福富软件

数据库优化方法论

梁敬彬

福建富士通信息软件有限公司

Fujian Fujitsu Communication Software Co., Ltd.(FFCS)

DTCC2012

内容

DTCC2012

小余买鱼系列故事

小余买鱼1---诊断与改进

一天下午4点多，小余妈妈想做水煮活鱼给家人吃，让小余去买一条草鱼回来。

小余骑自行车到20里外的沃尔玛超市买到鱼然后返回。一到家，妈妈就开始责怪小余买鱼的时间花的太长了，因为都已经是下午6点半了，晚上7点一家人都安排好了外出的活动了，这下做水煮活鱼来不及了。。。。。

小余买鱼系列故事

原来问题出在这里：

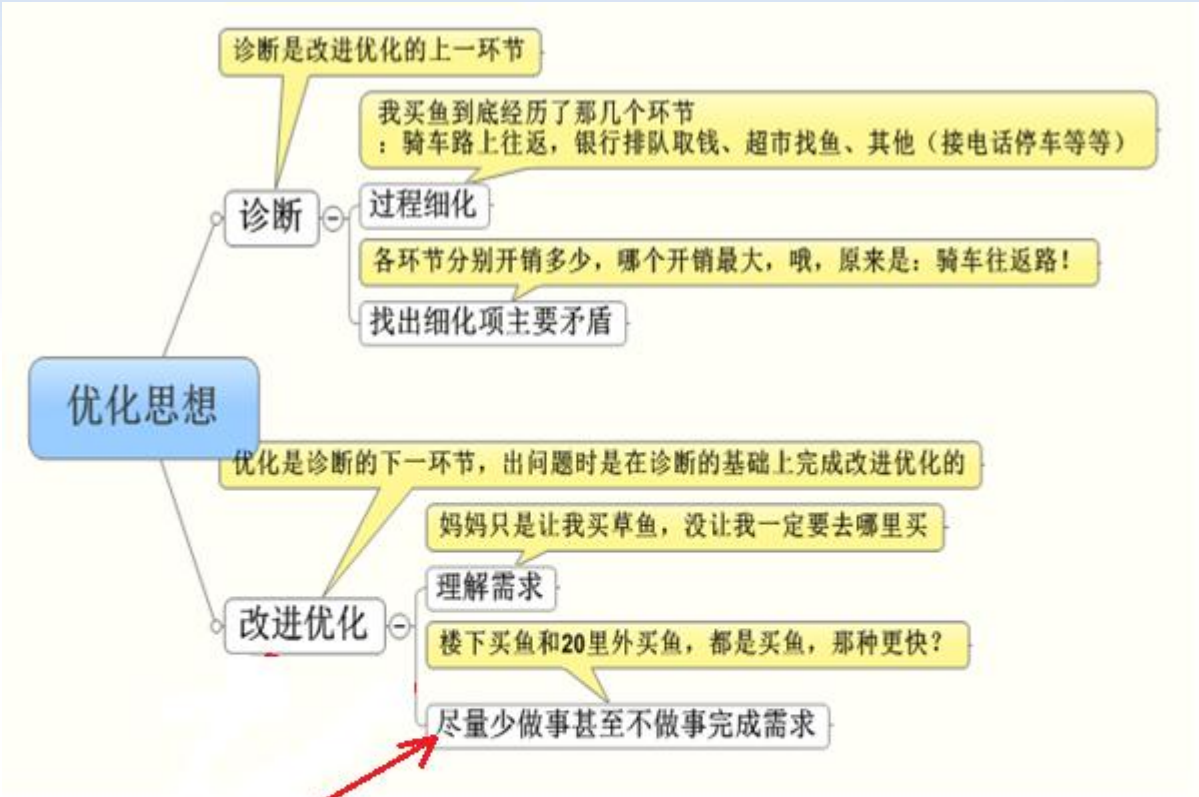
