# 宽表和窄表

一、宽表

（1）宽表：从字面意义上讲就是字段比较多的数据库表。通常是指业务主题相关的指标、维度、属性关联在一起的一张数据库表。由于把不同的内容都放在同一张表存储，宽表已经不符合三范式的模型设计规范，随之带来的主要坏处就是数据的大量冗余，与之相对应的好处就是查询性能的提高与便捷。这种宽表的设计广泛应用于数据挖掘模型训练前的数据准备，通过把相关字段放在同一张表中，可以大大提高数据挖掘模型训练过程中迭代计算时的效率问题。（一句话，空间换时间，便于训练迭代、减少表关联数量，修改少量数据时不需要该多张表）

（2）“宽表”特点：一个表就是一个class，每个字段就是一个filed，操作简单。但不便扩展。

二、窄表

（1）窄表：严格按照数据库设计三范式。尽量减少数据冗余，但是缺点是修改一个数据可能需要修改多张表。

（2）“窄表”特点：方便扩展，能适应各种复杂的数据结构（树形、继承等），无论有多少配置，都不用修改表结构。但代码逻辑可能需要包装一下。

宽表和窄表的建设该如何选择？

这个问题相信纠结了很多从是数据库开发、数据仓库开发和后台开发人员；单单考虑这个问题，难给出一个绝对的答案；本人从事数据仓库开发工作到现在已经有一年半时间了，对于这个问题，我也曾经纠结过，但是是否有绝对的答案呢？事实上任何东西都没有绝对的说法。

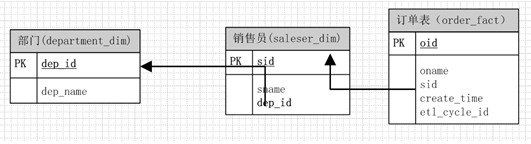
考虑这样的一个问题，一个公司有这样的一个需求：

设计销售领域的订单事实表，该事实表应该包含哪些维度和度量？事实表和维表该分别如何去设计？

好了，我们把关键信息拿出来，首先我们要有维度包括：销售员、销售员所属部门、下订单的时间；度量：销售量；

那么，订单事实表，其实就是一个商品销售的清单；

依照这个思路，我们建立的第一个模型可能是以下这样的：



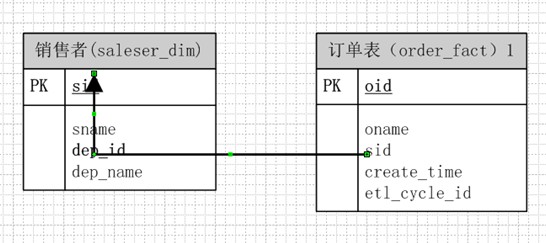
单单看上去，貌似是符合我们的问题的需要，而且符合数据库的范式设计：没有冗余字段；但是情况真的就是这样吗？

答案是否定的，确实对于一般的OLTP系统而言这样的表设计确实减少了冗余和，增删改查等操作也很方便，但是往往对于我们的统计系统、OLAP、数据挖掘而言，情况却并非如此，举个例子：我们要统计每个部门各自的销售量为多少？那么对于上表,sql是这样的：

select a.\*,b.sid into #dep\_saleser from department a,saleser\_dim b on a.dep\_id = b.dep\_id;

select count(1),a.dep\_name from #dep\_saleser a,order\_fact b on a.sid=b.sid group by a.dep\_name;

对于这么一个简单的需求已经要写两了sql去实现了，其实数据库表模型的的设计是灵活的，我们完全可以根据我们的业务去设计我们的数据表；考虑到部门和销售员可以是同属于销售者这个维度，只是他们是有上下级别关系的那么依照这个思路，我们的模型可以建立为下面这样：



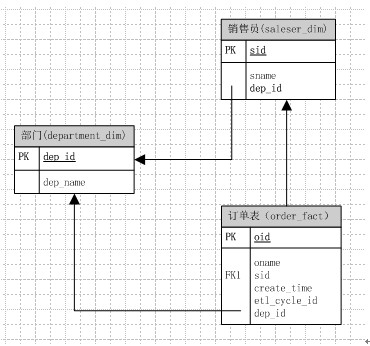
那么统计每个部门各自的销售量，可以用如下sql去实现：

select count(1),a.dep\_name from saleser\_dim a,order\_fact b

on a.sid=b.sid group by a.dep\_name;

