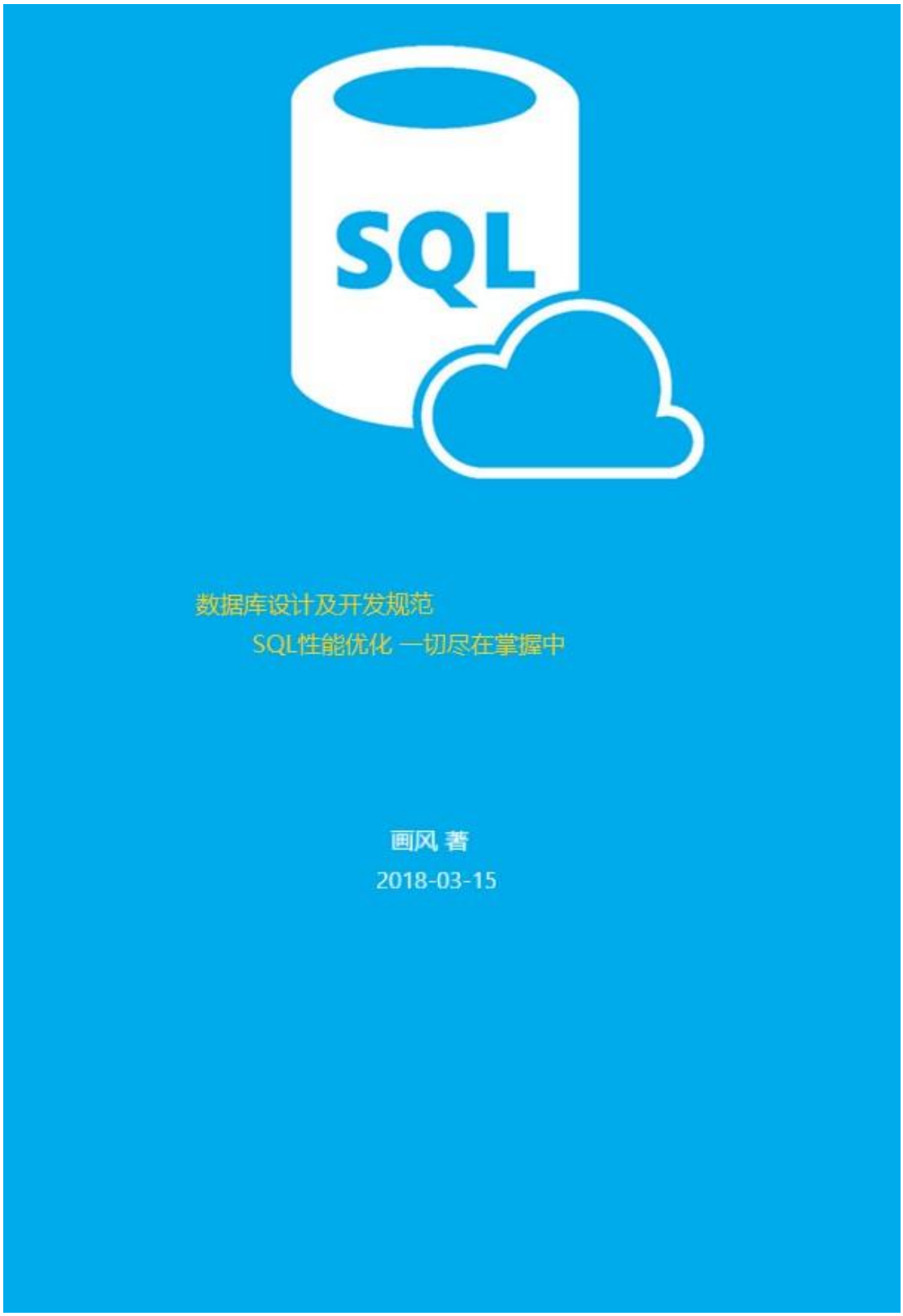
数据库设计及开发规范



目录

1 引言

1.1 编写目的

本文档用于规范数据库设计、开发等方面的内容。

1.2 预期读者

本文档的预期读者为本项目组全体成员，以及其他与项目有关的管理人员。

1.3 术语定义

OMP：Operation Management Platform

1.4 参考文档

《数据库编码规范.pdf》

2 数据库规范

2.1 设计规范

2.1.1 命名规范

数据库对象的命名规则的范围为管理平台设计开发所涉及的表，对于其他外部系统所创建的表不在本规范约束范围内， 数据库对象如表、列、序列、过程、函数等在命名时要遵循如下规则：