因为小余没来北京参加DTCC 2012数据库技术大会！

那参加后啥效果呢？

可获锦囊一袋，遇到危急时刻，可拆开。。。。。

DTCC2012

锦囊内藏如下内容：



随着功力的增长，后续这里会有变化哦。

小余买鱼2---需求与设计

一个月后，小余妈妈又准备开始做水煮活鱼了，妈妈还让小余去买一条草鱼回来。

不过这次情况发生变化了，家附近的农贸市场因故关闭了，由于住的比较偏僻，还真的只能去20里外沃尔玛超市买鱼了。

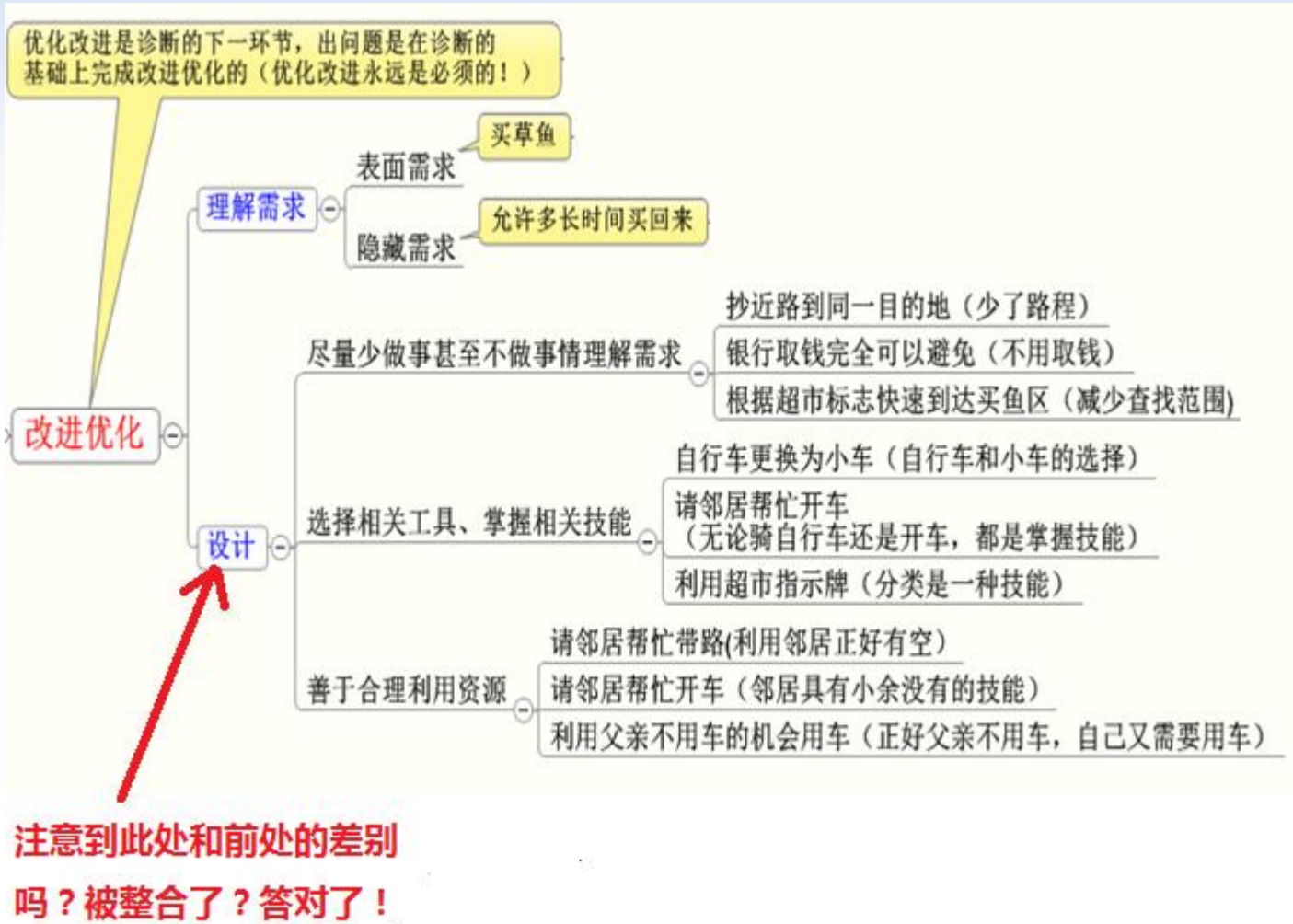
如果是以前,小余必然就是直接兴冲冲的一头冲出门，帮妈妈买鱼去。不过经历过第一次买鱼的经历后，他学会了思考，变得更成熟了。。。。。。（以下略去3000字。）

