Table of Contents

SQLE 中文技术参考手册

关于SQLE

SQLE 是由上海爱可生信息技术股份有限公司开发并开源，支持SQL审核、索引优化、事前审核、事后审核、支持标准化上线流程、原生支持 MySQL 审核且数据库类型可扩展的 SQL 审核工具。

产品特色

1. 支持通过插件的形式扩展可审核上线的数据库类型，无需升级软件，导入审核插件即可获对应数据库类型的审核上线能力，使用平台所有功能；
2. 支持标准的 HTTP API，可与其他内部流程系统对接；
3. 支持 DDL，和 DML 同时审核，并实现同工单内语句上下文关联；
4. 支持在审核规则外对语句做必要的对象验证，防止实际执行时库表不存在等情况。

主要功能

如何使用具体的功能可以参考第三章：功能模块

平台管理

1. 支持用户、角色和权限的精细管理；
2. 支持 LDAP 登录；
3. 支持配置上线数据库；
4. 支持基于角色的资源隔离；
5. 支持配置审核规则模板；
6. 支持对不同的数据库应用不同的规则模板。

SQL 审核

1. 支持基于规则的审核建议输出；
2. 支持工单审批流程，支持工单隔离；
3. 支持邮件推送审批事件；
4. 支持生成回滚语句；
5. 支持SQL录入关键字联想；
6. 支持审核 MyBatis XML 文件；

7. 支持审核建议按 SQL 归类去重展示；
8. 支持审核报告下载。

SQL 优化

1. 支持索引优化(使用文档)；

SQL 上线

1. 支持 SQL 上线；
2. 支持对大表进行 Online DDL(使用文档)；
3. 支持定时上线。

SQL 审核任务

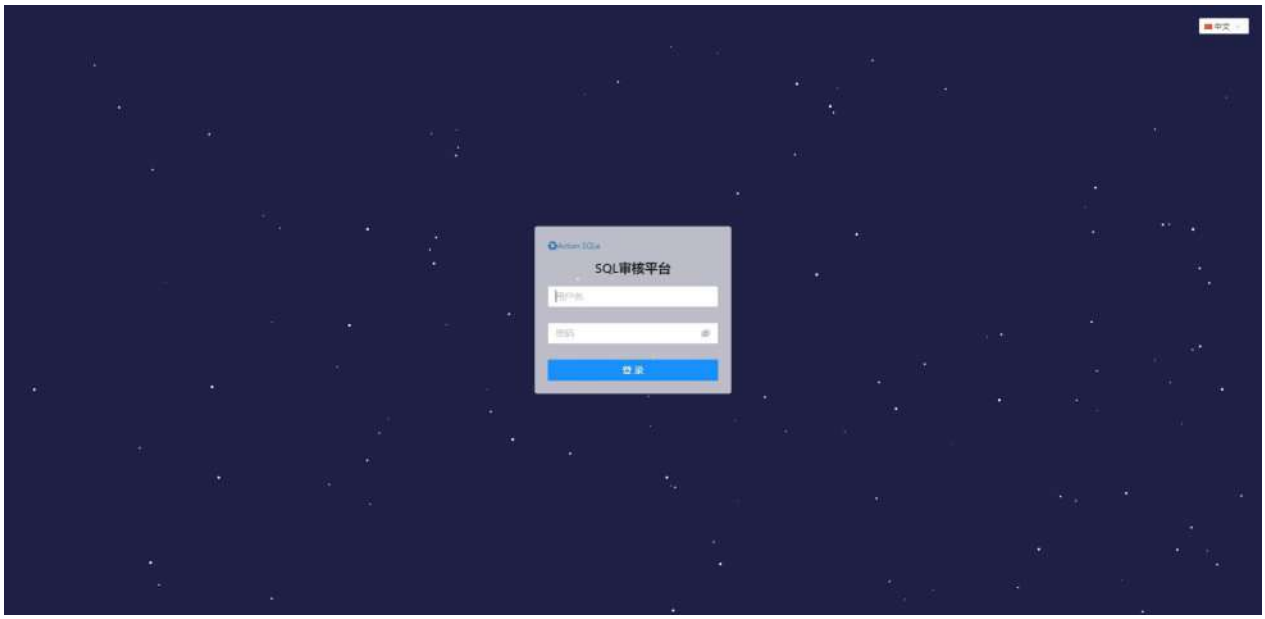
1. 支持通过标准接口收集来自业务的 SQL 统计信息；
2. 提供 MyBatis 文件扫描上传程序（scanner），可集成CI/CD；
3. 支持 MySQL 库表元数据采集审核；
4. 支持 Oracle TopSQL 采集审核;
5. 支持静态审核，即脱离上线数据库审核；
6. 支持定时审核，既有 SQL 进行生成审核报告。

企业版功能

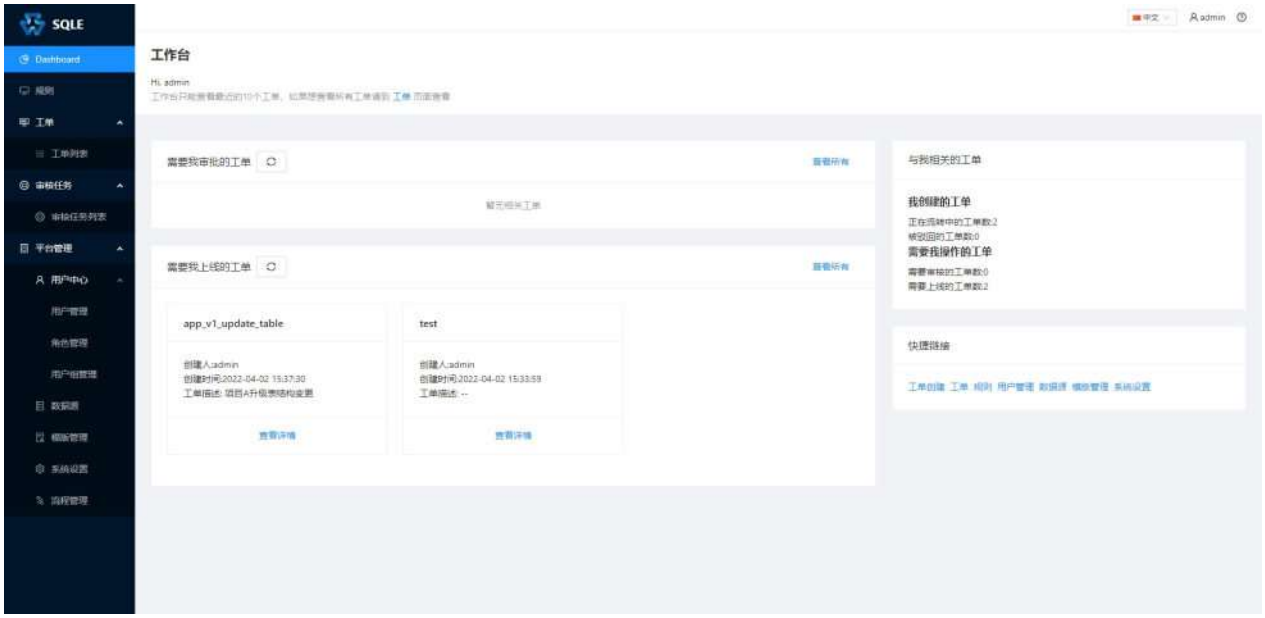
1. 支持审核白名单；
2. 支持审批流程可视化配置；
3. 支持 MySQL 慢日志采集进行 SQL 审核；
4. 支持 Java 应用程序运行SQL 审核；
5. 支持工单通过微信企业号推送。