- 命名要使用富有意义中文名称缩写，要以字母开头，不能超过 30 个字符。
- 数据库对象名称由如下部分组成：范围、类型、名称实体，各词汇间采用“_”连接。

- 其中各数据库对象的范围和类型的具体含义及取值详见各数据库对象的命名规则。
- 数据库对象的名称不允许是数据库的 **保留字** 和 **关键字**。

数据库对象	格式	样例	说明
表			
订单表	<范围(可缩写)>_<类型(t OR v OR mv)>_<表名(可缩写)>	pro_t_orders	其中 pro 代表所属模块为产品，t 为表单类型,orders 为表单名称
索引			
普通索引	IDX_<表名>_N<序号>	IDX_PRO_T_ORDERS_N1	PRO_T_ORDERS表第一个普通索引
位图索引	IDX_<表名>_B<序号>	IDX_PRO_T_ORDERS_B1	PRO_T_ORDERS表第一个位图索引
唯一索引	IDX_<表名>_U<序号>	IDX_PRO_T_ORDERS_U1	PRO_T_ORDERS表第一个唯一索引
视图			
订单普通视图表	<范围(可缩写)>_<类型(t OR v OR mv)>_<表名(可缩写)>	pro_v_orders	其中 pro 代表所属模块为产品，t 为表单view类型,orders 为表单名称
其他对象			
存储过程	P_<存储过程名称>	P_GET_SYSDATE	获取系统时间
函数	F_<函数名称>	F_GET_ORDER_TYPE_COMMENT	根据编码转换字典表对应订单类型中文名称
包	PKG_<包名称>	PKG_PRINT	打印
主键	PK_<表名>	PK_PRO_T_ORDERS	
外键	FK_<表名>_<序号>	FK_PRO_T_ORDERS_1	
序列	SEQ_<序列名称>	SEQ_REGISTER_NUMBER	受理编号序列
别名信息类型定义	<用户名>_<表名>		

2.1.2 表规范

2.1.2.1 建表的参数设计

- 不允许将表建立在默认系统表空间上
- 表和索引建立在不同表空间上
- 建表时必须指明所存储的表空间
- 生成表脚本时非空的列放在表的前部，可空的列放在表的后部

- 数据缓冲池的类型：查询频繁且数据量较少的参数表 采用 buffer pool keep
- INITIAL: 对初始化数据量大的表，设置的值要大于初始化数据
- PARALLEL: 对于 OLTP(联机事务处理)系统，不允许使用该参数

2.1.2.2 主外键设计

- 数据约束优先考虑利用数据库提供的约束机制，在数据库产品所提供的机制无法满足的情况下，再考虑通过编程实现
- 主键的设置通常不使用实际意义的列做主键，具体情况应结合业务特性综合考虑
- 字表在外键的字段上必须建立索引
- 由 Sequence 产生的 ID 列，不作为组合 PK 的列
- 删除约束时使用 keep index 参数

2.1.2.3 列设计

- 定长字符类型列使用 CHAR 类型，最大长度为 2000；不定长字符类型列使用 VARCHAR 类型，最大长度为 4000
- 日期字段需定义为 DATE 类型。如果定义为 VARCHAR 或者 CHAR 时需要进行转换，影响效率。需要数据精确到微秒的字段定义为 TIMESTAMP 类型
- 列表为 null 时，需要定义 default 值，避免因为 null 而造成索引不能被用到的情况
- 使用 NUMBER 类型是必须指定长度。由 NUMBER 的到精度与密度来保障数据的一致性
- 表中字段的命名长度不应该超过 30 个字节

