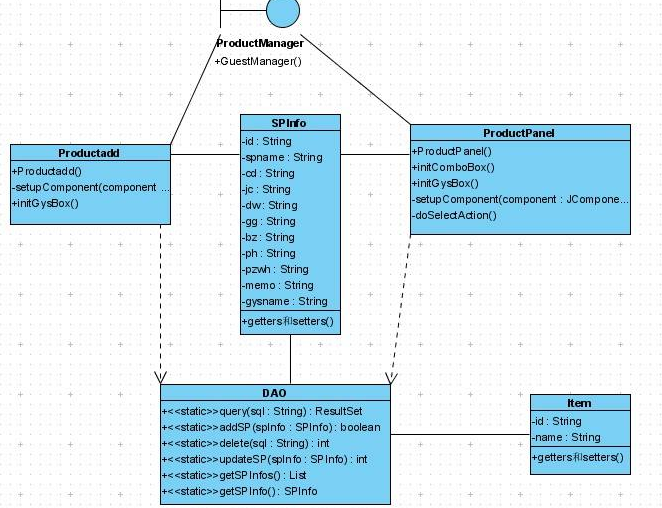
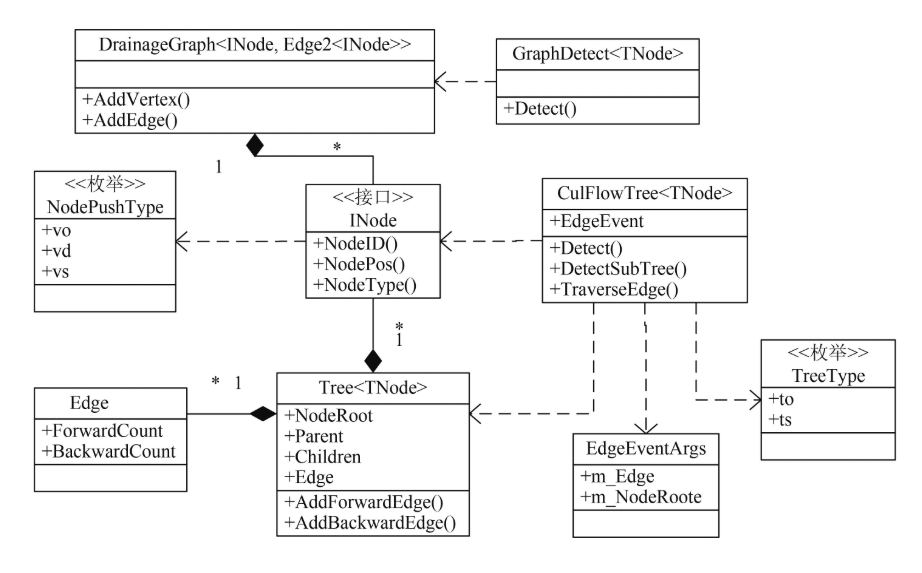
# 质量检测溯源信息化系统