产品展示

登录界面



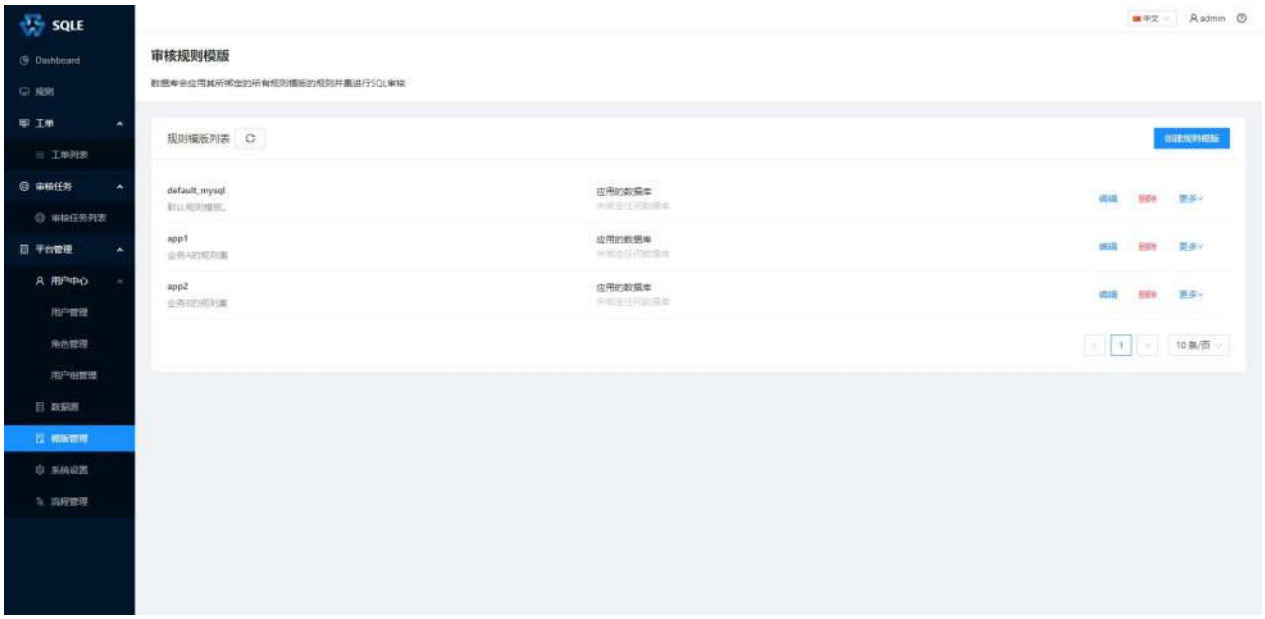
dashboard 界面



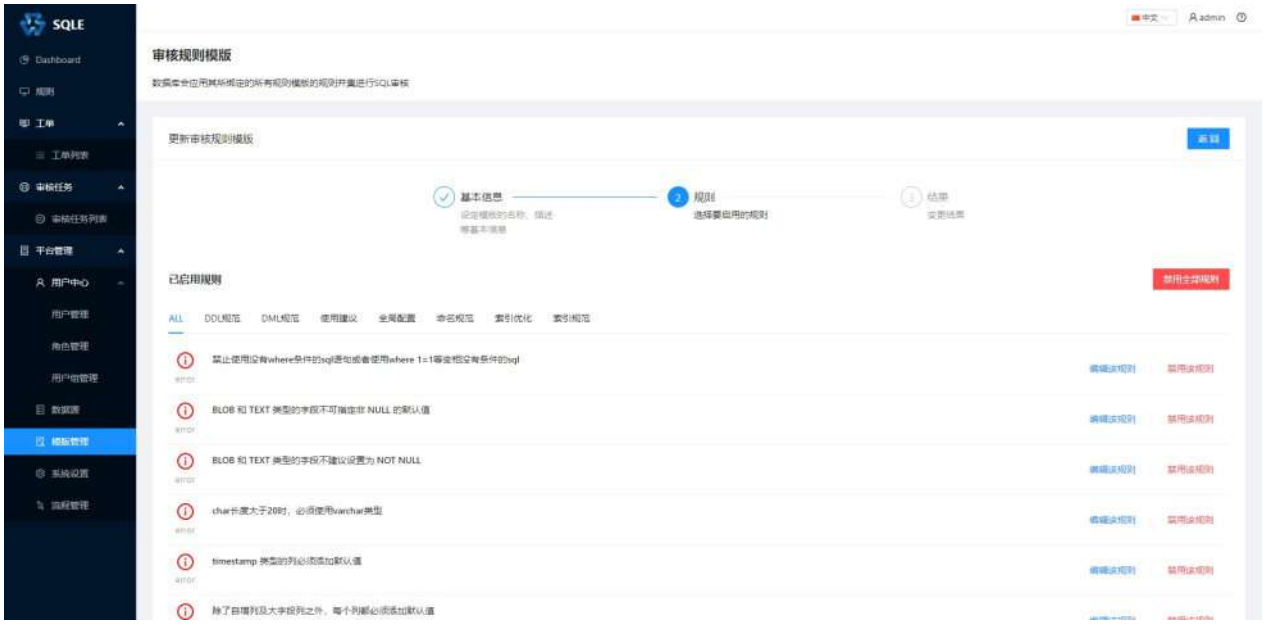
审核规则界面

规则列表

可以为不同的项目配置不同的规则集

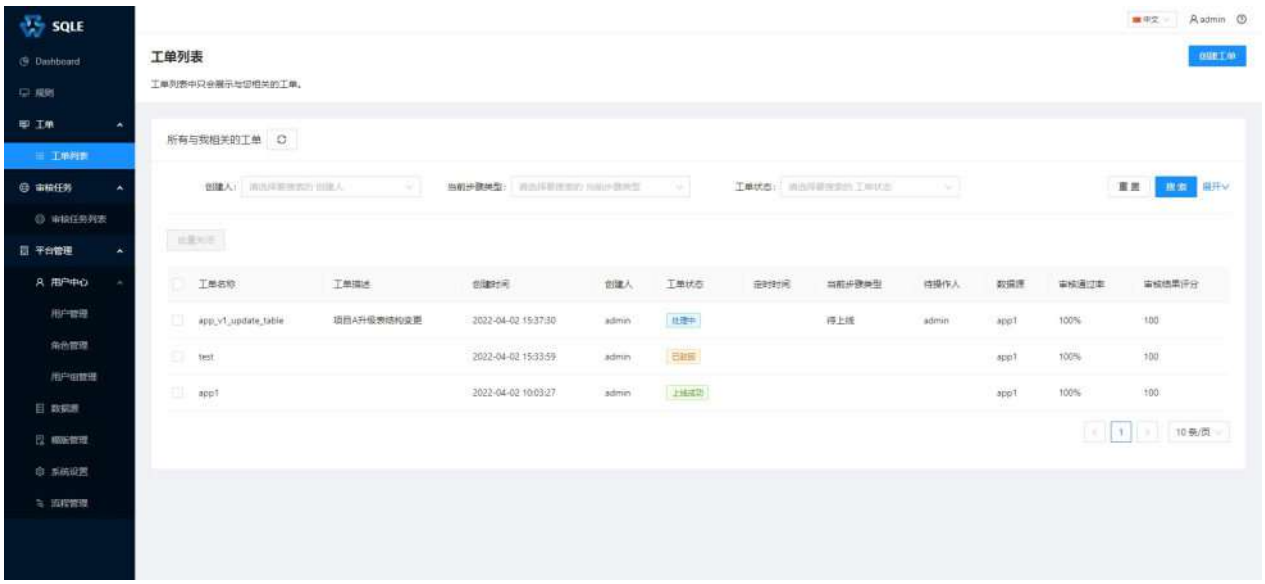


修改规则集

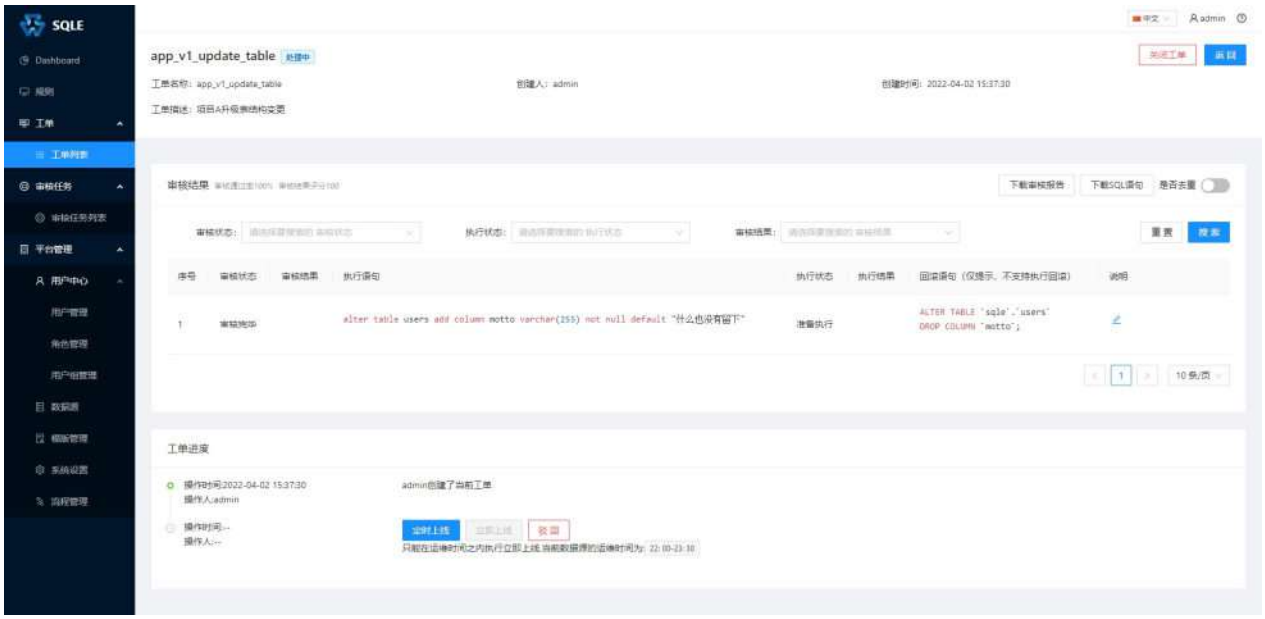


工单界面

工单列表

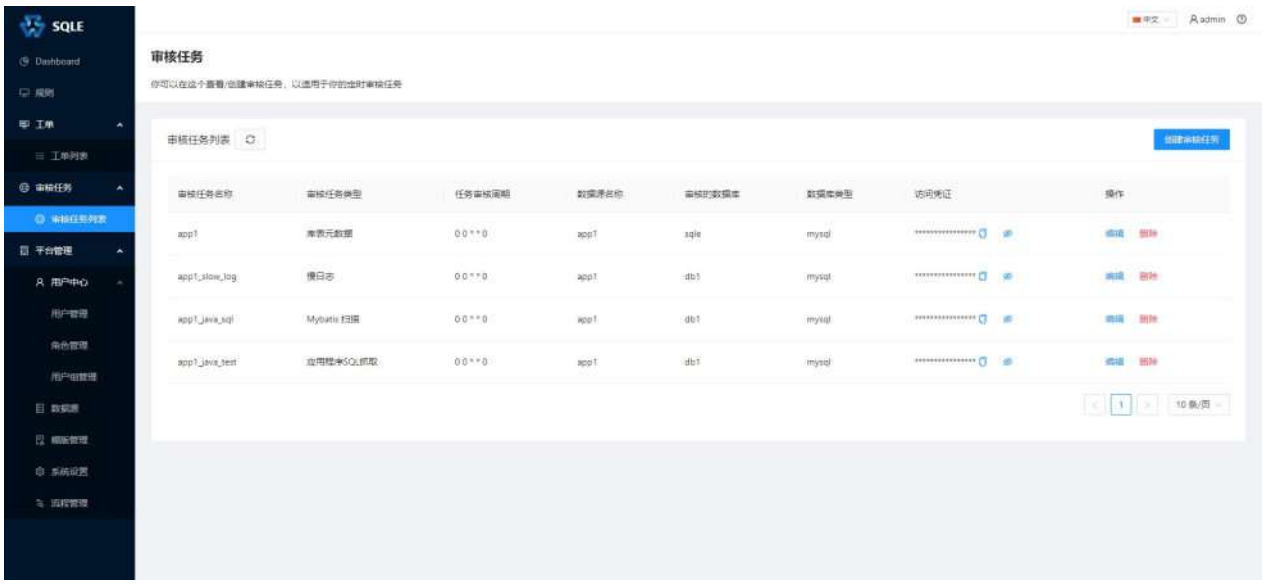


工单详情

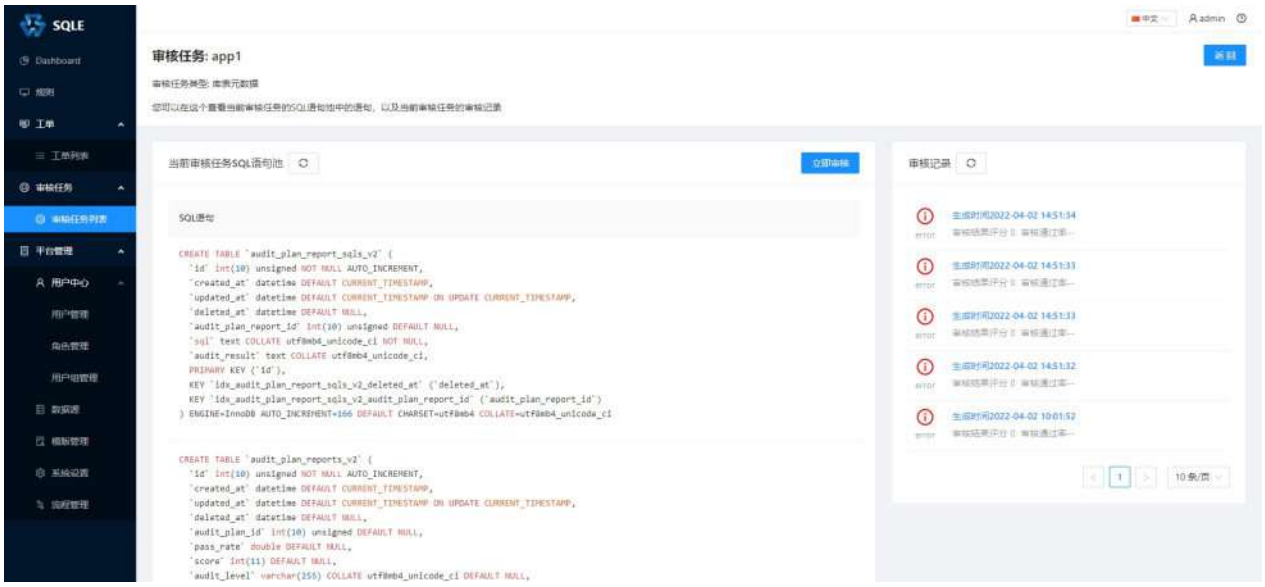


审核任务界面

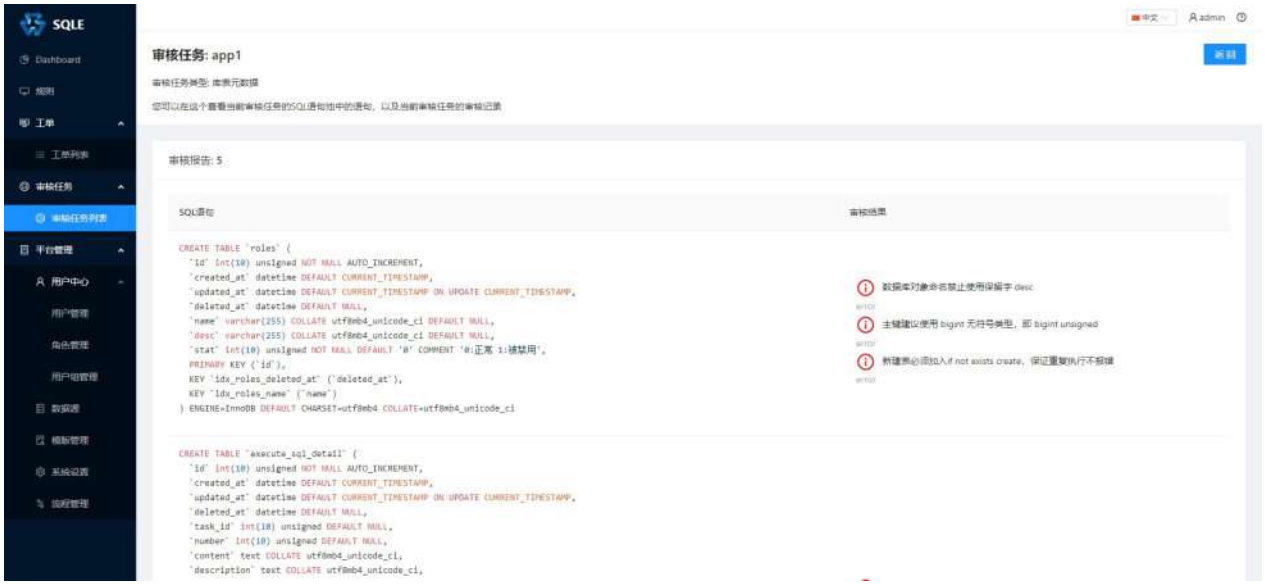
审核任务列表



审核任务SQL集



审核任务审核报告



安装部署

目录

源码安装

环境准备

- CentOS 7
- MySQL 5.7
- Docker
- Docker Compose（可选）

获取源码

SQLE 前端与后端代码分两个仓库维护，所以想要通过源码安装，需要构建前后端代码。

首先选择一个源码安装的工作目录，例如 ~/sqle-build/。

获取后端源码：

```
cd ~/sqle-build/
git clone https://github.com/actiontech/sqle.git
```

获取前端源码：

```
cd ~/sqle-build/
git clone https://github.com/actiontech/sqle-ui.git
```

编译源码

编译前端代码，并将前端代码拷贝至后端代码目录：

```
cd ~/sqle-build/sqle-ui
docker container run --rm -v $PWD:/app -w /app node:15.3.0 sh -c "yarn install && yarn build"
rm -rf ~/sqle-build/sqle/ui && cp -r ~/sqle-build/sqle-ui/build/ ~/sqle-build/sqle/ui/
```

编译后端代码并打包：

```
cd ~/sqle-build/sqle
make docker_rpm
```

这是会在当前目录下生成一个 SQLE 的 RPM 包，后续的步骤就是部署 RPM 包。

部署

安装部署 SQLE 的 RPM 包可以使用两种方式，分别是：

RPM 部署

具体请参考 RPM 部署。

Docker 部署

前提是通过前面的步骤成功生成了 RPM 包。

```
cd ~/sqle-build/sqle
make docker_image
make docker_start
```

启动成功后即可开始使用。

RPM 部署

环境准备

- CentOS 7
- MySQL 5.7

下载安装包

下载 SQLE 的 RPM 安装包，下载连接点击这里。

安装 SQLE Server

执行 RPM 的安装命令：

```
rpm -ivh /path/to/sqle-ce-${version}.qa.el7.x86_64.rpm --prefix=/opt/sqle
```

安装完 SQLE Server 后，需要修改默认的配置文件模板名。

```
cd /opt/sqle/etc
mv sqled.yml.template sqled.yml # 修改后确保 sqled.yml 的文件 owner 为 actiontech-universe:actiontech
```

然后根据实际情况修改相应的配置，配置文件说明见开始使用中**配置文件**一节。

准备 SQLE 存储数据库

准备一台 MySQL5.7 作为 SQLE Server 的后端存储数据库。

执行下面命令创建 SQLE Server 需要的 schema：

```
CREATE DATABASE IF NOT EXISTS sqle default character set utf8mb4 collate utf8mb4_unicode_ci
```

将 sqle 填入配置文件的 server.db_config.mysql_cnf.mysql_schema 中。

启动 SQLE Server

SQLE Server 进程通过 Systemd 管理。在成功安装 RPM 后，执行启动命令：

```
systemctl start sqled
```

检查是否启动成功：

- 检查工作目录 /opt/sqle 下是否生成 pid 文件
- 检查进程运行状态是否正常（执行 systemctl status sqled.service）

启动成功后即可开始使用。

Docker 部署

Docker 使用说明

Docker hub 地址

actiontech/sqle-ce

Docker 参考命令

```
docker run -d -it \
--name sqle-server \
-p 10000:10000 \