确实对于这个模型而言，有些情况下会出现冗余（填写用户，没有填写部门；填写部门没填写用户）；但是对于提取数统计的逻辑又相对来说要简单了好多；

考虑到要实现取数简单，我们还可以想出另外一种方法：



　　看上去好像不错哦~~，取数据也就一句sql就搞掂了，但是却是最最槽糕的情况，有可能一个销售员，前几天登记的部门是a，但是其实他的所属于的部门为b，那么对于上面这个模型，我们得改动销售员和订单表；而对于上面的其他两个模型都仅仅需要改动一张表就行了，造成查询数据部一致往往也就是这种数据模型所造成的。

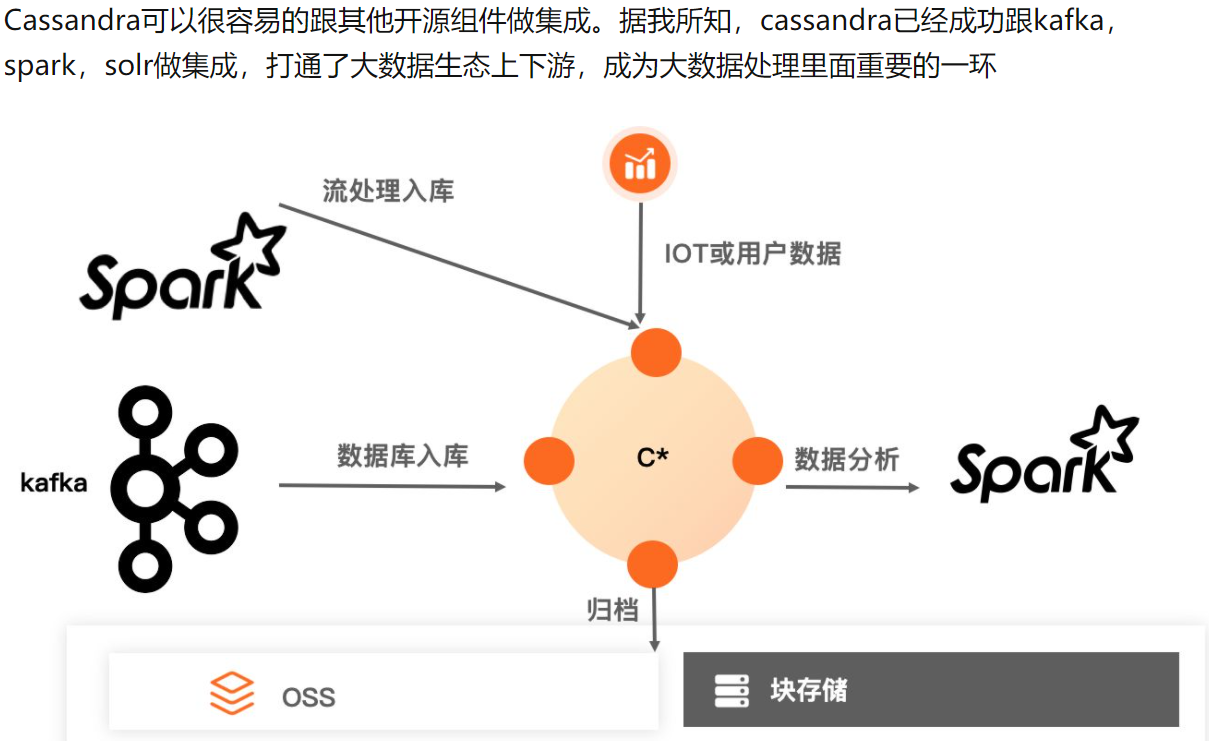
    所谓的宽表就是字段比较多的表，包含的维度层次比较多，造成冗余也比较多，毁范式设计，但是利于取数统计，而窄表往往对于OLTP比较合适，符合范式设计原则；