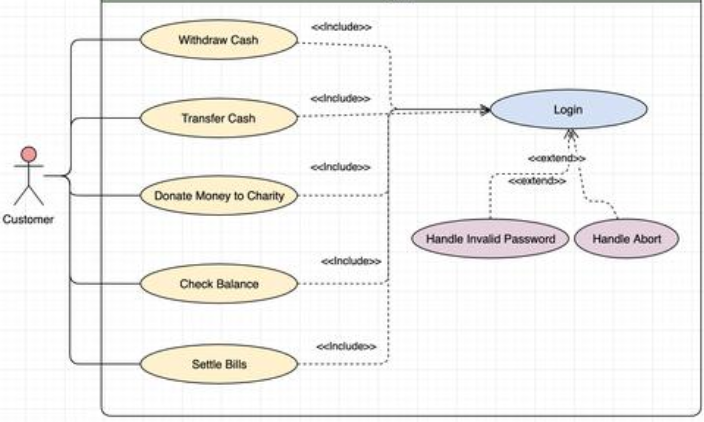
# 1、系统总体结构

## 类图：

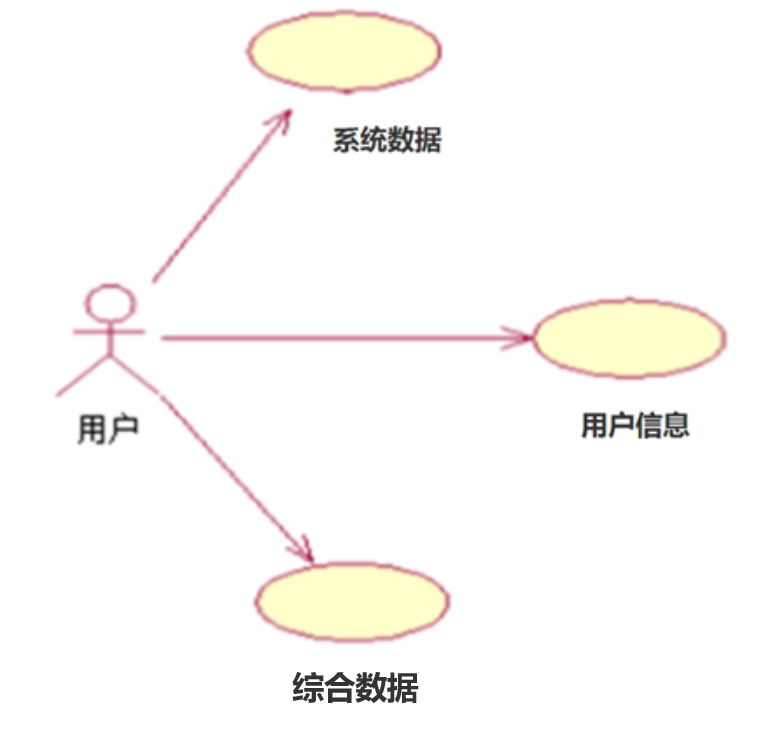


# 2、数据库设计

## 1.E-R图



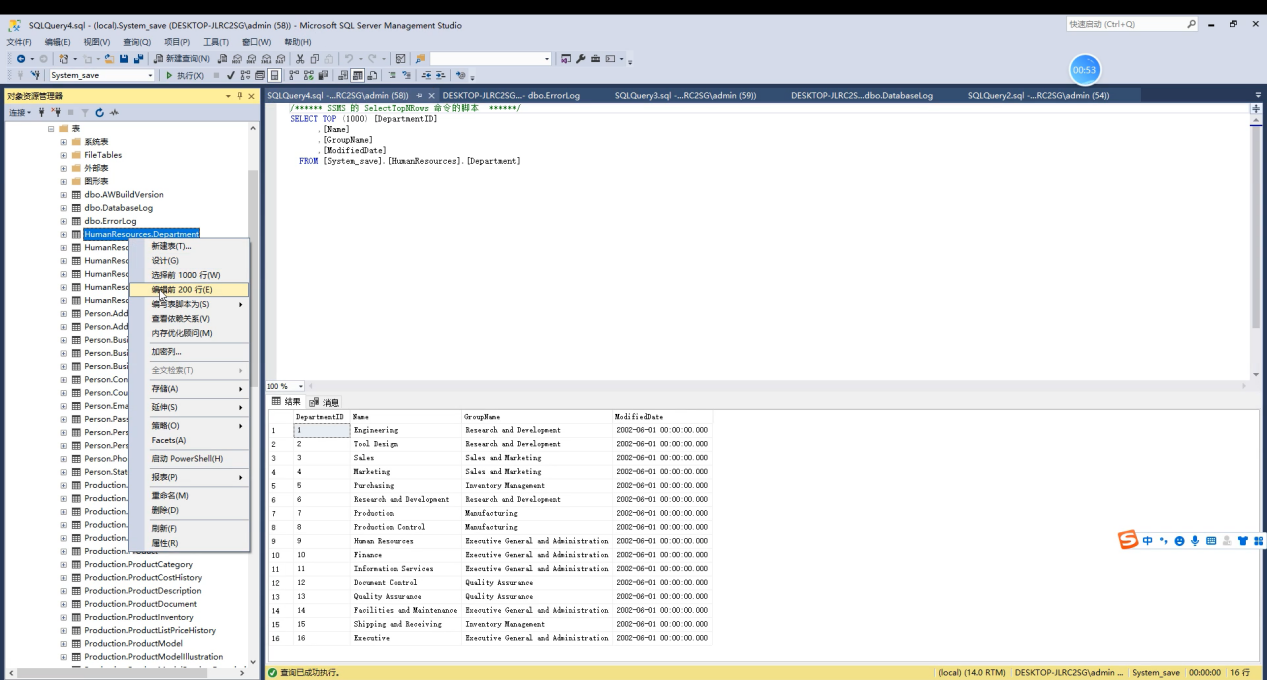
## 2.用例图

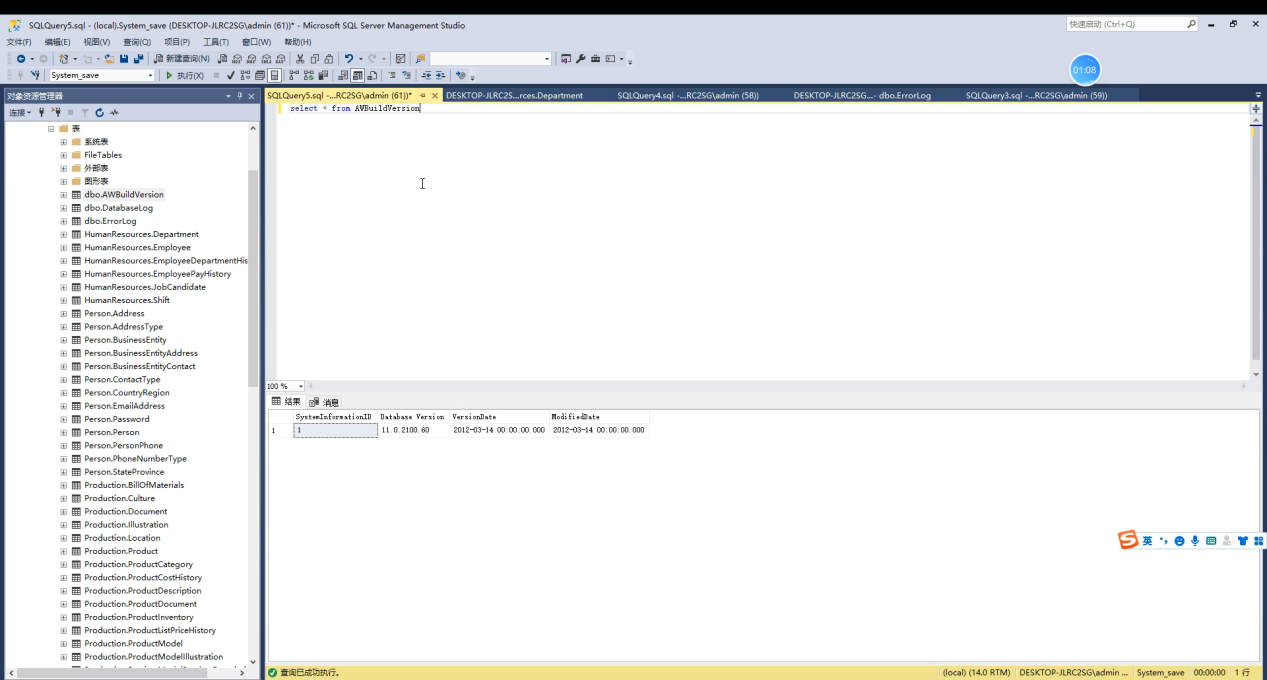


质量检测溯源信息化系统

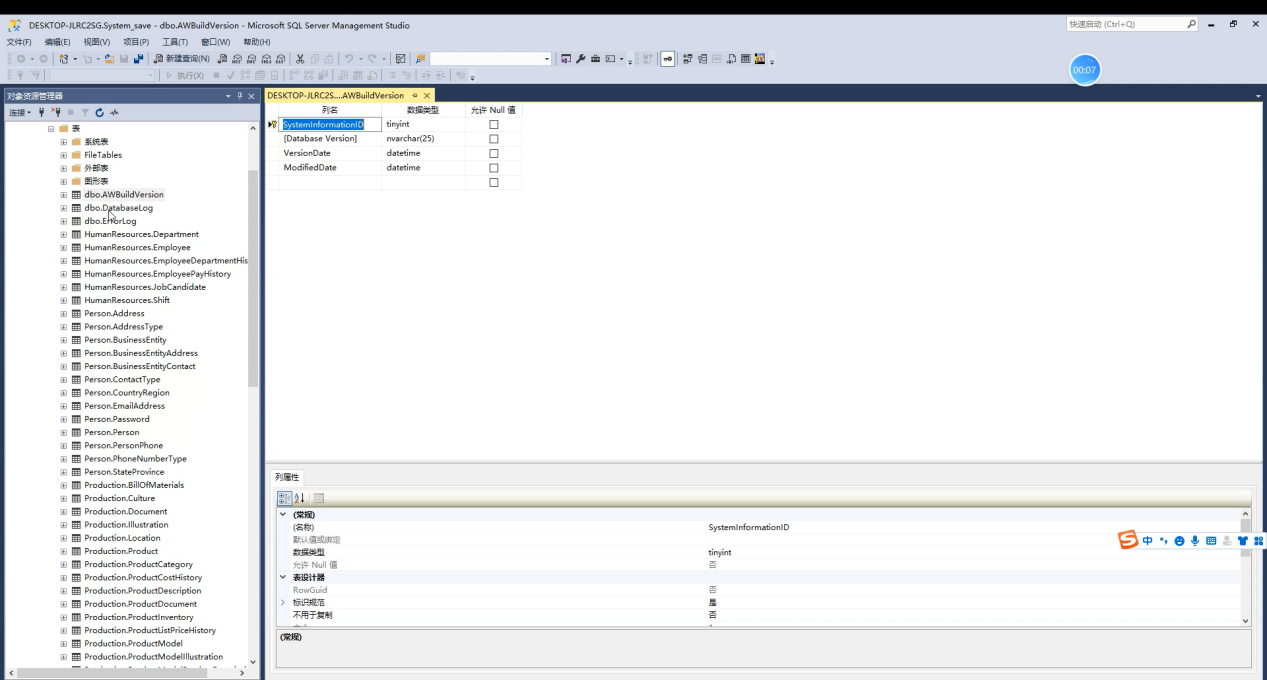