-e MYSQL_HOST="10.10.10.10" \
-e MYSQL_PORT=3306 \
-e MYSQL_USER="username" \
-e MYSQL_PASSWORD="password" \
-e MYSQL_SCHEMA="sqle" \
actiontech/sqle-ce:latest
```

参数说明

使用环境变量传递参数，这些指定的配置会映射到 sqled.yml 配置内，指定数据库的配置等；

1. MYSQL_HOST: 数据库地址；
2. MYSQL_PORT: 数据库端口；
3. MYSQL_USER: 数据库用户；
4. MYSQL_PASSWORD: 数据库密码；
5. MYSQL_SCHEMA: SQLE 服务使用的指定的 schema；
6. DEBUG: 输出 debug 日志；
7. AUTO_MIGRATE_TABLE: 是否自动创建表格和初始化数据。

注意点

1. 使用 docker 命令运行容器需要提前准备 MySQL 服务并且创建对应的数据库 schema，参考：RPM 部署，建议参考下一节采用 docker compose 部署；
2. 如果 sqle 容器启动失败可以使用 `docker logs sqle-server` 查看启用日志，其中 `sqle-server` 是容器名称。

Docker Compose 部署

使用 docker-compose 部署不需要提前准备 MySQL 环境，可以一键生成 SQLE 环境

配置文件

下载地址：docker-compose.yaml

使用说明

将 docker-compose.yaml 文件下载到本地目录，并进入目录内，通过环境变量 `SQLE_IMAGE` 指定不同版本的 SQLE 镜像，默认是 latest。

```
SQLE_IMAGE=actiontech/sqle-ce:latest docker-compose up -d
```

启动成功后即可开始使用。

开始使用

访问 SQLE

在检查进程启动成功后，即可通过 SQLE UI 访问：http://127.0.0.1:10000

ps：IP 和 Port 请根据前面的配置自行替换

默认账户

- 用户名：admin
- 密码：admin

功能说明

请参考功能模块一章。

配置文件

在安装前或者安装后，可能会需要修改默认的配置文件。

默认的配置文件如下：

```
server:
 sqle_config:
  server_port: 10000
  enable_https: false
  cert_file_path: './etc/cert.pem'
  key_file_path: './etc/key.pem'
  auto_migrate_table: true
  debug_log: false
  log_path: './logs'
  secret_key: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' #从 v1.2203.0 版本引入
 db_config:
  mysql_cnf:
   mysql_host: '127.0.0.1'
   mysql_port: 3306
   mysql_user: 'root'
   mysql_password: '123456'
   mysql_schema: 'sqle'
```

可根据实际环境自定义相关配置，下面是配置文件项的说明：

sqle_config（SQLE Server 的运行配置）

- server_port: sqle 服务的 http 端口，默认10000
- enable_https： 是否开启https，默认不开启
- cert_file_path： https 证书路径
- key_file_path: https 私钥路径
- auto_migrate_table: 自动创建表结构，初始化数据
- debug_log: 开启debug模式，打印更多日志，会打印业务SQL，开发环境可开启
- log_path： 日志目录
- secret_key: 全局AES加密秘钥，影响登录和用户密码等的存储；**生成环境建立填写替换掉程序默认值**；格式是32位随机字符串。

db_config.mysql_cnf（SQLE Server 存储数据的数据库运行配置）

- mysql_host: 指定数据库地址
- mysql_port: 指定数据库端口
- mysql_user: 指定数据库用户
- mysql_password: 指定数据库密码
- mysql_schema: 指定数据库 schema

功能模块

目录

权限管理

目录

背景

目前 SQLE 的权限设计遵循如下原则：

- admin 用户是拥有最高权限的账户，可以进行任何操作；
- 普通用户对资源可进行的操作，应该受限于该用户所关联的角色的权限范围；
- 一个角色的权限由「资源」和「动作权限」两部分构成。只有当两项都不为空时，该角色才能对具体的资源对象进行实际操作；
- 普通用户所关联的角色由两部分组成，一是该用户直接绑定的角色，二是该用户所在的用户组所绑定的角色。

