**数据库代码规范**

|  |  |
| --- | --- |
| 模板编号： |  |
| 文件版次： | V1 |
| 编写人/日期： |  |
| 审核人/日期： |  |
| 批准人/日期： |  |

**变更记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **版本号** | **修改单号** | **修改条款及内容** | **修改人/日期** | **审核人/日期** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

注：对该文件内容增加、删除或修改均需填写此变更记录，详细记载变更信息，以保证其可追溯性。

# 目的

1、 代码规范相当重要。代码规范提高软件代码的可读性，使得开发人员快速和彻底的理解新代码。好的代码风格不仅会提高可读性，而且会使代码更健壮，更为重要的是在修改时不容易出错。

2、在现代软件开发中，维护工作会占用80%的时间，尤其是后台数据库脚本的开发和修改，而且开发者和维护者通常不是同一个开发人员。这意味着你经常要阅读和修改别人开发的脚本，别人也同样可能需要阅读和修改你写的脚本程序。

本规范基于SQL、PL/SQL语言规范，定义了应用程序的排版规则、脚本规则和开发最佳实践；用于保障后台数据库的规范性、可读性和可维护性。

# 预期读者

所有应用开发组人员。

# 适用范围

使用SQL、PL/SQL语言编写脚本。

# 整体约束

在使用SQL、PL/SQL语言开发过程中，要求每个脚本文件排版风格一致、代码风格一致；必须有注释，包括源文件注释、包注释、函数注释和过程注释等；必须遵守统一的命名规则；注意表建立规则（包括基表、视图、物化视图、分区表）、索引、游标、任务的使用、异常块的处理、日志输出等。

# 规范内容

## 开发工具

为提高开发效率，降低开发门槛，帮助新员工快速搭建开发环境，避免工具软件使用冲突影响工作，对开发设计人员使用的日常开发工具，进行如下约定：

数据库：服务器端为Oracle 11gR1，客户端版本要求10g以上

数据库连接器：PLSQL、TOAD

代码管理工具：推荐使用TortoiseSVN 1.7以上版本

文本编辑器：Edit Plus、UltraEdit、Sublime

文件字符集：UTF-8

建模工具：POWER DESIGNER 15.3

说明：

SQL脚本开发完毕后，需进行上传至SVN并本地最好留有备份，具体上传规则和上传位置见公共资料文档，注意使用匹配的TortoiseSVN，避免版本不匹配，出现软件SVN更新操作异常。

## 文件

### 文件后缀

脚本使用下列文件后缀：

| 文件类别 | 文件后缀 |
| --- | --- |
| SQL脚本 | .sql |
| 数据文件 | .dbf |
| 导入/导出文件 | .dmp |
| 备份数据文件 | .dmp |
| 备份脚本 | .sh |

### 常见文件名

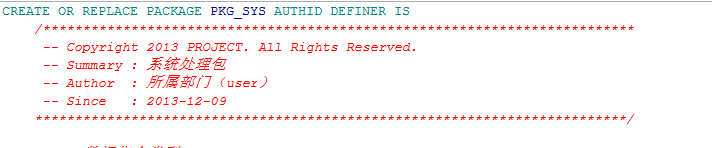
常用的文件名包括：

|  |  |
| --- | --- |
| 文件名 | 用途 |
| 00\_STORAGE.sql | 数据库脚本备份 |
| 10\_TABLES.sql | 数据库表结构脚本备份 |
| 20\_FUNCTION.sql | 数据库函数脚本备份 |
| 30\_VIEWS.sql | 数据库视图脚本备份 |
| 40\_PROC.sql | 数据库过程脚本备份 |
| 50\_JOBS.sql | 数据库过程脚本备份 |
| 60\_DATAS.sql | 数据库数据脚本备份 |
| 70\_LOGS.sql | 数据库日志脚本备份 |
| 80\_ANY1.sql | 数据库其他备份（同义词、连接、SQL语句等） |
| 90\_ANY2.dmp | 数据库其他备份（数据库导入导出备份） |