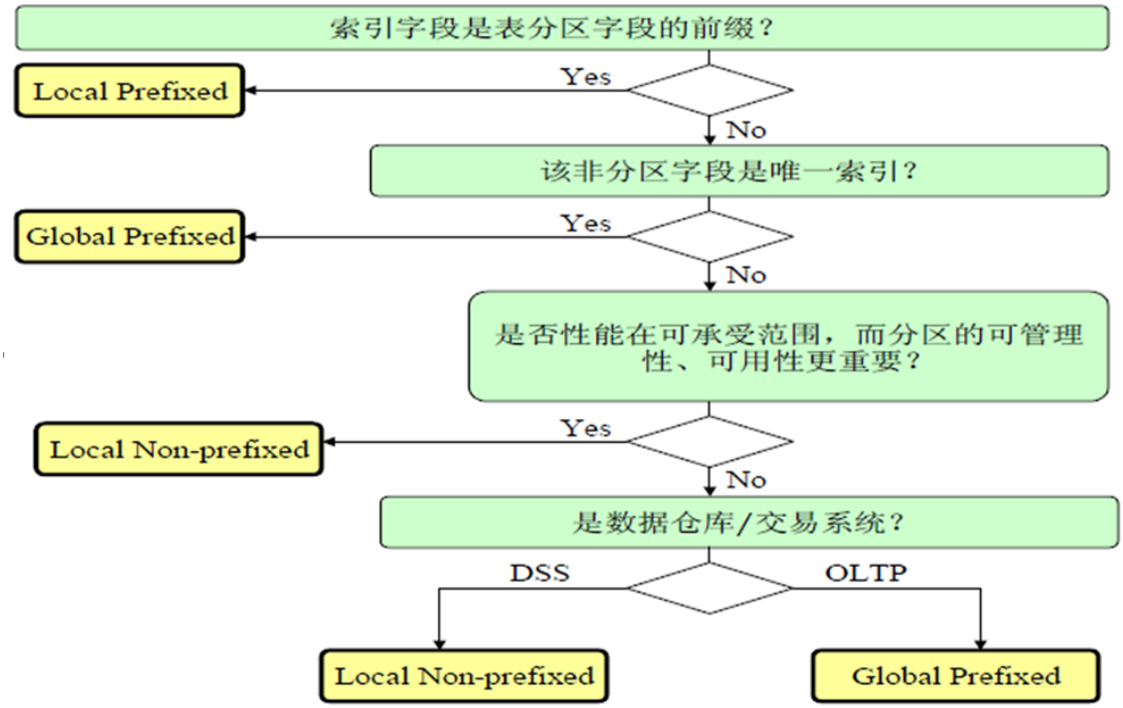
- 记录数达到千万级的表，必须进行分区，分区一般遵循以下原则：
    i. 数据具有明显的范围属性，比如日期，大小等，且经常进行范围条件查询的表，采用范围分区。
    ii. 数据具有明显的列表属性，比如地点，省份等，且经常用列表条件查询的表，采用列表分区。
    iii. 数据不具有明显的范围属性或者列表属性，且数据量很大，则可以采用 hash 分区。

2.1.2.4 临时表

- 对于只对本事务有效的临时表使用 ON COMMIT DELETE ROWS 关键字创建该表
- 对于只对本会话有效的临时表使用 ON COMMIT PRESERVE ROWS 关键字建该表
- 对于临时表空间要求比较大的业务系统，临时表要存储在独立的表空间中，并且临时表空间的数据文件需要放在独立的磁盘上

2.1.3 索引规范

- 选择使用普通 B 树索引
- 小表(数据量小于 10000 条记录为标准)不需要建立索引
- 创建或重建索引时需指定使用 NOLOGGING 子句，提高执行效率 对于分区索引，建全局分区或者本地分区规则如下：



- 建立分区索引必须指定表空间，并且指定的表空间要与数据表空间分开。
- 对于 OLTP 应用的业务系统，单个表上索引的个数不超过 5 个
- 将记录差别数最多的列放在索引顺序的最前面
- 对于 OLTP 应用的业务系统索引数据的重复率尽量不超过 20%
- 进行 order by column desc 排序时，创建 column desc 索引
- 频繁使用的 index 需要放入库缓存的 keep 池中

2.1.4 存储过程、函数、包规范

- 存储过程、函数和包中不允许频繁使用 DDL 语句
- 存储过程、函数和包中必须有相应的出错处理功能
- 存储过程、函数和包中变量在引用表字段的时候，使用%rowtype 类型