**数据库操作:**

记录查询：





数据库字段设计



# 适用范围

本规范的适用人员范围包括本公司的所有应用开发人员以及在实验室承接数据库应用开发的软件人员。

本规范的适用IT范围包括数据库对象的命名规范、设计原则、SQL语句的设计和使用、SQL语句的性能优化建议、其他与性能有关的设计原则以及设计工具的选择。

# 命名规范

## 通用规范

1. 不同的数据库产品对对象的命名有不同的要求，因此，数据库中的各种对象的命名、后台程序的代码编写应采用大小写敏感的形式，便于应用系统适应不同的数据库平台；
2. 采用英文单词或英文短语（包括缩写，如果没有约定的缩写，则采用该单词前4个字母）作为名称，不能使用无意义的字符或汉语拼音；
3. 每个数据库对象表，视图，字段，存储过程，不要用任何数据库的关键词作为名称。凡是能写注释的地方，应该写上注释，以便查看；
4. 名称应该清晰明了，能够准确表达事物的含义，最好可读，遵循“见名知意”的原则；
5. 长度不要超过30个字符，在PD设置生成时，部分数据库对象会自动截取到30位；
6. 所有名称的字符范围为：A-Z, a-z, 0-9 和\_(下划线)，共63个字符。不允许使用其他字符作为名称（注释除外），尽量不要使用0-9。