## 命名规则

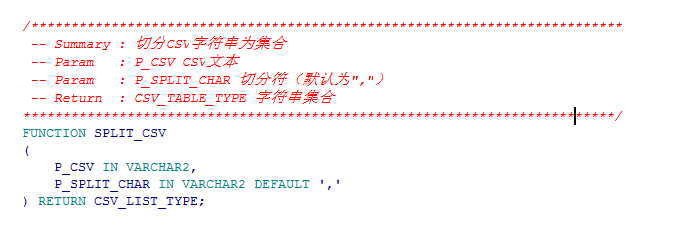
### 包命名

说明：包名称以PKG和单一名词组合而成，中间以“\_”分割，如：PKG\_SYS。



### 函数命名

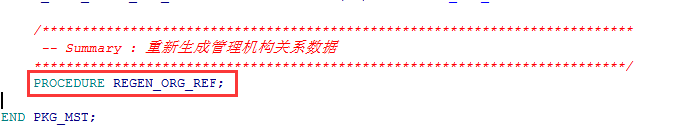
说明：函数名称简写，中间以“\_”分割,具有实际意义如：SPLIT\_CSV.要求函数描述说明、参数和返回值。



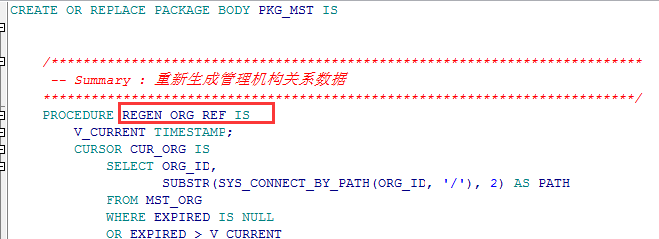
### 过程

说明：过程均使用在包下，禁止单独定义；首先在包内进行声明过程，描述过程用途、参数。然后在包体描述过程用途、参数、脚本编写。

包声明：



包体：



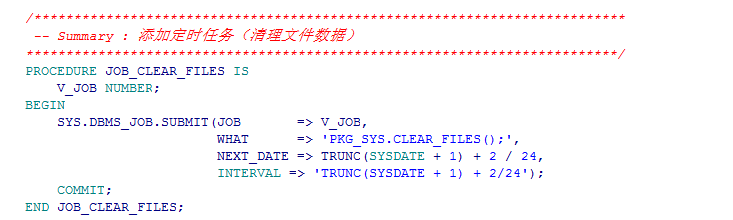
### 定时JOB

说明：JOB在包内声明，包体实现，并注释描述、参数。

包声明：



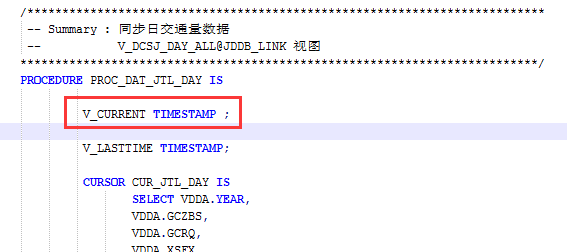