<https://www.cnblogs.com/Leo_wl/p/8515794.html>

# Cassandra

Cassandra是云原生和微服务化场景中最好的NoSQL数据库。

<https://www.cnblogs.com/cjsblog/p/12863749.html>



作者：阿里云网站  
链接：https://www.zhihu.com/question/26410789/answer/820606497  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**什么时候应该考虑使用Cassandra**

每个数据库服务器都是为满足特定的设计目标而构建，这些设计目标定义了数据库适合和不适合的场景。  
Cassandra的设计目标如下：

* 分布式：在多个服务器节点上运行。
* 线性扩展：通过添加节点实现水平扩容
* 全球分布：群集可以在地理上分布。
* 写优于读：写入比读取快一个数量级。
* 民主对等架构：没有主/从。
* 支持分区容忍度和可用性而不是一致性：最终一致（参见CAP定理：https：//[http://en.wikipedia.org/wiki/CAP\_theorem](https://link.zhihu.com/?target=http://en.wikipedia.org/wiki/CAP_theorem" \t "_blank)。）
* 通过主键支持快速目标读取：关注主键读取，其他路径次优。
* 支持具有生命周期的数据：Cassandra数据库中的所有数据都具有已定义的生命周期，生命周期到期后自动删除数据。

功能列表中没有关于ACID，关系型库中常见聚合等功能。此时您可能会想，“那它有什么用？”ACID，Join和聚合对于所有数据库至关重要。没有ACID意味着没有原子，没有原子操作，你如何确保任何事情都正确发生，如何保证一致。Cassandra的的确确确实不能保证，所以如果您考虑选型某数据库来跟踪银行的账户余额，您可能应该考虑其他选择。

## 理想的cassandra使用场景

事实证明，Cassandra对某些应用程序非常有用。  
理想的Cassandra应用程序具有以下特征：

* 写入大幅度超出读。
* 数据很少更新，并且在进行更新时它们是幂等的。
* 通过主键查询，非二级索引。
* 可以通过partitionKey均匀分区。
* 不需要Join或聚合。

我最推荐使用Cassandra的一些好场景是：

* 交易日志：购买，测试分数，观看的电影等。
* 存储时序数据（需要您自行聚合）。
* 跟踪几乎任何事情，包括订单状态，包裹等。
* 存储健康追踪数据。
* 气象服务历史。
* 物联网状态和事件历史。
* 汽车的物联网数据。
* 电子邮件

作者：阿里封神  
链接：https://www.zhihu.com/question/26410789/answer/818427511  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

阿里云目前已经提供的 云cassandra服务，[https://www.aliyun.com/product/cds](https://link.zhihu.com/?target=https://www.aliyun.com/product/cds" \t "_blank)

cassandra是比较适合互联网在线应用的。 宽表市场排名： 排名第一。且 Datastax、ScyllaDB、CosmosDB都兼容CQL，已成事实标准

### 理由一：Cassandra为互联网业务而生

如果业务系统有事务的要求，我们建议使用MySQL或者阿里云POLARDB数据库，其提供的事务特性及SQL能力更匹配BOSS、CRM、ERP等业务需求。  
而对于互联网业务有如下特点：极致在线、高并发、大容量、可调的一致性、灵活扩展，MySQL并非最佳选择，Cassandra应运而生。

* 极致在线：支持多机房部署，单节点/单机房故障时，业务不中断。
* 扩展性强：支持从160GB到10PB的容量，支持从数千QPS到数千万的QPS，支持实例规格扩展，支持从单节点到多节点，支持从单机房到多机房，还将提供从同城到异地等多种扩展能力。匹配业务系统从小到大持续发展的特点，既满足不同发展时期对性能、成本、可靠性、灾备等不同要求，又可保障业务的平滑演进及连续性。
* 可调一致性：支持多种读写一致性级别，用户可依据自身业务系统特点灵活配置。比如对于物联网等业务，可以放宽一致性要求以获取更高的性能；对一致性要求较高的业务，则可选择QUORUM等强一致性级别。  
  如果您有这样的业务：聊天消息、新闻推送、历史订单、对象存储、购物车、计费系统、评论等，以及数据驱动的业务如风控、推荐、用户画像、物联网、日志分析等，阿里云Cassandra数据库是一个极好的选择。