“妈妈，我回来了！”妈妈看到小余提着鱼，连连称赞，非常满意。

小余买鱼2---需求与设计

因为参加DTCC 2012 数据库技术大会特别认真，再获锦囊一袋！

新锦囊内藏如下内容：



注意到此处和前处的差别

吗？被整合了？答对了！

DTCC2012

小余买鱼3---资源的利用

又过了几天，妈妈再次让小余去买鱼。这次楼下附近的农贸市场开放了。

小余兴冲冲的让表哥帮忙一起开车去买鱼。

结果咋样呢？小余这次能否还能让妈妈满意呢？

小余买鱼3---资源的利用

答案揭晓，请看大屏幕

看样子是失败了，啥原因失败了？

因为参加DTCC 2012数据库技术大会走神了！

DTCC2012

小余买鱼3---资源的利用

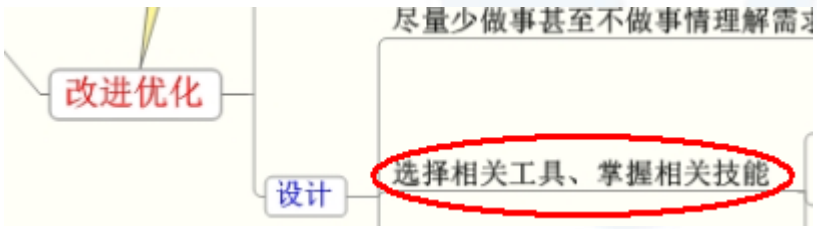
好了，不开玩笑了，真正原因应该如下：

因为农贸市场很近，走路仅需要5分钟可到达，妈妈的心理预期时间是在15分钟以内。

然而到地下车库开车、去农贸市场找地方停车，这里就花费了15分钟时间，超过了妈妈的预期时间。

妈妈当然不满意！

这就是要注意什么场景选择什么样的处理方式（从技术角度来看就是什么应用选择什么技术）。也就是对新锦囊妙计中设计的第2点的再次强调，这是非常重要的 。



小余买鱼3---资源的利用

事实上事情其实还更糟。

小余买鱼的这段时间爸爸正准备去公司参加紧急会议，结果车被开走了，最后导致会议迟到了。爸爸迟到这件事和上图设计中的第3点的相关：善于合理利用资源。

一来爸爸去出差了，二来买鱼的路途遥远，当然要合理利用资源。而情况变化后，就要及时考虑清楚了，车开走了，别人需要怎么办？

你事先沟通过了吗？你想过了吗？

小余买鱼4---真正的需求

又过了一个月，妈妈又准备让小余买草鱼来招待刚上门做客的大舅了。

不过因为离晚饭时间很近了，妈妈希望能在15分钟内买好鱼，而此时家附近的农贸市场依然没有开张，该怎么办呢？

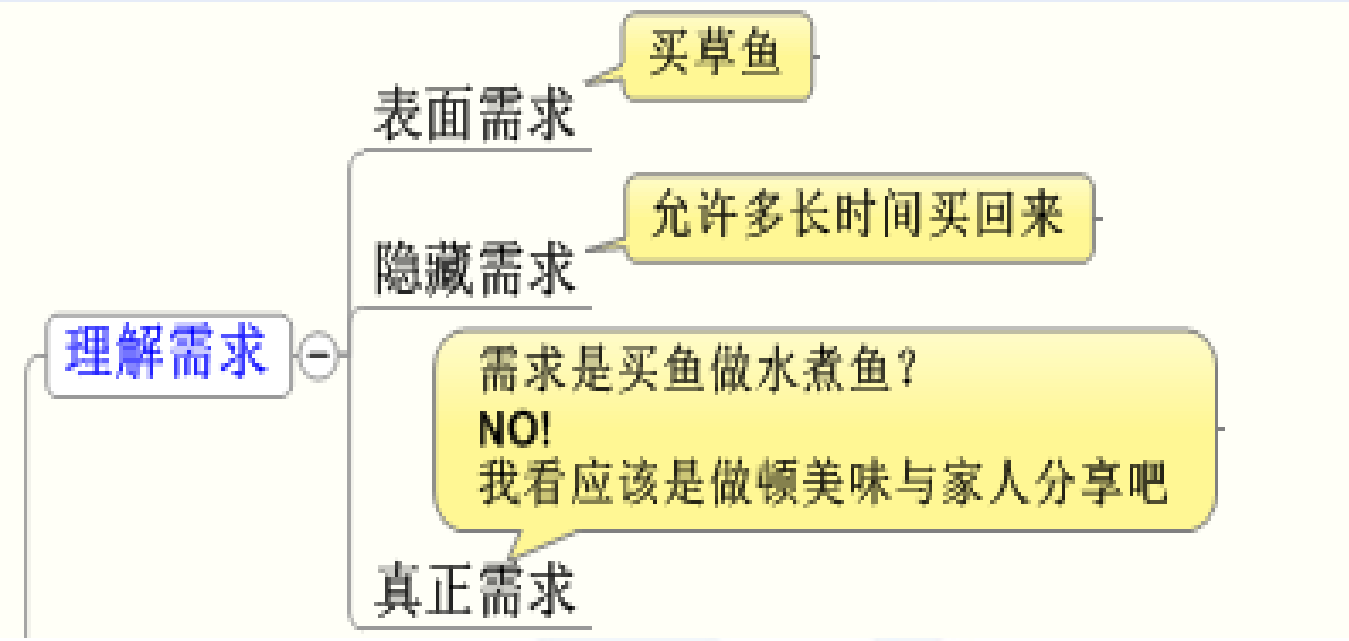
小余判断，无论如何都不可能完成这个任务了，不过小余还是开动了脑筋。最终居然让妈妈满意的点点头。你们谁能猜到小余做了什么事吗？