## 数据库(Database)的命名规范

数据库名称采用项目名称或项目名称缩写，名称全为大写，例如：CAPP、PDM。

## 表(Table)的命名规范

1. 数据表名由前缀、模块名称与实际名称组成，以\_(下划线)衔接；
2. 前缀使用大写字母T，表示表；
3. 模块名称用来表示某种从属关系，表示这种从属关系的一个主要目的是，从表名能够大概知道如何去找相关的人员；
4. 实际名称采用可以明确表示该表意义的英文单词或单词组合，每个单词首字母大写；
5. 表名中含有单词全部采用单数形式，例如：用Order而不是Orders。

## 字段(Field)的命名规范

1. 使用骆驼式命名法，第一个单词以小写字母开始，后面每一个单词的首字母都采用大写字母；
2. 如果字段的名称仅有一个单词，那么建议不使用缩写，而是用完整的单词；
3. 布尔型字段以一些助动词开头，例如：isChecked表示用户是否通过检查；
4. 数据库字段必须有明确的字段说明；
5. 主键以表名（实际名称）与Id组成，例如：orderId；
6. 外键与主键表的主键列名保持一致。

## 约束的命名规范

### 唯一索引命名

1. 唯一索引名由前缀、表名与字段名组成，以\_(下划线)衔接；
2. 唯一索引名全部为大写。
3. 前缀使用大写字母U，表示唯一；
4. 如果存在多字段唯一索引，取每字段前三个字符加下划线组合，例如： 在custom、cutting、curtail上建立唯一索引，命名为U\_表名\_CUS\_CUT\_CUR；
5. 如果前三个截取字符相同，就从字段名称中不同的字符开始取三个字符加下划线组合，如：在 custid、custom、custname上建立唯一索引，命名：U\_表名\_TID\_TOM\_TNA。

### 主键命名

1. 主键名由前缀、表名与字段名组成，以\_(下划线)衔接；
2. 主键名全部为大写。
3. 前缀使用大写字母P，表示主键；
4. 如果存在多字段主键，取每字段前三个字符加下划线组合，例如：在 custom、cutting、curtail上建立主键，命名为PK\_表名\_CUS\_CUT\_CUR；
5. 如果前三个截取字符相同，就从字段名称中不同的字符开始取三个字符加下划线组合，如在 custid、custom、custname上建立主键，命名：P\_表名\_TID\_TOM\_TNA。

### 外键命名

1. 外键名由前缀、表名、主表名与字段名组成，以\_(下划线)衔接；
2. 外键名全部为大写。
3. 前缀使用大写字母F，表示外键。

# 常用字典对照表

一个系统尽量采用统一单词。

|  |  |
| --- | --- |
| **名称** | **英文** |
| 资源 | ideaShop |
| 分类 | product |
| 属性 | prop |
| 参数 | param |
| 标签 | tag |
| 地区 | region |

# 数据类型

|  |  |  |
| --- | --- | --- |
| **字段类型** | **数据类型** | **长度** |
| ID | varchar | 36 |
| 时间 | date | 36 |