### 理由二：开发者&DBA容易上手

一个熟悉MySQL的开发者，可以在半天内完全掌握Cassandra用法；一个熟悉MySQL的DBA，可以在一天内掌控Cassandra。Cassandra是一个易用易运维的数据库，有如下优点：

* CQL借鉴SQL语法，方便MySQL及传统数据库开发者直接使用
* 支持安全认证、SSL及完整鉴权体系，让DBA更省心
* 商业化前会支持备份恢复的能力，让DBA更放心
* 支持完整的索引服务，比如localindex、MV视图、SASI全文索引，方便开发者使用
* 原生提供各类语言的客户端：Java、Python、PHP、.Net、Nodejs等原生客户端(非thrift模式)连接Cassandra，性能跟JAVA客户端一致

### 理由三：大数据存储的利器、支持搭配X-Pack Spark

Spark是目前主流默认的大数据处理引擎，Cassandra是目前主流的BigData NoSQL数据库。Cassandra+Spark可以处理风控、推荐、物联网等很多数据驱动的业务。阿里云数据库团队也提供了X-Pack Spark引擎支持Cassandra，满足用户诉求。

Apache Cassandra™在高负载下提供了更高的性能，在许多用场景中都超过了它的NoSQL数据库竞争对手。

Apache Cassandra: 高度可伸缩、高性能的分布式数据库，设计用于处理许多商用服务器上的大量数据，提供高可用性，没有单点故障。

Apache HBase: 基于谷歌的BigTable的开源、非关系型、分布式数据库，是用Java编写的。它是Apache Hadoop项目的一部分，在HDFS上运行，为Hadoop提供类似于BigTable的功能。

MongoDB: 跨平台的面向文档的数据库系统，避开了传统的基于表的关系数据库结构，转而使用具有动态模式的类JSON文档，从而使数据在某些类型的应用程序中的集成更加容易和快捷。

Couchbase: 为交互式应用程序优化的分布式NoSQL面向文档的数据库。

Apache Cassandra™在高负载下提供了更高的性能，在许多用场景中都超过了它的NoSQL数据库竞争对手。

Apache Cassandra: 高度可伸缩、高性能的分布式数据库，设计用于处理许多商用服务器上的大量数据，提供高可用性，没有单点故障。

Apache HBase: 基于谷歌的BigTable的开源、非关系型、分布式数据库，是用Java编写的。它是Apache Hadoop项目的一部分，在HDFS上运行，为Hadoop提供类似于BigTable的功能。

MongoDB: 跨平台的面向文档的数据库系统，避开了传统的基于表的关系数据库结构，转而使用具有动态模式的类JSON文档，从而使数据在某些类型的应用程序中的集成更加容易和快捷。

Couchbase: 为交互式应用程序优化的分布式NoSQL面向文档的数据库。

有网友说：

我目前就把一个十几台机器的Cassandra集群当做一个超大型的k-v数据库在使用，非常满意它的性能。

# 优缺点

无聚合：较新版本的Cassandra对具有单个分区的聚合的支持有限。这用途非常有限。由于Cassandra是键值存储，因此即使可能完成SUM，MIN，MAX，AVG和其他聚合之类的操作也非常耗费资源。如果您的应用程序需要进行临时分析，那么Cassandra可能不适合您。

不可预测的性能：由于Cassandra具有许多不同的异步作业和后台任务，这些作业和后台任务不是用户计划的，因此性能可能不可预测。这意味着您可能会看到与查询或查询量无关的性能影响。这会使故障排除性能问题变得相当困难。

基于JVM：是的，我知道它在上面被列为“ Pro”，但有一些警告。JVM虽然速度很快，但仍然是垃圾收集语言。这意味着内存管理是由语言本身而不是应用程序完成的。对于许多用户而言，这永远不是问题，但是如果您获取大量信息（无论是请求量还是数据大小），都需要调整JVM以适合实现的特定需求。这意味着需要具备数据库编写语言所需要的专业知识和知识。