小余买鱼4---真正的需求

我估计谁也猜不到这次小余怎么让妈妈满意了，让我来公布答案吧。

答案就是：

最终小余让妈妈别买鱼了，用冰箱里的牛肉做水煮肉片。



内容

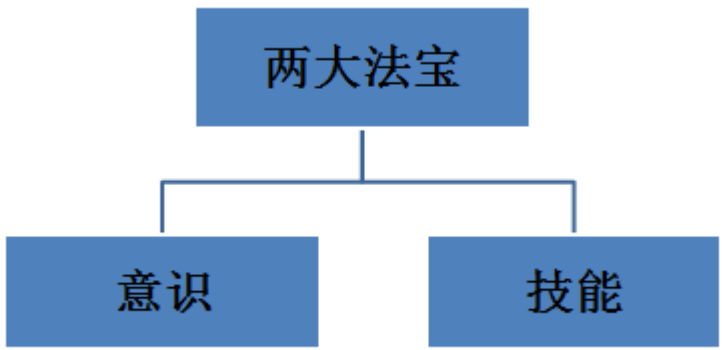
1. 小余买鱼系列故事
2. ***买鱼买出方法论***
3. 方法论应用案例
4. 总结

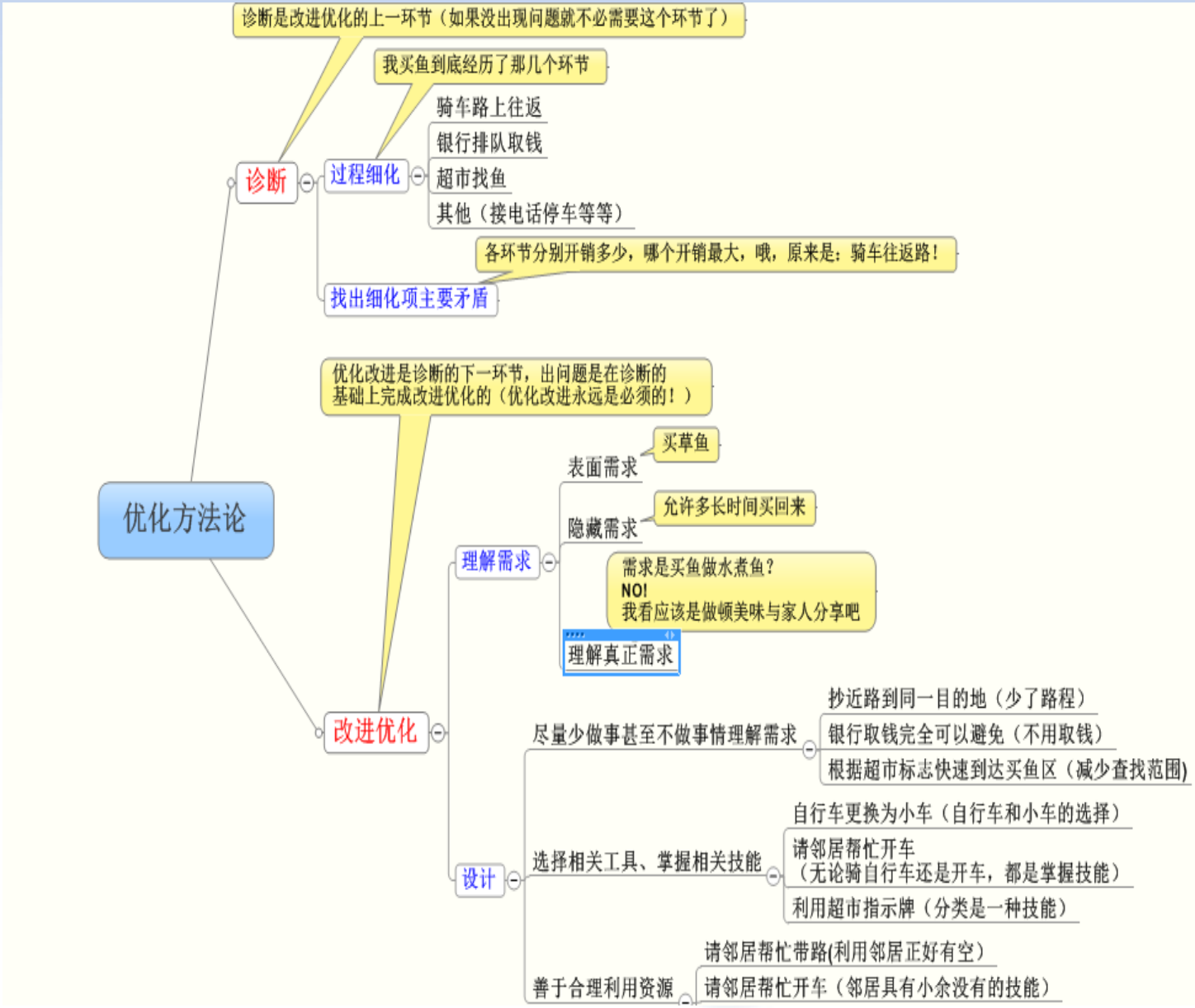
买鱼买出方法论

1. 一套流程

优化方法论，详见下页

2. 两大法宝





两大法宝：意识和技能

小余思考自己买鱼具体经历了哪些环节；揣摩最长时间耗在哪里；了解楼下是否有鱼；判断银行排队取钱能否避免；探索要去某地买鱼是否有近路。。。。。

上述部分和专业技术能力无关，我们将其归纳为意识类。

小余买鱼过程中曾经骑自行车去、也曾开车去。

然而无论是开车还是骑自行车，都是必须训练学习方可掌握，我们将其归纳为技能类。

两大法宝：意识和技能

生活中的优化就是意识和技能的结合，两者都非常重要。

首先说技能，掌握技能的重要性毋庸置疑，比如要到很远的沃尔玛买鱼，你既无开车的技能也没骑车的本领，那只能靠走路到达目的地，那估计到达时店铺也打烊了。

接下来谈意识，生活中有不少场景甚至是仅靠意识而未使用特定技能最终解决问题的。比如小余直接去楼下买到鱼了，还需要考虑会不会骑车吗？再比如小余让妈妈改做水煮牛肉了，还需要掌握开车的本领吗？

内容

1. 小余买鱼系列故事
2. 买鱼买出方法论
3. ***方法论应用案例***
4. 总结

方法论应用案例

案例1

某电信营运商生产系统出现故障，短信平台产生大量积压，出现不少客户投诉。