# 数据库原型设计

## 表的设计

### 主、外键

1. 每个表都必须要有主键。主键是每行数据的唯一标识，保证主键不可随意更新修改，在不知道是否需要主键的时候，请加上主键，它会为你的程序以及将来查找数据中的错误等等，提供一定的帮助；
2. 一个表的某列与另一表有关联关系的时候，如果加得上的话，请加上外键约束。
3. 适量建外键。为了保证外键的一致性，数据库会增加一些开销，如果有确凿的并且是对性能影响到无法满足用户需求的证据，可以考虑不建外键。否则，还是应该建外键；
4. 不要以数据操作不方便为理由而不建外键。是的，加上外键以后，一些数据操作变得有些麻烦，但是这正是对数据一致性的保护。正是因为这种保护很有效，所以最好不要拒绝它；
5. 以缺省的方式建立外键（即用delete restrict方式），以达到保护数据一致性的目的；外键在保护数据一致方面非常有效。如果不建外键，数据库中容易出现垃圾数据，并且无人知晓。当数据量很大的时候，查找这些垃圾数据也是相当困难的。而应用程序在设计时，往往没有考虑或者也无法照顾到垃圾数据。因此垃圾数据很可能造成应用程序工作不正常，并且表现出来的现象会很奇怪，让人摸不着头脑。

### 列的设计

1. 字段的宽度要在一定时间内足够用，但也不要过宽，占用过多的存储空间,对于长度不确定的列，采用可变长度的数据类型如 varchar类型；
2. 字段的类型及宽度在设计以及后面进行开发时，往往要与应用的设计、开发人员商讨，以得到双方认可的类型及宽度；
3. 除非必要，否则尽量不加冗余列。所谓冗余列，是指能通过其他列计算出来的列，或者是与某列表达同一含义的列，或者是从其他表复制过来的列等等。冗余列需要应用程序来维护一致性，相关列的值改变的时候，冗余列也需要随之修改，而这一规则未必所有人都知道，就有可能因此发生不一致的情况。如果是应用的特殊需要，或者是为了优化某些逻辑很复杂的查询等操作，可以加冗余列；
4. 除非必要，否则尽量不使用LONG, TEXT, BLOB, CLOB, NCLOB, LONG, LONG RAW这一类的数据类型，而是使用其他可以替代的数据类型；优先使用varchar2类型替代char类型，除非列宽有严格的要求而且得到应用严格支持；

### 记录数

1. 单表的记录数一般控制在两千万条 (参考值，各应用可以根据实际情况进行适量调整) 以内；
2. 记录数在两千万和两亿条之间的表一定要采用分区技术，并根据应用的使用情况创建合适的分区标准，单个分区内的记录数一般控制在两千万条(参考值，各应用可以根据实际情况进行适量调整)以内，同时表的索引使用对应的分区索引；
3. 记录数超过两亿条的表一定要考虑信息生命周期，必须考虑历史数据的剥离，并在应用设计中完成对历史数据的相应处理功能（历史数据的剥离规则须经业务使用部门的确认）。