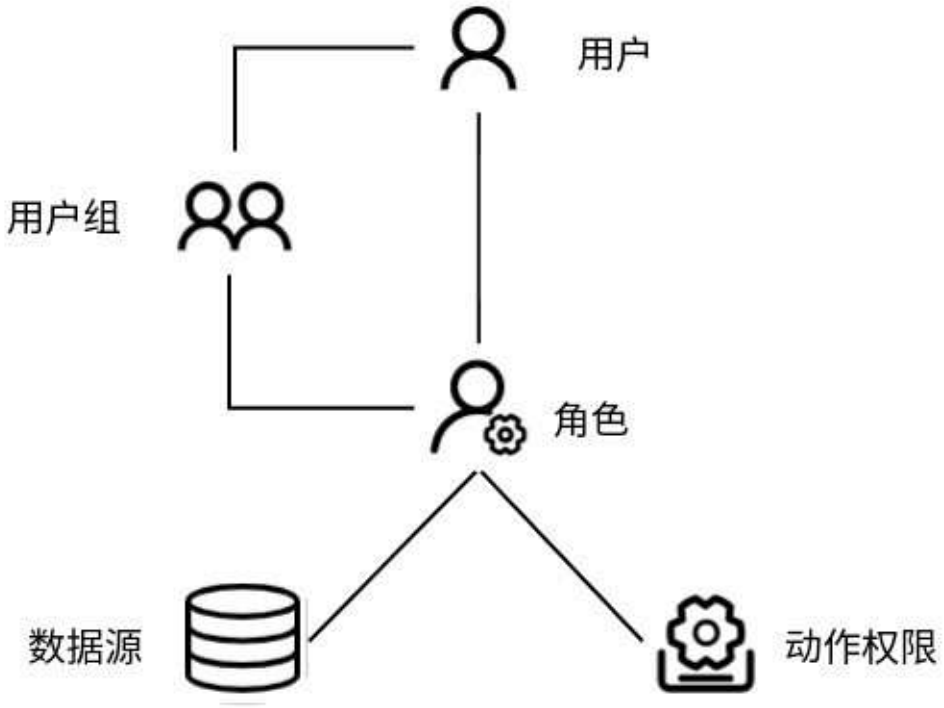
目前 SQLE 中的资源包括：

- 数据源
- 审核工单
- 审核任务

权限包括

- 创建/编辑工单
- 查看他人创建的工单

基于以上的描述，SQLE 实现了基于 RBAC 的权限管理系统。



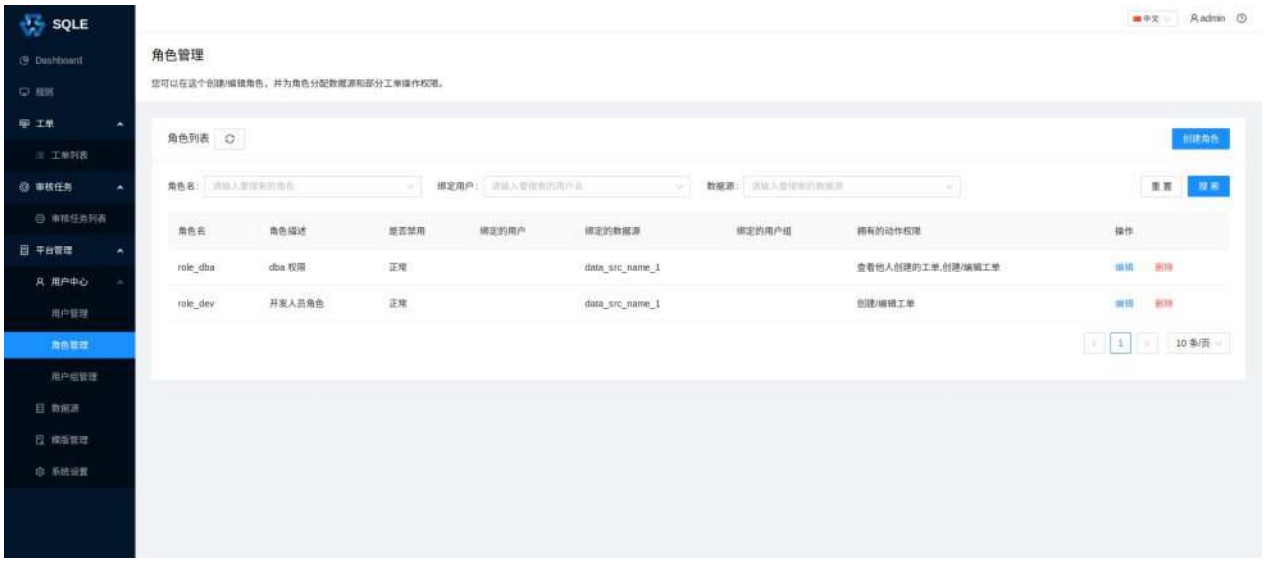
角色管理

创建角色

在左侧导航栏的「平台管理」-「用户中心」-「角色管理」页面中，点击「创建角色」，填写角色相关信息，如下图：

- 角色名；
- 角色描述（选填）：略；
- 数据源（选填）：如果当前还未添加数据源，可以在添加数据源后，通过「编辑」修改；
- 绑定用户（选填）：如果当前还未创建用户，可以在创建用户后，通过「编辑」修改；
- 绑定用户组（选填）：如果当前还未创建用户组，可以在创建用户组后，通过「编辑」修改；
- 动作权限(选填)：可以在创建时进行复选。或者在创建后，通过「编辑」修改。目前支持的动作权限有：
  - 查看他人工单（20100）：当角色拥有该动作权限时，角色可以查看绑定的数据源上其他用户创建的工单；
  - 创建/编辑工单（20200）：当角色拥有该动作权限时，角色可以在绑定的数据源上创建工单。

修改角色



点击角色的「编辑」按钮，如下图：

上诉操作将角色 DBA 下的用户数据源更新为 `data_src_name_1` 和 `data_src_name_2` ，并且绑定了用户组 `dba_group` 。

用户、用户组与角色

用户是否能够进行某种操作，受限于用户所关联的角色的权限。 当两个角色绑定的动作权限与数据源都相同时，可以认为这两个角色是「相同」或「相等」。 结合用户组管理来看，一个用户所**关联**的角色由两部分组成：

- 用户自身所**绑定**的角色；
- 用户所在的用户组所**绑定**的角色。

因此，一个用户的权限实际上是上述两组角色的并集。

当出现以下情况时，用户会失去一个角色的权限：

- 将用户与该角色解绑，且用户没有关联其他相同权限的角色；
- 将该角色禁用，且用户没有关联其他相同权限的角色；
- 将该角色所在的用户组禁用，且用户没有关联其他相同权限的角色；
- 将该角色移出用户组，且用户没有关联其他相同权限的角色。

用户管理

SQLE 的用户通常包括以下 3 类：

- 数据库运维工程师（DBA）：SQLE 的管理人员，他们通常负责平台的搭建与日常管理；
- 研发工程师：通常使用自己的账户创建工单；
- 项目管理员：通常是工单审批流程（见审核工单管理）中的一员。

创建用户

在左侧导航栏的「平台管理」-「用户中心」-「用户管理」页面中，点击「创建用户」，填写用户相关信息，如下图：

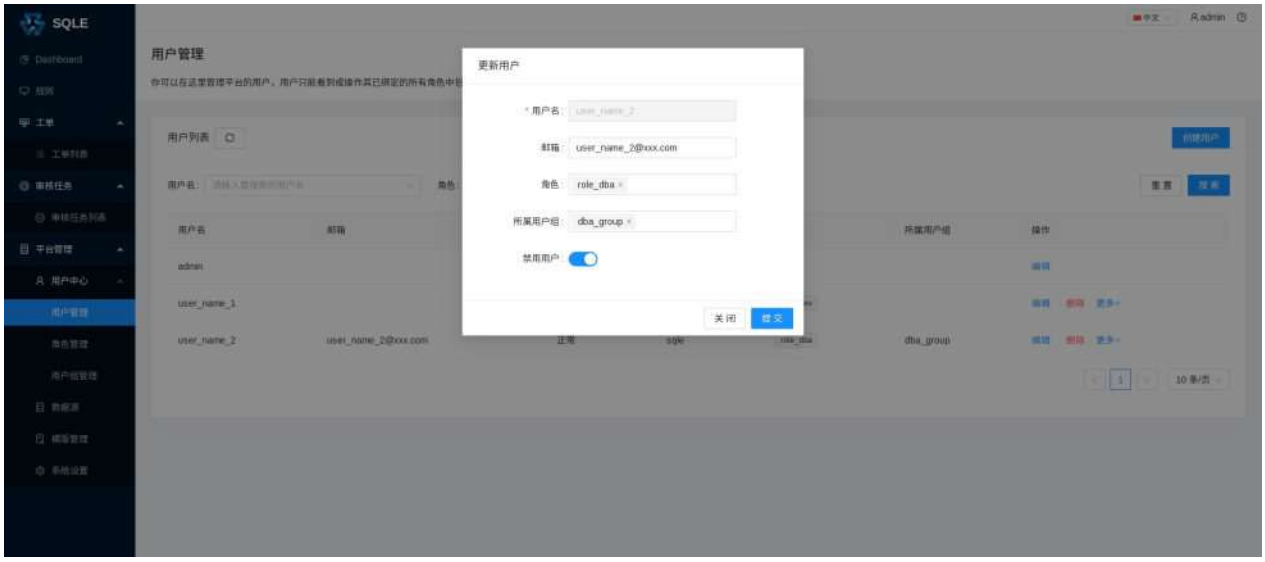
- 用户名；
- 密码；
- 确认密码；
- 邮箱：填写邮箱后，将会以邮件形式通知工单流转状态;
- 角色：如果当前还未创建角色，可以在创建角色后，通过「编辑」修改；
- 所属用户组：如果当前还未创建用户组，可以在创建用户组后，通过「编辑」修改。

修改用户

点击用户的「编辑」按钮，如下图：

上诉操作将用户 dba 的邮箱地址更新为 `user_name_2@xxx.com` ，并且绑定了角色和用户组。

管理员还可以对指定的用户进行「禁用」或者「启用」。 当用户被管理员禁用后，该用户将无法登录，也无法进行任何操作。 管理员用户永远无法禁用自己本身。



用户组的管理

当管理员需要批量管理用户时，可以使用用户组来管理多个相同属性的用户。 比如依照「运维工程师」、「研发工程师」、「项目管理员」进行分别授权与管理。

需要注意的是：

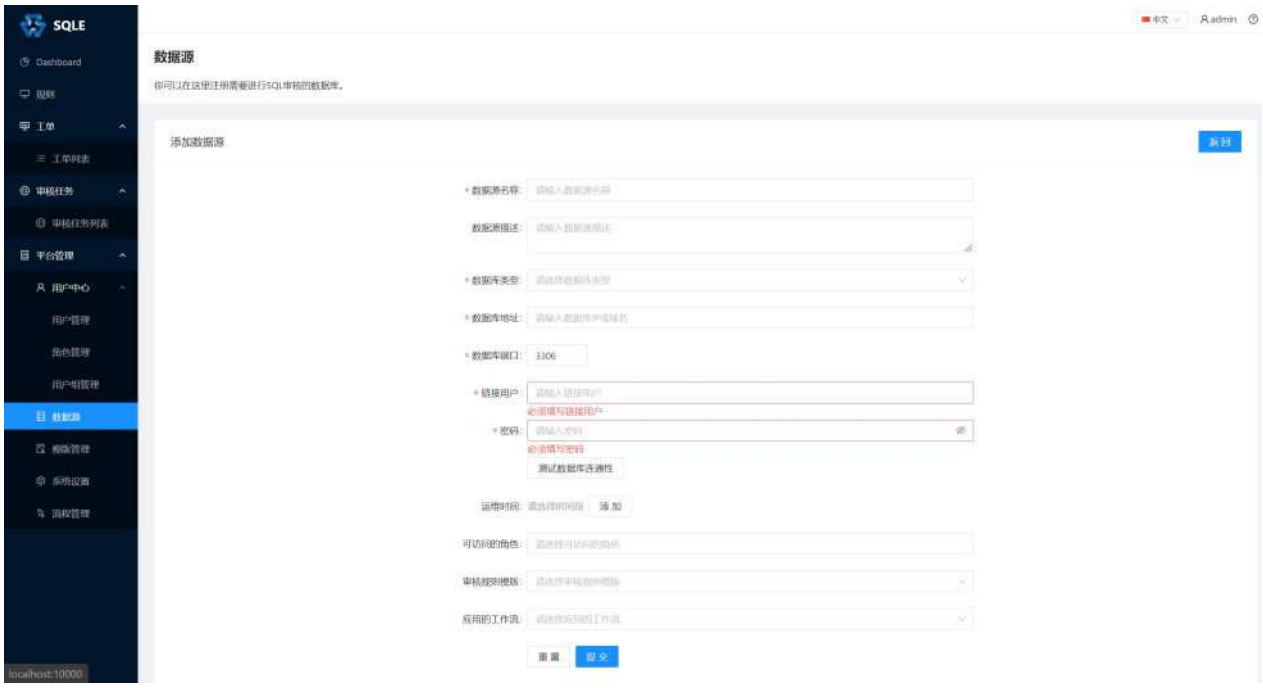
- 用户会继承所在用户组所绑定的角色。
- 当一个用户组被禁用，用户不会被禁用，但会失去该用户组所绑定的角色。