2.1.5 别名

- 对于只读用户，必须创建与表相同名字的别名
- 别名的访问顺序：public 别名 -> private 别名 -> 与表同名的对象

2.1.6 Database Link 别名

- 只允许从其它数据库中查询少量数据时使用 dblink
- 不使用 dblink 更新其它数据库的数据

开发规范

2.2.1 变量命名规范

变量的命名体现其作用域和数据类型，规则如下：

- <变量作用域>_<有意义的变量名字>_<变量类型>_<后缀>
- 变量名不能超过数据库限制(30 个字符)
- 供别的文件或函数调用的函数，不能使用全局变量交换数据

项目	格式	说明
全局变量	Gv_变量名称_变量类型	
局部变量	Lv_变量名称_变量类型	
常量	C_常量名称_常量类型	
光标	Gr_光标名称_光标类型	
参数 1	P_参数名称_参数类型	一般参数
参数 2	P_column_name	与数据库表中列对应的参数和变量
数组	数组名称_ary	
记录	记录名称_rec	
COLLECTION	名称_coll	

2.2.2 SQL 开发规范

2.2.2.1 SQL 书写规范

- 每行不能写超过 80 个字符
- 使用两个空格缩进代码，比如：

```
BEGIN
  FOR l_count IN 1..10 LOOP
    x_result := x_result + l_count;
  END LOOP;
END;
```

- 关键词要大写(比如 INSERT)
- 常数符号要大写,比如：

```
CONSTRAINT G_MAX_VALUE NUMBER :=10;
...
  IF(1_value > G_MAX_VALUE)
  THEN
    ...
```

- 语句中出现的所有表名、列名全部小写，系统保留字、内置函数名、SQL 保留字大写，连接符 OR、IN、AND、以及=、<=、>=等前后加上一个空格
- SQL 语句的缩进风格：

  一行有多列，基于列对齐原则，采用下行缩进 WHERE 子句书写时，每个条件占一行，语句另起一行时，以保留字或者连接符开始，连接符右对齐
- SELECT 语句中不可以用 *,必须 SELECT 字段列表，以节省内存，提高效率

- 避免频繁 COMMIT ,尤其是把 COMMIT 写在循环体中每次循环都进行 COMMIT 。避免在一个事务中出现 2 次 COMMIT 现象。例如执行一半执行一次 COMMIT，执行完另外一半又执行一次 COMMIT。
- LIKE 子句如果非必要时尽量使用前端匹配，如写成 LIKE'STRING%'，不要写成 LIKE'%STRING%'
- 批量 INSERT 大量数据时可以采用 APPEND 和 NOLOGGING 方式，提高处理速度
- EXP 时可以采用 DIRECT=Y,INDEX=NO 的方式提高处理效率，IMP 时可以指定较大的 buffer。如果是 ORACLE 10g 以上则可以使用 EXPDP 和 IMPDP 来提高处理的速度
- EXP 和 IMP 时，如果系统中存在主外键约束，在 IMP 时可以设置 CONSTRAINTS=N，避免在 IMP 时因为主外键而报错
- SQL 中尽量不要使用数据库未文档化的功能，比如 WMSYS.WM_CONCAT

2.2.2.2 索引与分区使用规范

- 表的记录数少于 10000 条，执行全表扫描
- 在写查询条件时注意引用索引
- 批量提取数据，使用按分区扫描
- 比较值与索引列数据类型要保持一致性
- 查询列与索引列次序保持一致
- 拍序列与索引列次序保持一致
- 对于复合索引，SQL 语句必须使用主索引列，按照复合索引组成列的顺序书写

- 尽量不要对索引列进行计算，如有特例对索引列计算较多，则需要建立函数索引
- IN、OP 子句常会使索引失效，在表数据记录数大于 10000 条的情况下，考虑把子句拆开
- 对于索引的比较，不使用 NOT
- 删除一个表的所有数据时，使用 TRUNCATE，而不是 DELETE。不能在事务中使用该语句，并且务必确认表中数据可以全部被删除

2.2.2.3 SELECT 列和 WHERE 条件规范

- 在查询语句中查询表达式左边不允许出现函数及其它运算表达式，所有左边的表达式都可以用其它的方法实现
- WHERE 条件中不要使用常量比较，将常量绑定到变量中使用
- 查找数据时只取出确实需要的那些列，不要使用 * 来代替所有列名。要清楚明白地使用列名，而不能使用列的序号
- 不要 order by 和 group by 排序操作。必须使用排序操作时，请遵循如下规则：