## 索引的设计

1. 索引是从数据库中获取数据的最高效方式之一。95%的数据库性能问题都可以采用索引技术得到解决。但大量的DML操作会增加系统对索引的维护成本，对性能会有一定影响，对于插入相当频繁的表要慎重建索引，索引也会占相当的存储空间，所以要根据硬件环境和应用需求在空间和时间上达到最好的平衡点，主要原则：
2. 适当利用索引提高查询速度：当数据量比较大，了解应用程序的会有哪些查询，依据这些查询需求建相应的索引；最好亲自试验一下，模拟一下生产环境的数据量，在此数据量下，比较一下建索引前后的查询速度；索引对性能会有一定影响，对于DML频繁列的索引要定期维护（重建）。但是，索引的结构对于索引的更新（比如在插入数据的时候）是有一定优化的，所以不要在没有试验以前过分夸大它对性能的影响。最终还是以试验为准；
3. 不要建实际用不上的索引，与上条相关，如果建的索引并不提高任何一应用中的查询速度，则要把它删除；有些数据库有相关工具可以发现实际未被使用的索引，可以利用一下；
4. 索引类型的选择：要根据数据分布及应用来决定如何建立索引，一般的高基数数据列（高基数数据列是指该列有很多不同的值）时 ,建立BTree索引（一般数据库索引的缺省类型）；当低基数数据列（该列有大量相同的值）时，可以考虑建立位图索引（如果所选数据库支持的话），但位图索引是压缩类型索引，所以DML（增、删、改）的代价更高，要综合考虑；
5. 索引列的选择：如果检索条件有可能包含多列，创建联合主键或者联合索引，把最常用于检索条件的列放在最前端，其他的列排在后面；不要索引使用频繁的小型表，假如这些小表有频繁的DML就更不要建立索引，维护索引的代价远远高于扫描表的代价；
6. 主键索引在建立的时候一定要明确的指定名称，不能让系统默认建立主键索引（可能有些数据库无法指定主键名，则例外）；
7. 外键必须需建索引。当有一定数据量，并且经常以外键所在列为关联，进行关联查询时，需要建索引（可能有些数据库自动为外键建索引，则例外）；
8. 当有联合主键或者联合索引时，注意不要建重复的索引。举例说明：
9. 表EMPLOYEES，它的主键是建立在列DEPARTID和EMPLOYEEID上的联合主键，并且创建主键的语句中DEPARTID在前，EMPLOYEEID在后。在这样一个表里，通常就没有必要再为DEPARTID建一个索引了；联合索引的情况也一样；
10. 更复杂的情况，比如表EMPLOYEES，有一个索引建立在列CORPID, DEPARTID, EMPLOYEEID三列上，在创建语句中也依据上述顺序，就没有必要再为CORPID建立索引；也没有必要再建立以CORPID在前，DEPARTID在后的联合索引；如果EMPLOYEEID需要索引，那么为EMPLOYEEID建立一个索引是不与上面的索引重复的；DEPARTID列也类似；
11. 控制一个表的索引数量，尽量使得一个表的索引数量小于五个；

## 视图的设计

1. 在不太清楚视图用法的情况下，尽量不建。因为一旦建了，就有被滥用的危险；
2. 如果需要建视图，只要是打算长期使用的，请写入数据库设计中。明确它的用途、目的；
3. 建立视图时要明确写出所有要选择出的列名而不要以SELECT \*来代替，可以使结构清晰可读性增强，也不会增加它对表的所有字段的依赖，而表是很可能修改的，特别是增加字段。就很有可能导致使用该视图的应用程序出错。

## 存储过程、函数、触发器的设计

1. 触发器的功能通常可以用其他方式实现。在调试程序时触发器可能成为干扰。假如你确实需要采用触发器，一定要经过测试再应用在生产系统中，而且必须集中对它文档化。
2. 请把程序包、存储过程、函数、触发器，与应用程序一同加入CVS中，进行版本控制。因为此四者包含了代码，应用程序对他们的依赖程度比对表、视图的依赖程度更高；
3. 适量但尽量少使用存储过程、函数、触发器。使用存储过程、函数、触发器的影响：

(1) 可以减少数据库与客户端的交互，提高性能；

(2) 有的数据库还对他们进行了某种程度的编译，在执行的时候，不用再对其中的SQL等语句进行解析，从而提高速度；

(3) 如果有多个应用，使用了不同的开发语言，当有某些关键的或者复杂逻辑希望共享，则可以考虑使用存储过程或者函数。因为存储过程等在数据库一级是共享的；

(4) 增强了应用对数据库的依赖，如果打算将来移植数据库的话，使用得越多，则移植的困难越大；数据库中的业务逻辑越多（存储过程等），应用以及存储过程等的维护难度也会增大；