数据源管理

数据源管理 是 SQLE 提供的管理线上数据库的功能。一个数据源对应着一个数据库实例。在审核 SQL 时，如果指定了关联的数据源（动态审核），可以得到更加详细的审核结果。详情可参考审核工单一章。

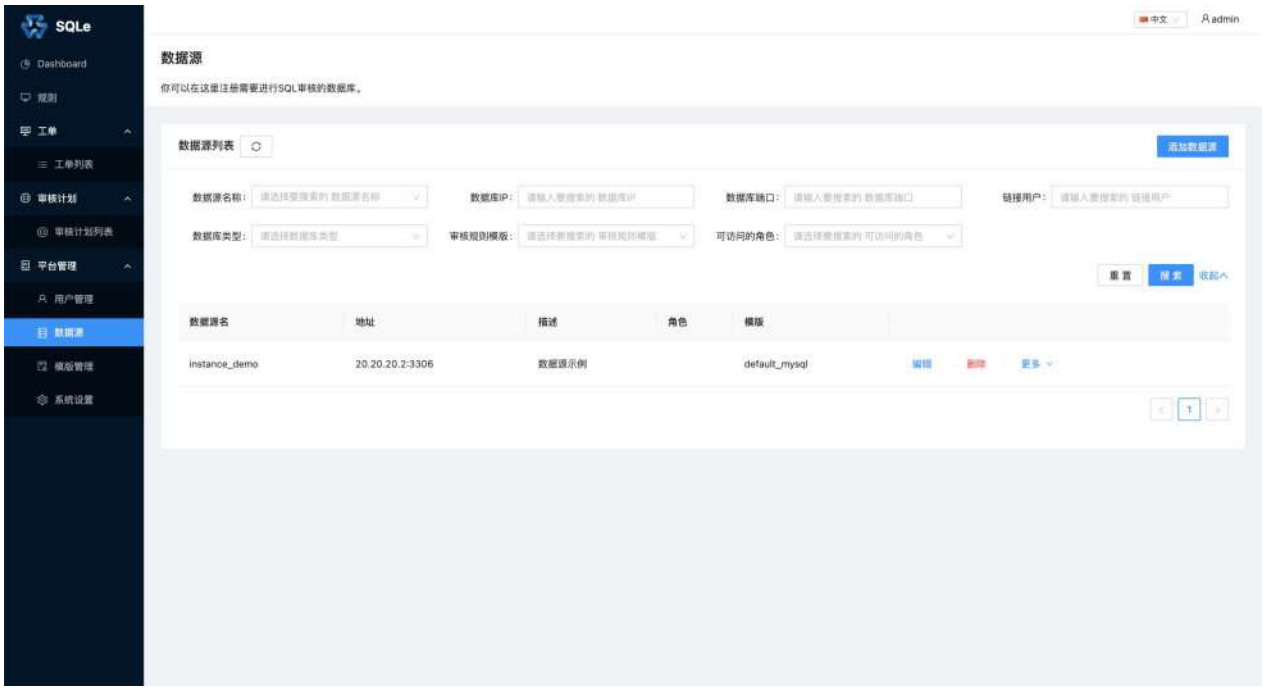
添加数据源

在左侧导航栏的「平台管理」中的「数据源」页面中，点击「添加数据源」，填写数据源相关信息，如下图：



- 数据源名称：略
- 数据源描述：略
- 数据库类型：社区版只支持添加 MySQL 数据库，详情见审核插件。
- 数据库IP：略
- 数据库端口：略
- 连接用户：略
- 密码：略
- 测试数据库连通性：填写完必要的信息后，点击此按钮可以测试 SQLE 是否能访问该数据库。
- 运维时间（选填）：如果配置了运维时间, 工单将只能在运维时间内上线(立即上线和定时上线都受此影响), 支持配置多个运维时间, 默认无运维时间,即不限制上线时间
- 可访问角色（选填）：表示数据源可以被哪些角色访问。
- 审核规则模板（选填）：表示数据源有哪些审核规则。
- 应用的工作流（选填）：表示在数据源上创建的工单所应用的工作流。社区版只支持默认的工作流 default，详情见工作流管理。

点击「提交」后，回到数据源列表页面，即可看见刚才添加的数据源：



如果数据源数量过多，可以通过筛选功能快速的找到某类数据源。

修改数据源

点击**数据源列表**页面的「编辑」按钮，可以修改指定数据源。**修改数据源**中的可修改项与**添加数据源**一致，可参考上面的一节的说明。

删除数据源

点击**数据源列表**页面的「删除」按钮，可以删除指定数据源。

模板管理

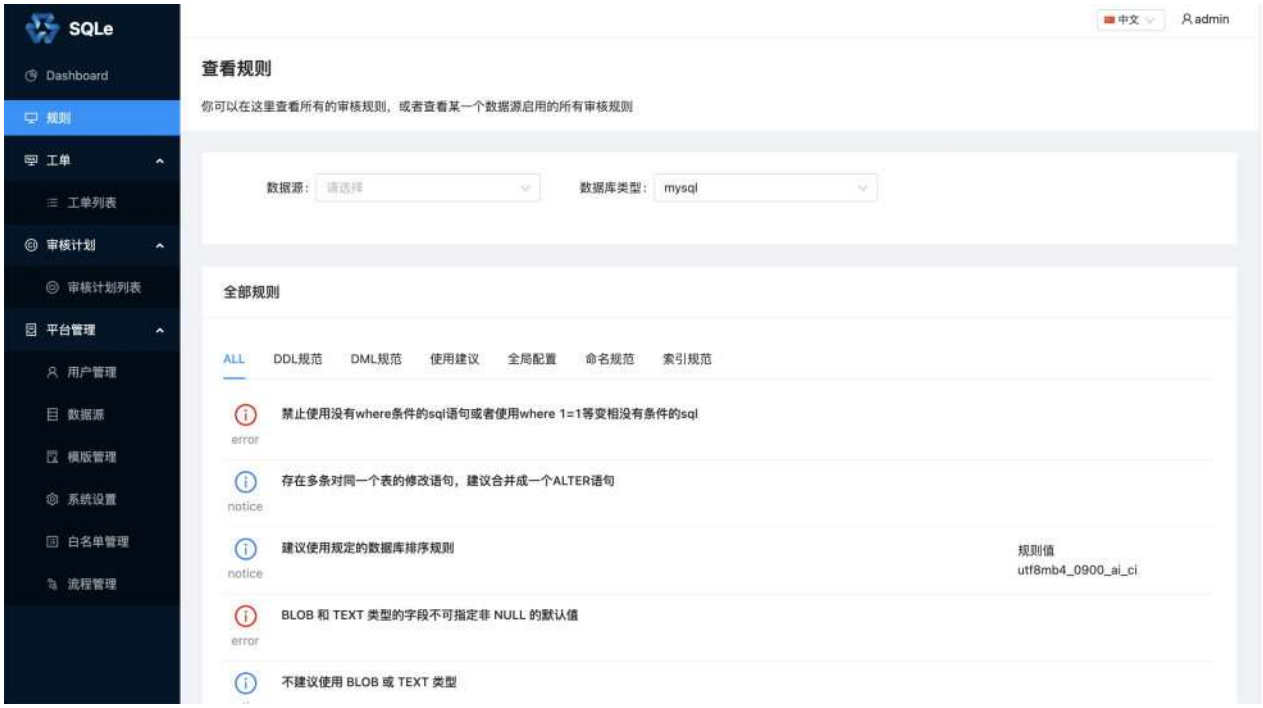
目录

规则模板管理

审核规则

审核规则是依据数据库特性经过长期实践总结出的一些规范。

SQLE 默认支持 70 余条 MySQL 审核规则。SQLE企业版额外支持部分企业版规则



审核规则有 4 个**规则等级**（严重级别由高到低）：

- Error
- Warn
- Notice
- Normal

部分规则还带有**规则值**：

- 规则「建议使用规定的数据库排序规则」 的规则值为 utf8mb4_0900_ai_ci；
- 规则「复合索引的列数量不建议超过阈值」 的规则值为 3；
- ...