  排序尽量建立在有索引的列上 查询结果集不要求唯一时，使用 union all 代替 union

2.2.2.4 多表连接规范

- 在表中的记录数低于 10 万条的情况下，可以使用多表连接
- 多表连接时，必须使用表的别名来引用列
- 使用 EXISTS 替代 DISTINCT 表达方式

```
#低效:
SELECT DISTINCT DEPT_NO,DEPT_NAME
    FROM DEPT D,EMP E
    WHERE D.DEPT_NO = E.DEPT_NO
```

```
#高效:
SELECT DEPT_NO,DEPT_NAME
    FROM DEPT D
    WHERE EXISTS ( SELECT 'X'
                   FROM EMP E
                   WHERE E.DEPT_NO = D.DEPT_NO);
```

- 多张大表进行 JOIN 时一条 SQL 语句中关联查询的大表尽量不要超过 3 个

2.2.2.5 嵌套查询规范

- 使用 NOT EXIST 代替 NOT IN 子句进行嵌套查询

```
#例如:
SELECT …
FROM EMP
WHERE DEPT_NO NOT IN (SELECT DEPT_NO
                      FROM DEPT
                      WHERE DEPT_CAT='A');
#为了提高效率,改写成:
SELECT ….
FROM EMP E
WHERE NOT EXISTS (SELECT 'X'
                  FROM DEPT D
                  WHERE D.DEPT_NO = E.DEPT_NO
                  AND DEPT_CAT = 'A');
```

- 避免嵌套连接，子查询(多级) 如：A = B AND B = C AND C = D

2.2.3 PL/SQL 开发规范

2.2.3.1 包规范

- 按照项目制定的文件组织划分包内容

2.2.3.2 游标规范

- 外部查询的多行数据返回使用游标进行处理，通过传递游标变量的形式返回数据到外部接口，由外部程序自行 FETCH 数据
- 打开游标前，必须显式检查游标的%ISOPEN 属性
- 使用 FETCH 语句后，要立即检查%NOTFOUND 属性，以便正常终止游标 FETCH 循环
- 无论 PL/SQL 程序是正常终止还是出错退出，都要关闭所有已打开的游标。在出错退出时，应该在其异常处理部分所有游标，这可以释放一部分的系统资源
- 尽可能使用显式游标，避免使用隐式游标

2.2.3.3 事务处理规范

- 在需要分割事务以使主事务的提交或者回滚独立于子事务的提交及回滚时，应使用自治事务
- 所有的存储过程均统一在结束处 COMMIT 或者 ROLLBACK

2.2.3.4 数据封装规范

- 按照业务逻辑实现功能模块的封装，将业务逻辑集中的在更少量的、良好设计的、易于维护的函数或者过程，不必每条 SQL 语句或者每天 PL/SQL 程序中重复这些逻辑
- 基于单一数据表的增、删、改、查采用标准 SP 进行封装，不允许同逻辑的处理出现在多个 SP 中

2.2.3.5 数据访问规范

- 后台数据按照逻辑划分成多个 SCHEMA，不同 SCHEMA 的数据不可互相访问

- 需要相互访问的表均存放在某一个的 SCHEMA 中，通过访问该 SCHEMA 中的接口表实现跨 SCHEMA 的数据访问

2.2.3.6 日志书写规范

- 采用公共的 API 包完成后台日志数据记录(API 完成输出错误信息提示、记录错误信息内容到数据库表、系统级的错误代码及错误信息等)
- 后台日志的信息记录级别包括 INFO、WARN、ERROR，其定义以及不同级别日志的采集标准如下：

  INFO-提示信息，供开发人员调试使用，由开发人员自行确定，主要是调试信息，程序运行中普通信息提示 WARN-警告信息,可能导致严重错误的警告信息 ERROR-错误信息，导致系统运行错误的信息
- 所有表操作的错误处理部分均应记录日志信息

2.2.3.7 错误处理规范

- 凡是涉及到表操作(INSERT,UPDATE,SELECT,DELETE)的 SQL 语句，都必须进行错误捕捉，不能将错误带到后面的语句
- 错误信息必须准确
- 在每个异常错误处理部分，捕捉到的错误要写入错误日志系统
- 写入错误日志函数统一提供