维护人员跟踪发现，原因是后台短信平台进程调用数据库中某个过程包，该过程包原先执行返回结果给后台仅需10秒不到，现在不知是何种原因过程包返回时间居然长达1分钟。所以导致短信后台程序处理缓慢许多，最终造成短信积压。

情况紧急，需要立即着手调查，该怎么处理呢？

诊断

- 用10046 trace 工具开始跟踪

```
alter session set events '10046 trace name context  forever,level 12';
```

- 执行你的数据库包

```
exec pkg_test('abc');
```

- 执行包完毕后结束跟踪

```
alter session set events '10046 trace name context off';
```

- 10046 trace 工具跟踪完毕后会输出分析结果，类似如下：

```
E:\admin\ora10\udump\ora10_ora_4832.trc
```

- 可格式化后进行分析，类似如下：

```
tkprof  E:\admin\ora10\udump\ora10_ora_4832.trc  d:\10046.txt  sys=no
sort=prsela,exeela,fchela
```

- 然后分析10046.txt的文件，这里响应时间从大到小展现该包所有SQL语句，即可有如下收获

1. 该过程包总共执行了多少SQL语句，具体内容是什么，分别开销了多少时长
2. 哪些是开销时长最长的语句。（由于有排序过，所以最长的一眼可看出，在最前端）

咦，这和《小余买鱼1》中的诊断方法有差别吗？

还真是过程细化和找细化项主要矛盾两个动作啊。

接着分析10046.txt文件，发现原来慢的SQL是类似如下2条简单的SQL，分别占用30秒和20秒，其他所有SQL单次执行都只零点几秒.