包体：



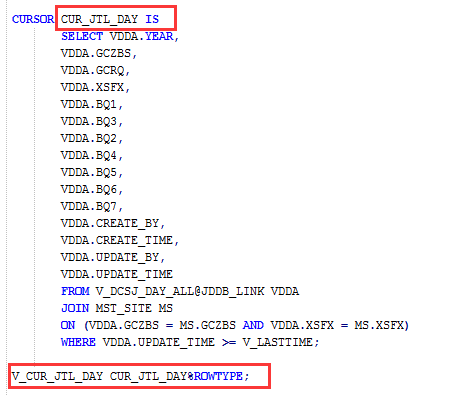
### 其他数据类型

说明：如表类型、字段类型等均按照以下规则：

（1）脚本变量命名：V\_NAME



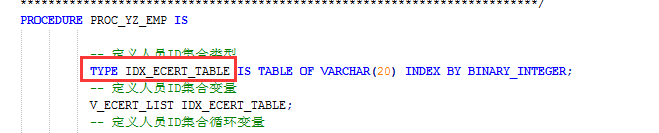
（2）游标及其变量声明：CUR\_NAME、V\_CUR\_NAME



（3）常量声明：C\_NAME



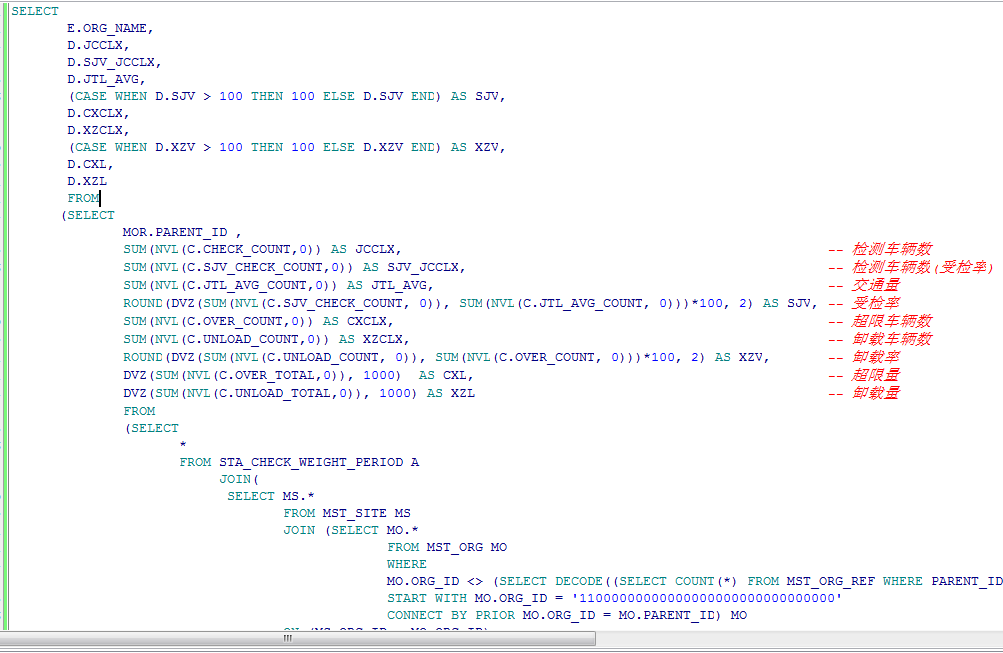
1. 其他变量类型声明：具有实际意义的名称，中间以“\_”分割,具有实际意义如：



## 注释

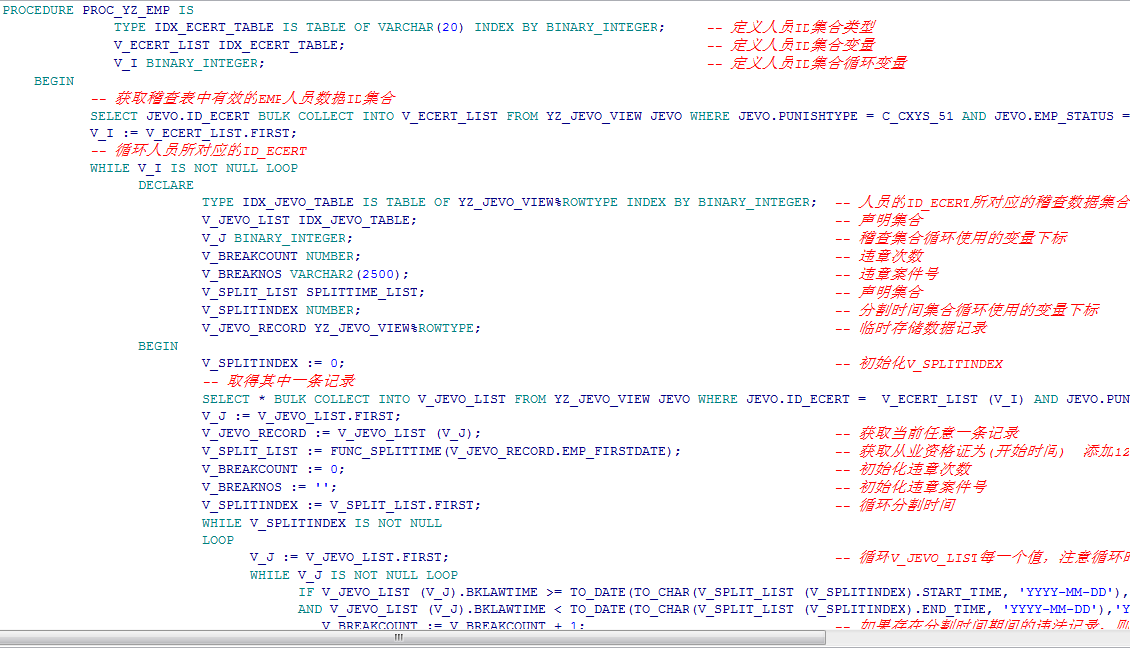
### 5.4.1 SQL注释

我们在编写SQL脚本的时候，尤其是复制SQL语句，一定要注释清楚变量、表级联关系、统计字段，以及SQL的查询功能说明，并上传SVN中去，这样方便开发和维护。如下所示：