针对不同的数据源，**规则等级**和**规则值**都是可以自定义的，详情见下一小节。

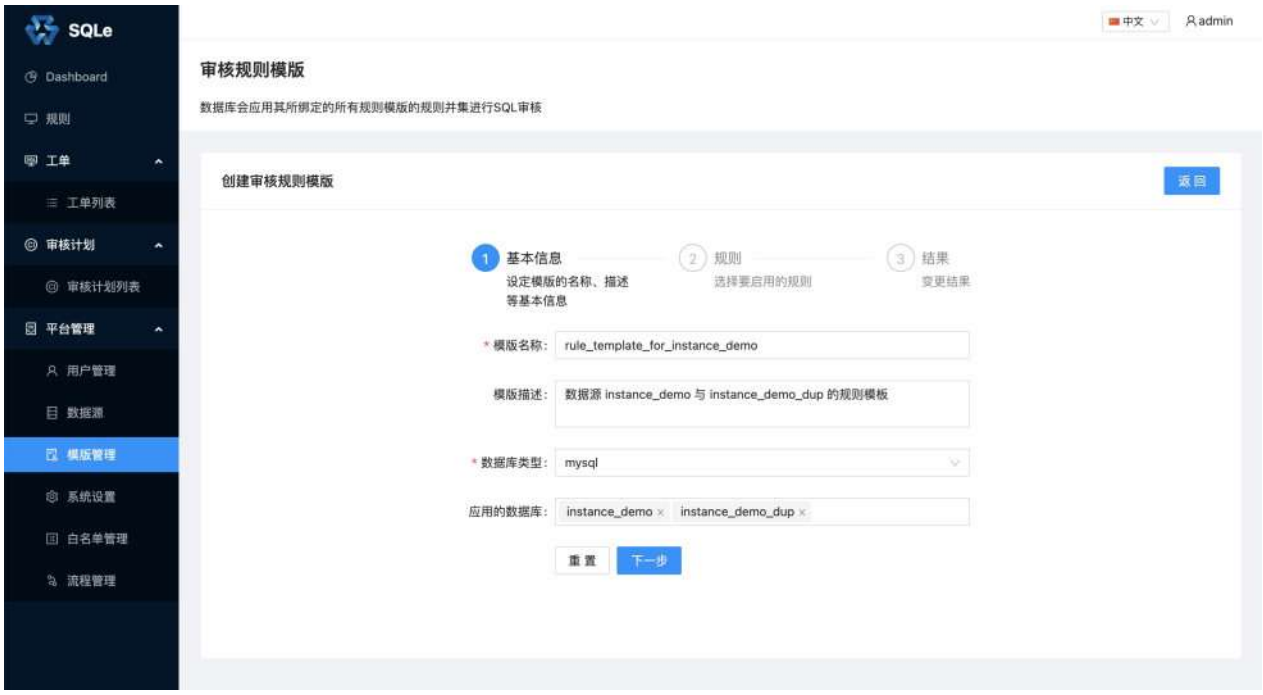
规则模板

不同数据源上的规范可能不同，即它们的审核规则集可能是不同的。在 SQLE 中，可以使用**审核模板**为不同的数据源配置不同的审核规则。

在左侧导航栏的「平台管理」中的「模板管理」页面中，点击「创建模板」

步骤 1：

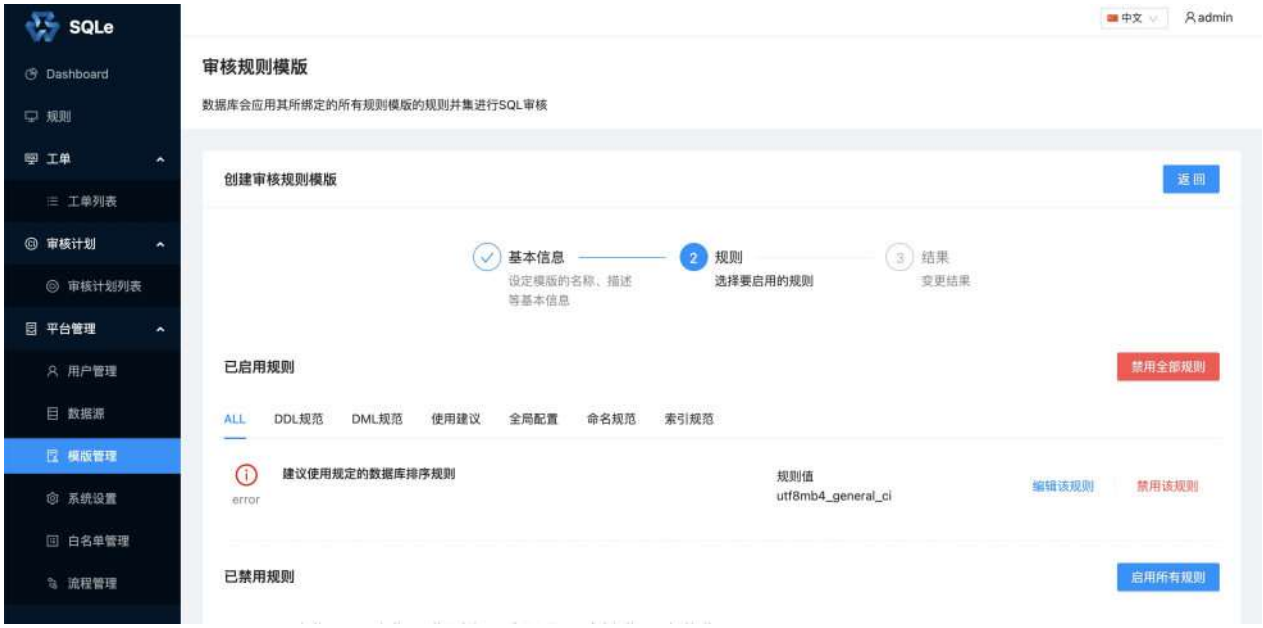
首先填写模板相关描述信息：



这里将新创建的规则模板同时应用到两个数据源上。

步骤 2：

然后选择需要启用的规则，并根据需求修改规则的**规则等级**和**规则值**：



这里仅启用一个审核规则，并且自定义了审核规则的**规则等级**和**规则值**（该规则默认的等级为Notice，值为utf8mb4_0900_ai_ci）。

流程模板管理（企业版功能）

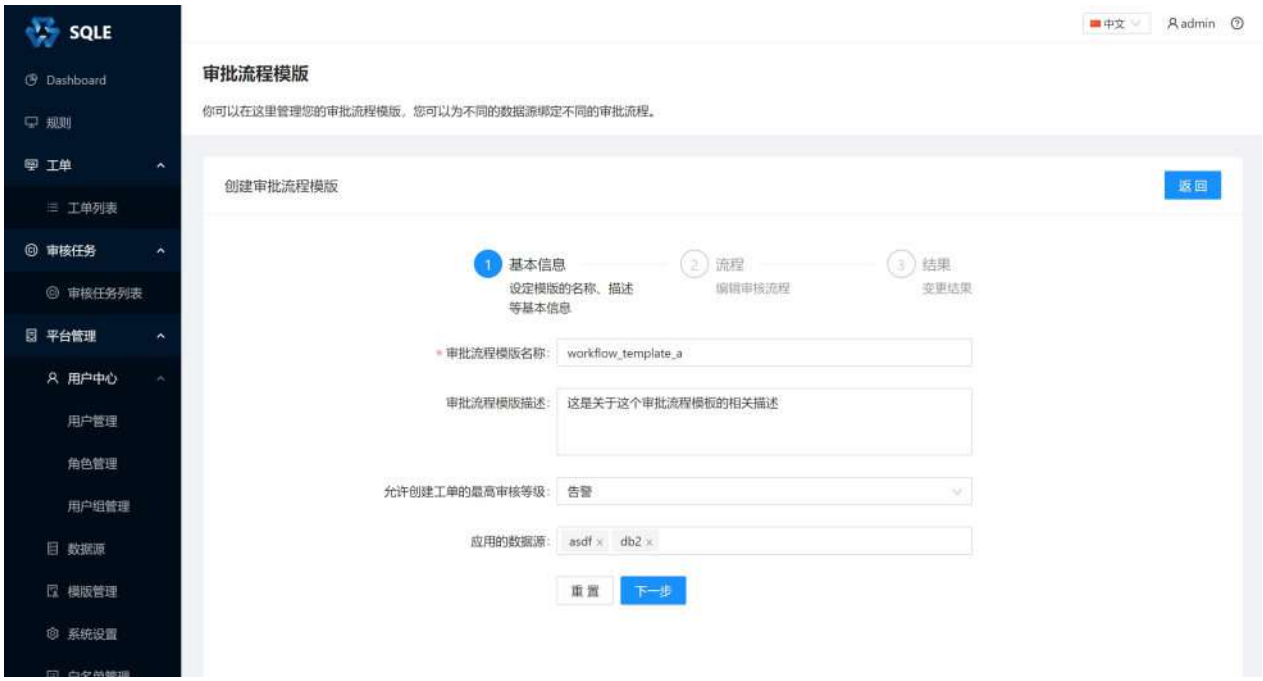
工作流要解决的主要问题是：为实现某个业务目标，在多个参与者之间按某种预定规则自动传递文档、信息或者任务。

SQLE 使用工作流来解决 SQL 上线的流程化问题。在 SQLE 中称为流程模板。

创建流程模板

在左侧导航栏的「平台管理」中的「流程管理」页面中，点击「创建流程模板」，填写流程模板相关信息。

步骤 1:



在这一步中, 指定了流程的基本信息

- 新建的流程名称为 `workflow_template_a` , 这个名称在流程模板中是唯一的
- 审核流程模板的描述
- 应用这个流程的工单必须所有SQL都没有触发大于 `告警` 级别的规则才可以提交工单
- `asdf` `db2` 这两个数据源将会使用此审核流程

步骤 2:

在这一步中，添加了两个审核步骤：

- 工单创建后，流转到 DBA初审
- DBA初审后，流转到部门领导复审

最后，部门领导复审后，流转到 DBA执行上线。

白名单管理（企业版功能）

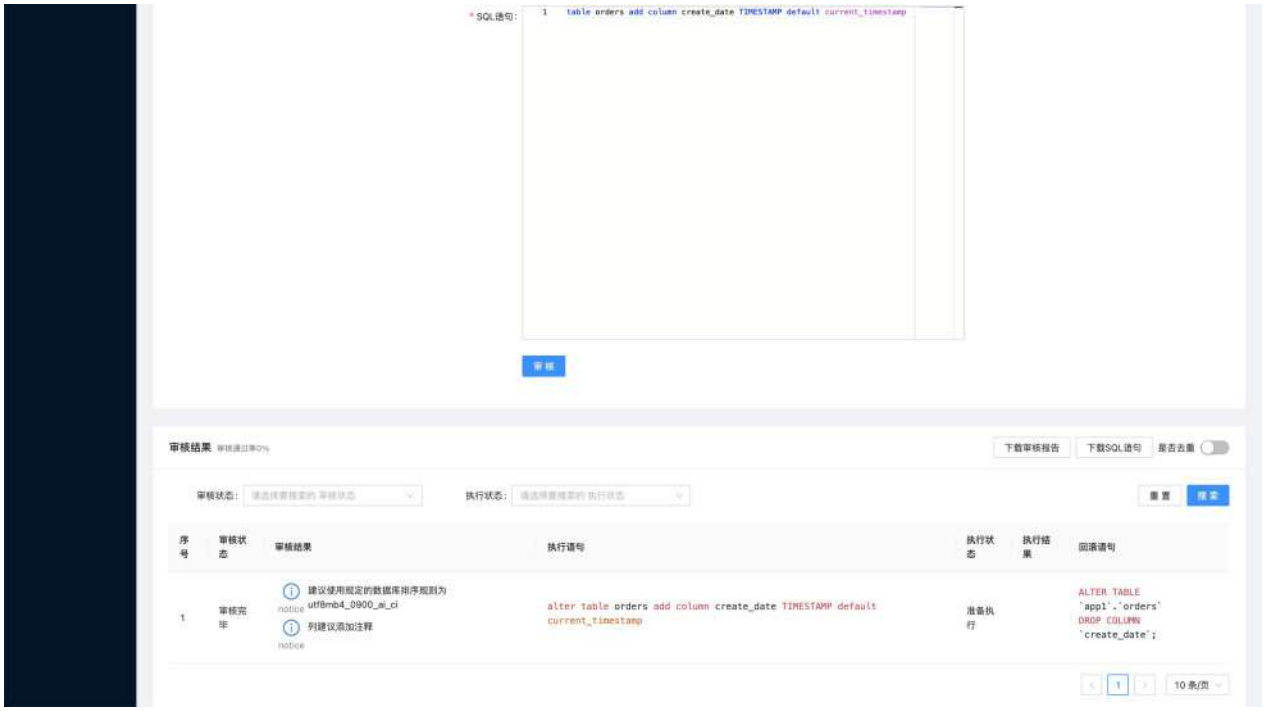
当我们已知一些 SQL 存在不规范之处，但又想忽略时，可以使用 SQL 白名单功能。