————————————————

版权声明：本文为CSDN博主「或许快要下雪了吧」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_40647378/article/details/105962764

TODO

<https://github.com/apache/cassandra>

体验一下

# RDBMS

1 sql语句基础

sql（结构化查询语言）分为两个部分：DML，DDL

DML 数据操作语言(查询和更新指令)

指令 含义

select 获取数据

update 更新数据

delete 删除数据

insert 插入数据

DDL 数据定义语言(create / alter / drop)

创建/修改/删除 database/table

创建/删除 index

2 聚合函数

聚合函数 count 配合 group by / having

mysql聚合函数如：count(), max(), min(), sum()等

group by 将数据库的数据用’by’后面接的规则进行分组

having是分组（group by）后的筛选条件；

where则是在分组前筛选

select

activity\_name, count(\*) as count

from

draw\_relation\_record

GROUP BY

activity\_name

HAVING

count(activity\_name) > 4

1

2

3

4

5

6

7

8

select 与 group by配合使用时

在有group by操作中，select后面接的结果集字段只有两种：要么就只有group by后出现的字段，要么就是group by后出现的字段＋聚合函数的组合

select 的内容除固定字段外，还有聚合函数时，需用group by

常用的五种聚合函数：min()求列中最小数值，max()求列中最大数值，avg()求平均值，sum()求列中字段对应数值的总和，count()求列的总条数

3 join连接

sql中的连接查询有三种方式：内连接，外连接，交叉连接

连接方式 场景

inner join 有效连接，两张表中都有的数据

left join 有左显示

right join 有右显示

full join 全连接，两张表的所有数据

cross join 两个表进行N\*M的组合, 笛卡尔积

如：

select Person.FirstName, Person.LastName, Address.City, Address.state

From Person

Left join Address

on Person.PersonId=Address.PersonId

on 和 where的区别

如果查询条件查询后才join连接，应该把查询条件放在ON后面；

如果要连接完毕后才筛选，应该把条件放在where后面

<https://blog.csdn.net/zxiang_123/article/details/104147776>

# MongoDB

使用场景

# 建关系型数据库的三范式

<https://www.huaweicloud.com/articles/5d645419dbe255f95ac3a96e4009a28d.html>

构造数据库必须遵循一定的规则。在关系数据库中，这种规则就是范式。范式是符合某一种级别的关系模式的集合。关系数据库中的关系必须满足一定的要求，即满足不同的范式。目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、第四范式（4NF）、第五范式（5NF）和第六范式（6NF）。满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多要求的称为第二范式（2NF），其余范式以次类推。一般说来，数据库只需满足第三范式（3NF）就行了。下面我们举例介绍第一范式（1NF）、第二范式（2NF）和第三范式（3NF）。

3.4.1 第一范式（1NF） 在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。 所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。例如，对于图3-2 中的员工信息表，不能将员工信息都放在一列中显示，也不能将其中的两列或多列在一列中显示；员工信息表的每一行只表示一个员工的信息，一个员工的信息在表中只出现一次。简而言之，第一范式就是无重复的列。

3.4.2 第二范式（2NF） 第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。如

图3-2 员工信息表中加上了员工编号（emp\_id）列，因为每个员工的员工编号是惟一的，因此每个员工可以被惟一区分。这个惟一属性列被称为主关键字或主键、主码。 第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

3.4.3 第三范式（3NF） 满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在图3-2

的员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。

简单的解释：

最基本的数据库范式有三个，第一范式(1NF)，第二范式(2NF)和第三范式(3NF)，分别定义如下，

1NF：如果关系模式R中的每个属性都是单值的，则称R∈1NF。

2NF：如果关系模式R∈1NF，且所有的非主属性完全函数依赖于(每一个候选)码，则称R∈2NF。

3NF：如果关系模式R∈2NF，且不存在非主属性传递函数依赖于码的情况，则称R∈3NF。