### 5.4.2 PL/SQL 注释

PL/SQL编程如果不写注释，那么后期维护会非常麻烦，因为一个负责的业务往往会写几百上千行的代码，后期运维有时候甚至需要花几天时间去看一个业务逻辑。所以在编写PL/SQL时候，注释尤为关键，一定要写好注释，写清楚业务逻辑需求，实现思路；如下所示：



## 编程惯例

### 5.5.1 数据库设计

（1）所谓数据库设计，一定要在遵循规则的基础上进行设计，对需求要足够了解，尽量把表设计的易用性更强，比如添加扩展字段，预留字段，通用标识字段等。

（2）主外键关联关系一定要添加，一般情况下不建议手工维护主外键，除非情况特殊（如需求不确定，数据不稳定，不准确等）

（3）数据库表结构一般情况下需要设置主键，需要设置4个数据交换字段CREATE\_BY、CREATE\_TIME、UPDATE\_BY、UPDATE\_TIME。

（4）在设计好表结构后，需要添加相关字段的约束，以便在最后一层控制数据入库，当然我们是按照三层CHECK的原则，前端、应用端、数据库端。最后保证数据的完整和有效性。

（5）在建立数据库索引时，应该认真考虑，如何建立，比如我们可以去对数据库的主键进行设置，让其有更多作用。

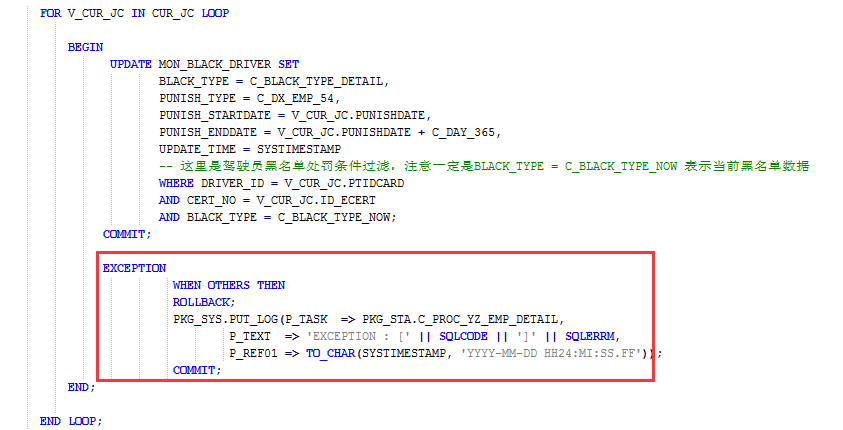
### 5.5.2 SQL优化

SQL优化的手段很多，在数据库表结构没有大问题的前提下，我们的数据优化就要从SQL语句去考虑问题。

1. 基本的SQL优化，如我们要大写SQL语句，oracle编译SQL是按照大写去编译的；
2. 如OR/AND使用的先后顺序，我们的SQL语句在编译时是从后向前进行编译的，这个时候就需要酌情去选择WHERE后过滤字段的先后顺序。
3. 在数据库大并且可选的情况下我们尽量避免使用GROUP BY进行分组，如果需要使用GROUP BY 尝试先使用WHERE条件过滤一部分数据，使GROUP BY的数据内容尽可能的减小，另GROUP BY 尽量选择索引字段。
4. 我们在使用索引做查询时，不用对索引字段进行附加操作，这样会导致索引失效。
5. 在聚合、统计分析时，尽量使用ORACLE分析函数。
6. 另外最后需要注意细节问题，比如IN/EXISTS、JOIN语句用法等基本无需强制进行优化，提升工作效率。
7. 建立索引、分区、物化视图、并行查询需要针对特定的场景下，不要盲目操作。

### 5.5.3 PL/SQ使用

1. 使用PL/SQL时注意在WHILE/LOOP/FOR循环里添加BEGIN END 匿名块，目的是进行异常数据的日志管理记录，另每一个作用域都需要进行异常的日志输出。如下所示：



1. 在无特殊需求的情况下，尽量减少视图的使用，本质上没有差别，可以使用SQL语句。
2. 在无特殊需求的情况下，尽量减少索引的建立，这样会增加IO的压力与内存。
3. 在无特殊需求的情况下，尽量减少触发器的使用，同样增加IO操作。
4. 在无特殊需求的情况下，尽量减少游标的使用，游标的查询性能比较昂贵。
5. 在无特殊需求的情况下，尽量减少存储过程使用，尽可能用一条复杂SQL进行查询替代，存储过程性能比较昂贵。
6. ...