添加白名单

字符串匹配

在左侧导航栏的「平台管理」中的「白名单管理」页面中，点击「添加白名单」，填写 SQL 白名单相关信息。

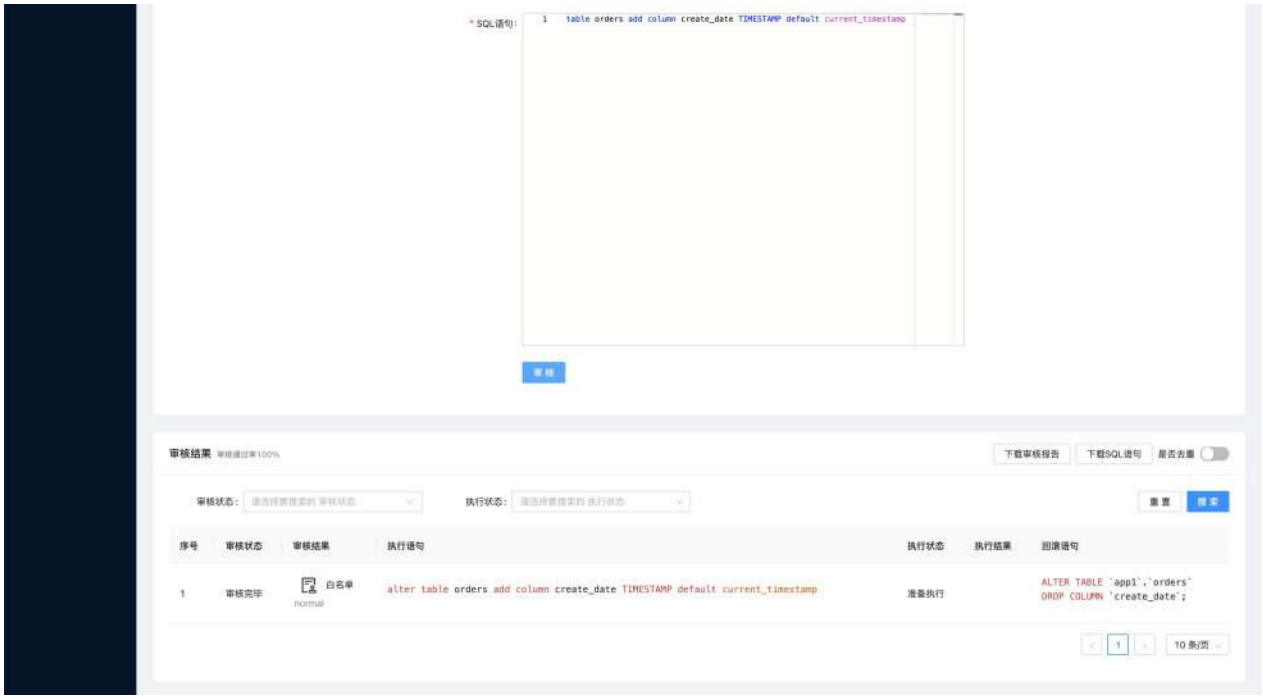
有一条 SQL：**alter table orders add column create_date TIMESTAMP default current_timestamp** 需要上线，在使用白名单前，提交审核：



审核通过率为 0% 无法上线。

假设因为某些原因无法修改 SQL，但是该 SQL 又必须上线。这时可以添加一条针对于该 SQL 的白名单：

再次提交审核：



审核通过率为 100% 可以提交工单上线。

指纹匹配

当我们想要忽略一类 SQL 时，又不想为每条 SQL 添加白名单时，可以在添加白名单时选择「sql指纹匹配」。

审核工单

目录

背景

很多公司通常设有 DBA（Database administrator） 这个职位，他们负责在系统上运行数据库，执行备份，执行安全策略等日常数据库运维的工作。在这之外，一项重要的工作是规范研发人员在应用程序中使用的 SQL 语句。

一个常见的场景是：产品在迭代过程中，某个表需要新增一个字段，这会涉及到改表操作。DBA依据数据库规范评估修改的合理性。如果评估没有问题，则将修改操作应用到线上数据库。如果公司对于这类操作比较严格，在上线前，可能还需要经过层层审批。所有上线流程中的人都同意后，再将修改操作应用到数据库。

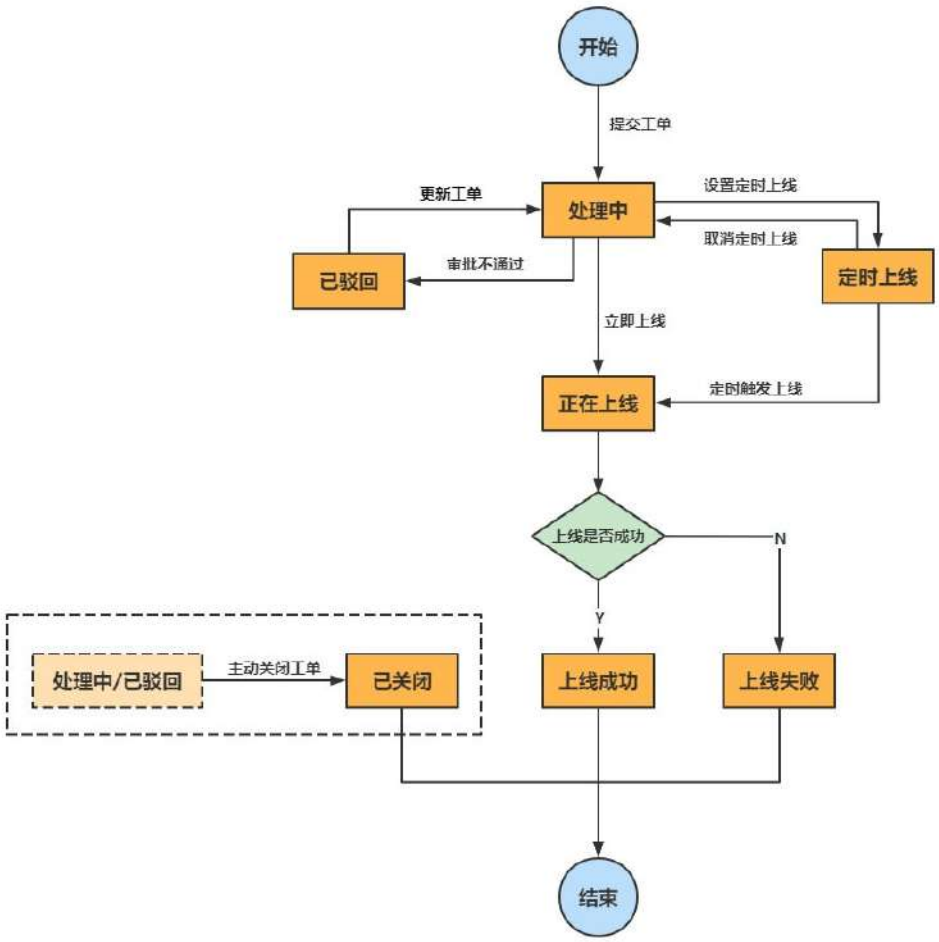
审核工单的出现主要是为了解决上述整个过程中的规范化、流程化的问题。

审核工单管理

在这一节中，会用一个具体的案例来逐一介绍「审核工单管理」中的各个功能模块。

工单状态

工单状态流转图



工单状态介绍

处理中

创建完的工单自动进入 `处理中` 状态，处理中代表工单正在流程审批中或者还未上线。

已驳回

1. `处理中` 的工单可以由每级的审批人或者上线人进行驳回，驳回后工单进入 `已驳回` 状态；
2. 此状态的工单可以由创建人或者超管用户进行修改并重新提交工单。

已关闭

1. `处理中` 或 `已驳回` 的工单可以由创建人或者超管用户在任何时刻关闭工单，工单流转结束无法再操作；

2. 已经进入上线流程的工单无法关闭。

定时上线

1. 审批通过的工单，会进入上线阶段，可以由上线人设置定时上线；
2. 已经设置定时上线的工单可以由上线人取消定时上线，此时工单回到 `处理中` ,可以再次进行驳回、定时上线、立即上线。

正在上线

1. 当上线人进行立即上线，工单会短暂的进入 `正在上线` 阶段，该阶段的时间取决于SQL是否上线完毕；
2. 定时上线的工单到时间会自动进行SQL上线，此时工单也会进入 `正在上线` 阶段；
3. SQL上线阻塞的工单会一直处于 `正在上线` 状态。