(5) 通常存储过程等没有面向对象的特性，不容易设计出易于扩展的结构。当存储过程比较复杂时，或者它们相互间的调用关系比较复杂时，可能难于维护。

## 数据计算

* 插值查询

提供插值查询能力，将未上传的数据补齐，并支持多种插值算法

* 聚合计算

提供AVG、SUM、MAX等15种聚合函数，可以将数据降精度聚合，并支持嵌套聚合

* 预处理

提供对数据的预处理能力，可将相关数据提前过滤和聚合，实现快速返回查询结果

## 压缩存储

采用高效压缩算法，可达10-20倍压缩率，大大降低存储成本。

## 数据库管理

实时监控,提供对数据库的写入、读取状态进行实时监控。

## 示范数据

提供示范数据，帮助用户体验时序数据库的各项功能。

* 高性能读写

每秒千万级数据点写入，亿级数据点聚合结果秒级返回

* 低成本存储

高效压缩算法，大大节省存储空间

* 强计算能力

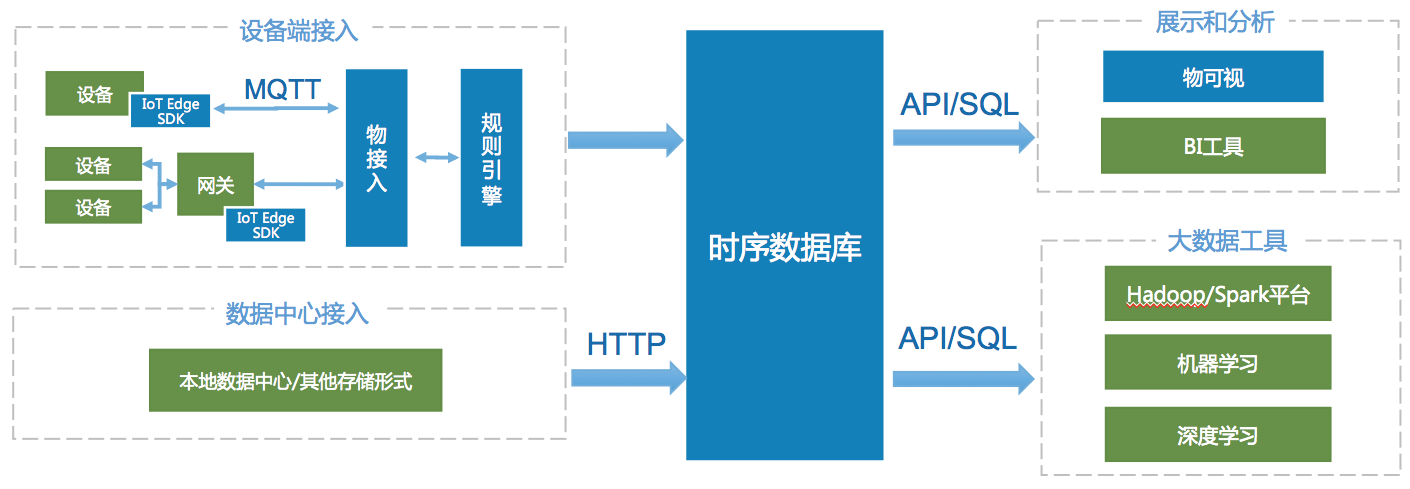
提供插值、预处理等多种计算方式；支持15种聚合函数

* 多生态支持

支持SQL查询、主流Hadoop/spark等大数据分析平台、多种可视化工具

* 高可靠服务

三副本、分布式部署，保证数据可靠性



**系统软件DATABASE ：**Time Series Database，时序数据库，用于保存时间序列（按时间顺序变化）的海量数据。

**度量（metric）：**数据指标的类别，如发动机的温度、发动机转速、模拟量等。

**域（field）：**在指定度量下数据的子类别。即一个metric支持多个field，如metric为wind，该metric可以有两个field：direction和speed。

**时间戳（timestamp）：**数据产生的时间点。

**数值（value）：**度量对应的数值，如56°C、1000r/s等（实际中不带单位）。如果有多个field，每个field都有相应的value。不同的field支持不同的数据类型写入。对于同一个field，如果写入了某个数据类型的value之后，相同的field不允许写入其他数据类型。

**标签（tag）：**一个标签是一个key-value对，用于提供额外的信息，如"设备号=95D8-7913"、“型号=ABC123”、“出厂编号=1234567890”等。

**数据点（data point）：**“1个metric+1个field（可选）+1个timestamp+1个value + n个tag（n>=1）”唯一定义了一个数据点。当写入的metric、field、timestamp、n个tag都相同时，后写入的value会覆盖先写入的value。