2.2.3.8 书写规范

- PL/SQL 语句的所有表名、字段名遵照数据字典的定义，系统保留字、内置函数名、PL/SQL 保留字\关键字大写，用户声明的标识符小写
- 对于子程序、触发器、包等带名的程序块，使用结束标识

- 连接符 OR、IN、AND、以及=、<=、>=等前后加上一个空格
- 对较为复杂的 SQL 语句加上注释，说明算法、功能
- 注释风格：注释单独成行、放在语句前面

  应对不易理解的分支条件表达式加注释 对重要的计算应说明其功能 过长的行数出现，应将其语句按实现的功能分段加以概括性说明 常量及变量注释时，应注释保存值的含义(必须),合法取值的范围 可采用单行/多行注释。(-- 或 /* */ 或者 #)

- SQL 语句的缩进风格

  一行有多列，超过 80 个字节时，基于列对齐原则，采用下行缩进

  WHERE 子句书写时，每个条件占一行，语句另起一行时，以保留字或者连接符开始，连接符左对齐。例如：

```
WHERE           f1 = 1
AND             f2 = 2
OR              f3 = 3
```

  INSERT 语句，必须书写字段，字段可 5 个或者 6 个一组，中间用 TAB 分开 多表连接时，使用表的别名来引用列 供别的文件或者函数调用的函数，绝不应该使用全局变量交换数据 TAB 统一定义为 4 个空格，建议使用 Ultraedit 作为 SQL 书写工具

2.2.3.9 书写优化性能建议

- 避免嵌套连接。例如：A = B AND B = C AND C = D
- WHERE 条件中尽量减少使用常量比较，改用主机变量
- 系统可能选择基于规则的优化器，所以将结果集返回数据量小的表作为驱动表(FROM 后边最后一个表)

- 大量的排序操作影响系统性能，所以尽量减少 ORDER BY 和 GROUP BY 排序操作。如必须使用排序操作，请遵循如下规则：

  排序尽量建立在有索引的列上 如结果集不需唯一，使用 UNION ALL 代替 UNION

- 索引的使用

  尽量避免对索引列进行计算 尽量注意比较值与索引列数据类型的一致性 对于复合索引，SQL 语句必须使用主索引列 索引中，尽量避免使用 NULL 对于索引的比较，尽量避免使用!= 查询列和排序列与索引列次序保持一致

- 尽量避免相同语句由于书写格式的不同，而导致多次语法分析
- 尽量使用共享的 SQL 语句
- 查询的 WHERE 过滤原则，应使过滤记录数最多的条件放在最前面
- 任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边
- IN、OR 子句常会使用工作表，使索引失效；如果不产生大量重复值，可以考虑把子句拆开；拆开的子句中应该包含索引

2.2.3.10 其他经验性规则

- 尽量少用嵌套查询。如必须，请用 not exist 代替 not in 子句

```
#错误写法
SELECT ......
  FROM emp
  WHERE dept_no NOT IN ( SELECT dept_no
         FROM dept
         WHERE dept_cat='A');
#正确写法
SELECT ......
         FROM emp e
         WHERE NOT EXISTS ( SELECT 'X'
```

```
        FROM dept
        WHERE dept_no=e.dept_no
        AND dept_cat='A');
```

- 用多表连接代替 EXISTS 子句

```
#错误写法
SELECT ......
        FROM emp
        WHERE EXISTS ( SELECT 'X'
        FROM dept
        WHERE dept_no=e.dept_no
        AND dept_cat='A');
#正确写法
SELECT ......
        FROM emp e,dept d
        WHERE e.dept_no=d.dept_no
        AND dept_cat='A';
```

- 少用 DISTINCT，用 EXISTS 代替

```
#错误写法
SELECT DISTINCT d.dept_code,d.dept_name
        FROM dept d ,emp e
        WHERE e.dept_code=d.dept_code;

#正确写法
SELECT dept_code,dept_name
        FROM dept d
        WHERE EXISTS ( SELECT 'X'
        FROM emp e
        WHERE e.dept_code=d.dept_code);
```

- 使用 UNION ALL、MINUS、INTERSECT 提高性能
- 使用 ROWID 提高检索速度。对 SELECT 得到的单行记录，需进行 DELETE、UPDATE 操作时，使用 ROWID 将会使效率大大提高
- 使用优化线索机制进行访问路径控制
- 使用 CURSOR 时，显示光标优于隐式光标