语句1（SQL1耗时30秒）

```
Select count(*) from t1;
```

语句2（SQL2耗时20秒）

```
Select  distinct t1.col1,t1.col2,t2.col3,t2.col4
from t1 ,t2
where t1.id=t2.id and t1.name='cc'
order by t1.col5;
```

其他SQL语句（合计才消耗0.5秒）

```
----SQL3   (0.03秒)
----SQL4   (0.028秒)
-- --SQLn…….(0.001秒)
--略去
```

改进优化（首次优化）

该表记录目前有5千万，每次都对全表进行扫描仅为了获取该表的记录。我们的需求是为了得到记录数，是否一定需要对全表进行扫描呢？

就好比小余要去买鱼，是否一定要去20里外的沃尔玛超市呢？

只有如果对该表存放序列值的非空字段SEQ_ID上建一个索引，全扫描该索引，一样可以获取到该表有多少记录的信息。看来，看来获取表记录数可以有两种方法了。选那种方法呢？

改进优化（首次优化）

大家知道，索引的大小比表的大小小的多，在更大的范围内遍历更快速还是在小的多的范围内遍历更快速呢？

好比买鱼是到远方的沃尔玛还是楼下的农贸商场买鱼那种更快一样，答案毋庸置疑。

T1表的SEQ_ID这个非空字段上建立一个索引该SQL执行速度从原来的30秒变为1秒左右。

改进优化（首次优化）

接下来进行SQL2的调优，和优化SQL1时一样首先开始查看分析SQL2语句的执行计划，发现SQL2的执行计划也是全表扫描。

这里t1.name=的取值为cc的返回仅仅10条记录，而T1表记录都在5千万左右，T2表在200万左右，需要全扫描这么大的两个表而获取仅有的10记录吗？

```
Select  distinct t1.col1,t1.col2,t2.col3,t2.col4
from t1 ,t2
where t1.id=t2.id and t1.name='cc'
order by t1.col5;
```

改进优化（首次优化）

这里又要再次利用到索引的原理，SQL1是利用到了索引一般比表小的多的特点，现在又是要利用啥呢？

哦，利用索引的快速定位原理。假如我们在name列建了一个索引，而现在是利用了索引的快速检索原理。

索引有个最大的特点是有序排列，当表记录检索到dc等以d打头的记录后，ORACLE就停止遍历了！

为啥，因为索引是有序的，当出现d打头的记录后，绝对后面不可能再出现c打头的记录了，因为我们是查询=cc的值，当然停住了。

随时停止检索相比遍历全表，明显是少做事和不做事，效率可以意料会提升不少。

需求与设计（再次优化1）

遇到疑问最重要要去如何分析？

首先应该想法去理解需求！

分析过程果然有新发现！原来该SQL1在过程包中类似如下：

```
begin
select count(*) into v_cnt from t1 ;
if v_cnt>0
then  …A逻辑….
else
then  …B逻辑…..
End;
```

我来翻译一下这段需求：

获取t1 表的记录数，判断是否大于0，如果大于0走A逻辑，否则就走B逻辑。

因此代码就如上所示来实现了。真正的需求是这样吗？

其实应该是这样的：只要T1表有记录就走A逻辑，否则走B逻辑。

需求与设计（再次优化1）

两者有区别吗？其实区别还是很大的，前者可是强调获取记录数，我们是不是一定要遍历整个表得出一个记录数才知道是否大于0？

真正需求的理解可以让我们这样实现，只要从T1表中成功获取到第一条记录，就可以停止检索了，表示该表有记录了，难道事实不是这样？

因此原先的SQL1 从Select count(*) from t1; 被改造为：

Select count(*) from t1 where rownum=1;

速度从1秒提升到0.01秒，几乎可以忽略不计了。

这里我们马上联想到《小余买鱼4》，妈妈真正的需求其实是要做美味的晚餐，站在这个角度，完全可以用现成的其他菜来代替非常麻烦的买鱼经历，少做事甚至不做事而快速满足需求，提升效率。

