时表的好处远远不止这些，将临时结果暂存在临时表，后面的查询就在tempdb中了，这可以避免程序中多次扫描主表，也大大减少了程序执行中“共享锁”阻塞“更新锁”，减少了阻塞，提高了并发性能。

但是也得避免频繁创建和删除临时表，以减少系统表资源的消耗。

**10).只在必要的情况下才使用事务begin translation**

SQL Server中一句SQL语句默认就是一个事务，在该语句执行完成后也是默认commit的。其实，这就是begin tran的一个最小化的形式，好比在每句语句开头隐含了一个begin tran，结束时隐含了一个commit。

有些情况下，我们需要显式声明begin tran，比如做“插、删、改”操作需要同时修改几个表，要求要么几个表都修改成功，要么都不成功。begin tran 可以起到这样的作用，它可以把若干SQL语句套在一起执行，最后再一起commit。 好处是保证了数据的一致性，但任何事情都不是完美无缺的。Begin tran付出的代价是在提交之前，所有SQL语句锁住的资源都不能释放，直到commit掉。

可见，如果Begin tran套住的SQL语句太多，那数据库的性能就糟糕了。在该大事务提交之前，必然会阻塞别的语句，造成block很多。

Begin tran使用的原则是，在保证数据一致性的前提下，begin tran 套住的SQL语句越少越好！有些情况下可以采用触发器同步数据，不一定要用begin tran。

**11).尽量避免使用游标**

尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

**12).用varchar/nvarchar 代替 char/nchar**

尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（NULL也包含在内），都是占用 100个字符的空间的，如果是varchar这样的变长字段， null 不占用空间。

**13）.查询select语句优化**

1.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，

如：

select id from t where num is null

1

可以在num上设置默认值0，确保表中num列没有null值，

然后这样查询：

select id from t where num=0

select id from t where num=10 or num=20

1

2

可以这样查询：

select id from t where num=10

union all

select id from t where num=20

1

2

3

4.不能前置百分

select id from t where name like ‘%abc%’

1

若要提高效率，可以考虑全文检索。

select id from t where num in(1,2,3)

1

对于连续的数值，能用 between 就不要用 in 了：

select id from t where num between 1 and 3

1

6.如果查询的两个表大小相当，那么用in和exists差别不大。

in：

例如：表A（小表），表B（大表）

select \* from A where cc in (select cc from B) 效率低，用到了A表上cc列的索引；

select \* from A where exists(select cc from B where cc=A.cc) 效率高，用到了B表上cc列的索引。

1

2

相反的

select \* from B where cc in (select cc from A) 效率高，用到了B表上cc列的索引；

select \* from B where exists(select cc from A where cc=B.cc) 效率低，用到了A表上cc列的索引。

**14）.更新Update语句优化**

1.如果只更改1、2个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志

**15）. 删除Delete语句优化语句**

1.最高效的删除重复记录方法 ( 因为使用了ROWID)例子：

DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);

**16）.插入Insert语句优化**

1.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

（4）给一个场景如何开发

（5）常用数据库语句，如何写，分组排序取最大/小

## 3.JAVA会不会

（1）spring /springMVC 的区别，有没有接触过springboot

## 4.SHELL脚本

（1） 定时语句/命令怎么写

（2） 查找一个文件中某个字符串出现的次数

## 5.其他

（1） 平时 有没有打建过系统环境，SVN,服务器之类的，如何搭建的

## 6.为什么要换工作，平时有没有对新技术进行钻研

# 七、读书笔记

第一章 决策支持系统的发展

当数据从操作型环境传向数据仓库环境时，需要对数据进行集成。

第二章 数据仓库环境