3 拓展

3.1 Oracle 与 Mysql 之间部分函数和语法对比

3.1.1 UUID 生成

- 在 Oracle 中生成随机数 UUID 的方法是 sys_guid()

```
SELECT sys_guid() FROM dual
```

SYS_GUID(),是 Oracle 8i 后提供的函数。SYS_GUID 产生并返回一个全球唯一的标识符(原始值)由 16 个字节组成,在 Oracle 9i 和 Oracle 10g 生成的是 32 个字节。在大多数平台,生成的标识符由主机标符，执行函数的进程或者线程标识符，和进程或线程的一个非重复的值(字节序列)组成。 可以用来生成唯一标识 ID；

- 在 Mysql 中生成随机数 UUID 的方法是 uuid()

```
SELECT uuid() FROM dual
```

MySQL 实现了 UUID，并且提供 UUID() 函数方便用户生成 UUID。在 MySQL 的 UUID() 函数中，前三组数字从时间戳中生成，第四组数字暂时保持时间戳的唯一性，第五组数字是一个 IEEE 802 节点标点值，保证空间唯一。使用 UUID() 函数，可以生成时间、空间上都独一无二的值。据说只要是使用了 UUID，都不可能看到两个重复的 UUID 值。当然，这个只是在理论情况下。

比较：本质上都是方便用户生成随机的唯一索引，SYS_GUID()生成的是 32 位的字节;uuid()生成的则是带 4 根- 的 36 位的字节。

3.1.2 表之间左右连接使用

在关系型数据库中，左连接使用 LEFT JOIN ……ON……,右连接使用 RIGHT JOIN ……ON……

```
SELECT t1.* FROM TABLE1 t1 LEFT JOIN TABLE2 t2 ON t1.id = t2.id
SELECT t2.* FROM TABLE1 t1 RIGHT JOIN TABLE2 t2 ON t1.id = t2.id
```

但是在 Oracle 中却有一种简化写法，采用 += 、=+ 来替代左连接和右连接，写法如下

```
SELECT t1.* FROM TABLE1 t1 , TABLE2 t2 t1.id += t2.id
SELECT t2.* FROM TABLE1 t1 , TABLE2 t2 t1.id =+ t2.id
```

3.1.3 判断并替换 null 函数

在 Mysql 中判断 null 函数并且替换 null 的函数是 ifnull(exp1,result)

```
SELECT ifnull(t1.user_name,'张三') as user_name FROM TABLE1 t1
```

在 Oracle 中判断 null 函数并且替换 null 的函数是 nvl(exp1,result)

3.1.4 条件语句(函数)

在 Mysql 中条件判断语句可以使用 IF(exp1,exp2,exp3)函数，缺陷是只能判断单重条件

```
SELECT
  IF(t1.user_name='',"zhang san",t1.user_name) as user_name
FROM TABLE1 t1
```

在 Oracle 中条件判断语句可以使用 decode(exp1,result1,exp2,result2,result3)

函数,可支持多重判断

```
SELECT
  decode(t1.user_name='',"zhang san",t1.user_name is null,'li
si',t1.user_name) as user_name
FROM TABLE1 t1
```

对于比较复杂的多重条件判断，推荐使用通用的 CASE WHEN 条件 THEN 结果 WHEN 条件 THEN 结果 ELSE 结果 END 这种形式

```
SELECT
  CASE WHEN t1.user_name=''THEN
  "zhang san"
  WHEN t1.user_name is null THEN
  "li si"
  ELSE t1.user_name END as user_name
FROM TABLE1 t1
```

3.1.4 字符串拼接

字符串拼接是 sql 中常用到，在 Mysql 中提供了 concat(str1,str2,...strn)和 concat_ws(separator,str1,str2,...)

```
#无添加任何字符拼接
SELECT concat(column1,column2,……) as column_sttr FROM TABLE1;

#各个字符串之间以'-'拼接
SELECT concat_ws('-',column1,column2,……) as column_sttr FROM TABLE1;
```

在 Oracle 中，字符拼接是以 "||" 方式,此种方式显得更灵活点

```
SELECT column1 ||column2 || column3 as column_sttr FROM TABLE1;
```