需求与设计（再次优化2）

对于SQL2,我查看了T2表，发现真有大量重复记录，怪不得需要用distinct来排重。

我很奇怪为什么会有重复记录，在问明开发设计人员后，我明白了，原来是源头程序有漏洞，导致t2表出现大量重复记录，现在所有应用只要涉及到访问t2表的，都需要增加distinct关键字来排重！

天啦，居然还有这回事！还有更神奇的，关于此处的order by，居然是多余的，展现的几个字段的输出根本无需排序，没这需求！

接下来思路很简单，即优化了源头程序（另外一个数据库包），保证插入t2表的数据再不会有重复记录，然后再把t2表记录进行删除重复记录的操作。

资源利用（花絮）

读者还记得我说的**T2**表排除重复记录的事情吧，我当时提供了技术方案给维护人员，方案是新建一张表出来，提取不重复记录，然后把旧表替换了，大致思路如下：

1. 首先建立新表

Create table t2_new nologging parallel 12
As select distinct * from t2 where …..;

1. 停应用
2. Rename t2 to t2_bk
3. Rename t2_new to t2;
4. 补上t2表的相关索引，并将t2表的logging属性恢复。

资源利用（花絮）

我之前有提醒是要在业务不繁忙的时候操作，比如凌晨打补丁的时候顺道操作。因为我有写了**parallel 12**，表示要并行用到**12**个**CPU**，（当时生产仅有**12**个**CPU**）。

可惜的是维护人员自作主张，在大白天系统繁忙的时候执行了上述语句，**结果导致12个CPU资源被这条create table t2_new的语句占据消耗着，引发生产系统短时间的压力繁忙，险些压垮了系统**，好在该语句在**5**分钟内结束，未造成严重的影响。

资源利用（花絮）

亲爱的听众，这个花絮让我们联想到小余买鱼故事中的哪个系列？

应该是《小余买鱼3》吧。

恭喜你，答对了！

让我们一起回想一下小余买鱼3的故事：

小余不会合理利用资源，开车去楼下附近农贸市场买鱼，导致要去远方开会的爸爸没车开，迟到了。

当然《小余买鱼2》却是会合理利用资源的典范，爸爸去出差了，要去那么远的地方，不选择开车去就是傻瓜了。

如果维护人员按我的要求在凌晨打补丁的时候顺便执行这个语句，不用并行也是不善于利用资源，因为凌晨系统没有什么进程在运行，不会有应用因为CPU被占用光而受到影响。

案例2

技术人员反映，某平台运行一天比一天缓慢，难以容忍，提出需要优化。

结果发现是CRONTAB自动调用的定时脚本运行非常缓慢，由于10分钟一个定时任务，而自动脚本执行事件却超过10分钟，导致所有的CPU都被该自动脚本耗光。

剖析该定时脚本，发现主要是因为调用其中的存储过程执行比较缓慢，慢的原因涉及到一个大表的全表扫描，但是只返回少数的记录，所以在合适的列加索引，就可以优化。

那你猜我是如何优化呢？

案例2

是加索引吧。

NO!

其实我是把定时脚本给剁了

剁了？

是的，因为我发现了这样的语句 **where deal_time >200301 and deal_time<=200304**

啥，我听不懂哦？

两大法宝

还记得前面《小余买鱼》故事总结的买鱼方法论中的“两大法宝”吧，总结的很简单，就是意识+技能。

不知道读者注意到没有，这次生产系统的数据库优化不断体现出这两大法宝的作用。

两大法宝

意识表现为：

1. 思考该数据库包具体涉及那几个模块，主要矛盾又在哪个模块？（选择用10046 TRACE 工具包来实现）；

2. 找到问题所在以后去理解需求，探索是否能少做事完成需求（选择用索引来替代全表扫描，从而减少访问路径）；

3. 去思考需求背后的真正需求（最终将select count(*) from t1 改造为select count(*) from t1 where rownum=1）；

4. 去分析资源如何合理应用 （在系统繁忙时使用并行，占用他人资源，险些酿成大祸）。

为什么称之为意识，因为这些并不是具体的知识点技能的使用，更像是一种经验的总结，习惯的培养，所以称之为意识。

技能表现为：

其实上述的意识正是和技能紧密联系的。比如 10046 TRACE诊断工具包的使用和学习；比如执行计划的查看与分析；比如理解索引的原理；比如知道如何使用并行的命令。。。。

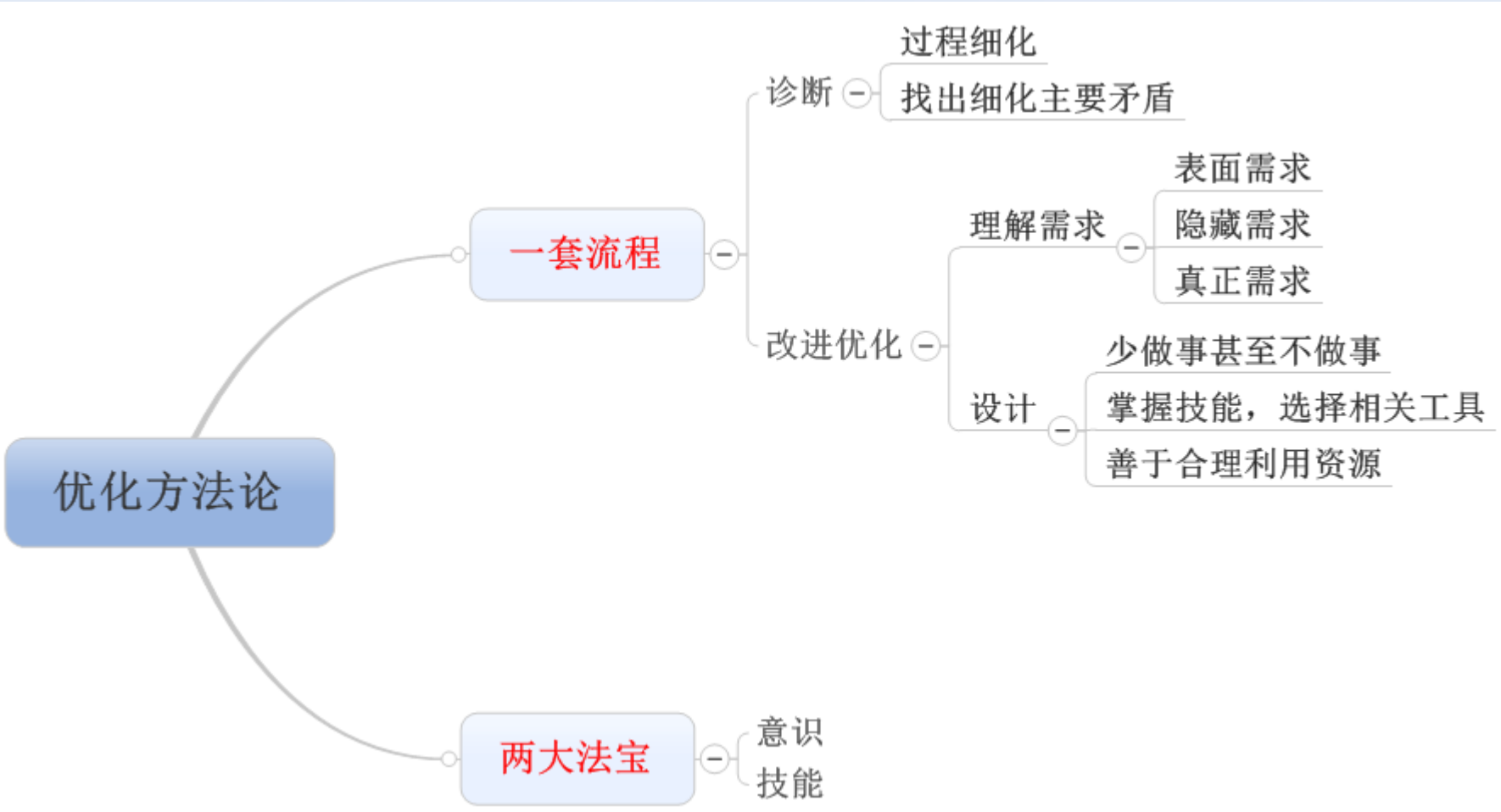
ORACLE优化的意识的内容大致就在总结的一套流程中，看起来内容不多，不过这些朴素的道理确是优化的核心思想，非常经典！

相比ORACLE优化意识而言，ORACLE技能涉及的范围却有不少，比如除了前面介绍的技能的一小部分外，还有理解ORACLE体系结构；理解表连接原理；理解AWR等性能报表的使用与分析；精通SQL和PL/SQL编程等等。。。。。

内容

1. 小余买鱼系列故事
2. 买鱼买出方法论
3. 方法论应用案例
4. ***总结***

总结就是：看图说话



诚谢各位领导、专家的关注和聆听！