上线成功

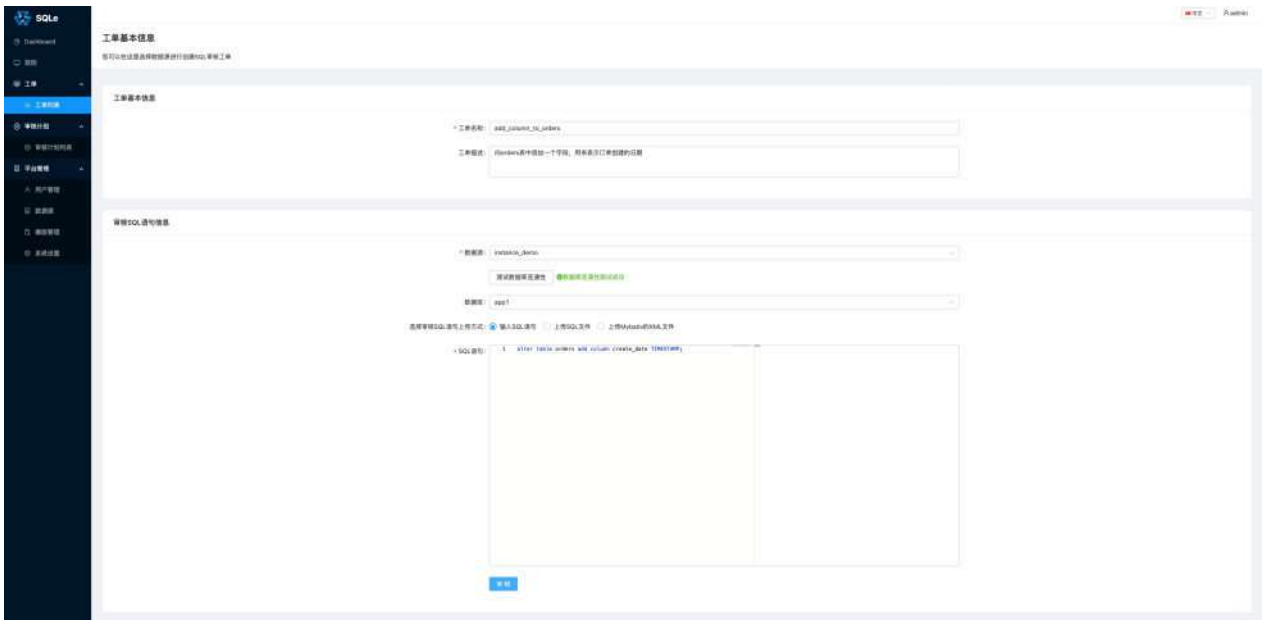
SQL上线成功的工单会进入 `上线成功` 状态，工单流转结束无法再操作。

上线失败

SQL上线失败的工单会进入 `上线失败` 状态，工单流转结束无法再操作。

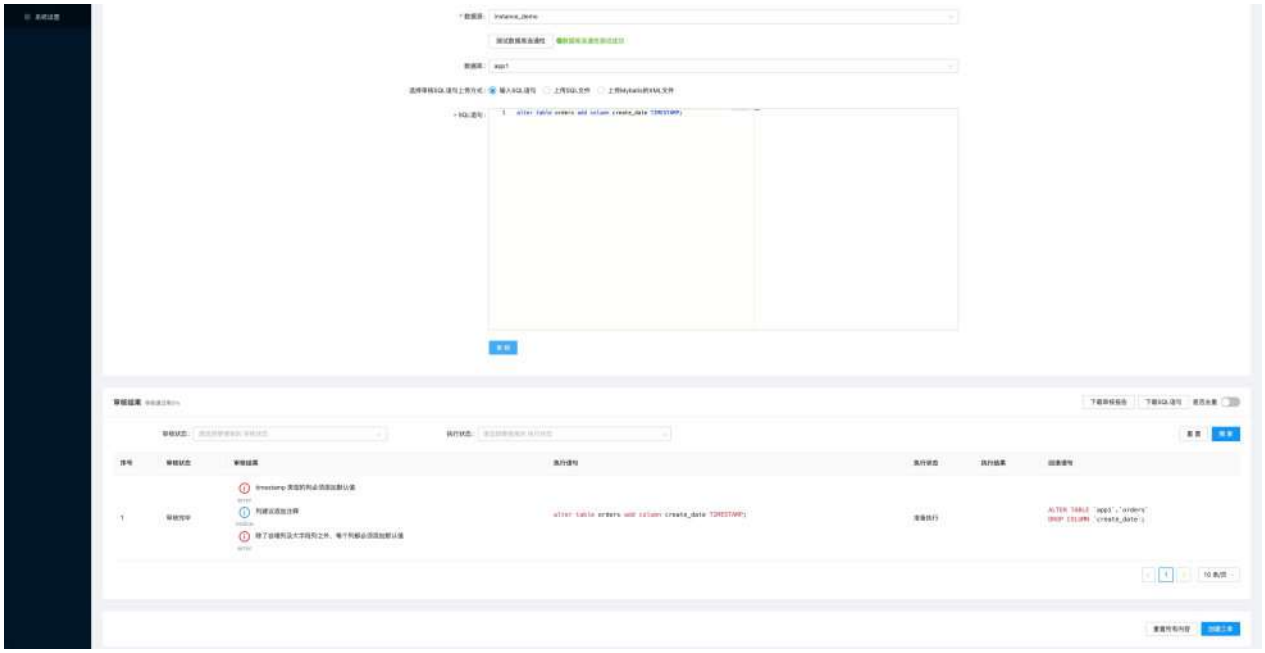
创建审核工单

在左侧导航栏的「工单」中的「工单列表」页面中，点击「创建工单」，审核工单相关信息，如下图：



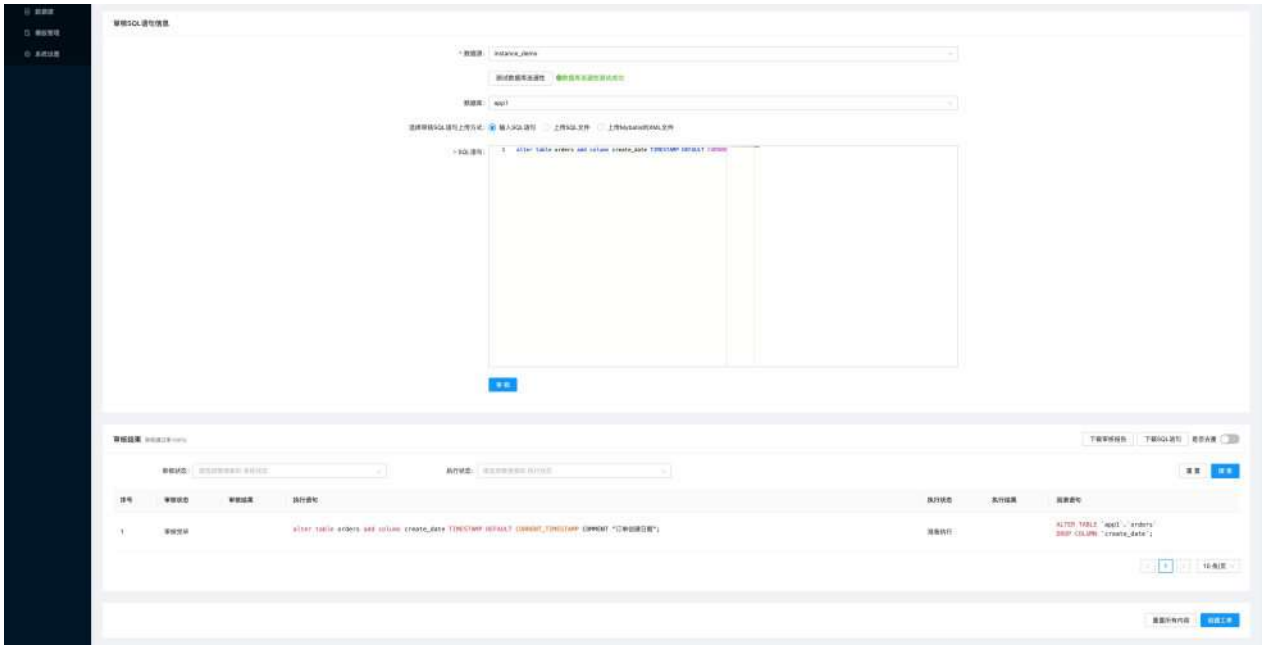
- 工单名称：该字段需要保证工单全局唯一性；
- 工单描述：略；
- 数据源：表示修改最终会应用到哪个数据源；
- 数据库：表示 orders 表所在的 schema，相当于执行 use 语句；
- SQL 语句：填写需要上线的 SQL 语句，该语句会被审核。

在确认需要上线的 SQL 语句后，点击页面下方的「审核」按钮：

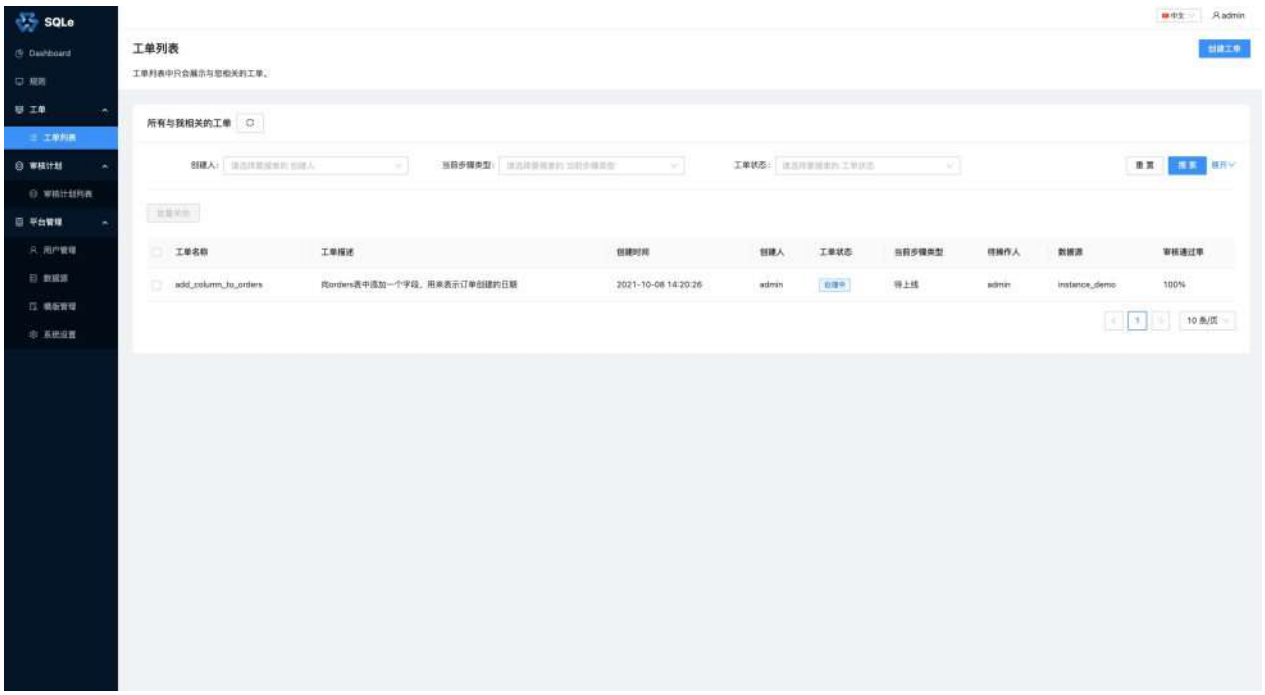


在**审核结果**列表中，可以看到被审核的 SQL 语句（**alter table orders add column create_date TIMESTAMP**）不符合该数据源上绑定的审核规则（或者说这条 SQL 触发了该数据源上的审核规则）。

按照**审核结果**给出的提示修改 SQL 语句，再次点击「审核」按钮：



可以看到，修改后的 SQL 已经完全符合规范，审核通过率也从 0% 变成了 100%。点击「创建工单」，回到工单列表，即可看到该工单显示为**处理中**：



请注意，能够创建工单有一个必要条件为，用户所关联的角色中至少有一个角色同时具备：

- 绑定了目标数据源；
- 拥有动作权限：「创建/编辑工单」。

为工单添加附加信息

在 `v1.2201.0` 往后的版本中, 工单的创建者, 修改者, 审核者可以为工单的任意一条SQL附加备注信息, 以便于记录特殊SQL的提交原因, 需要注意, 同一时间同一SQL只能存在一条备注, 后添加的备注将会覆盖之前的备注,备注默认为空

Online DDL

在 MySQL5.6 前，通过 Alter 语句直接修改表结构，如果表的数据量较大，可能造成长时间锁表的情况。为了解决这个问题，业界常用的做法是使用 Online DDL 的方式（即在修改表结构过程中，业务照常能够访问对应的表）。

通过集成优秀的开源 Online DDL 工具 gh-ost，SQLE 实现了 Online DDL 功能。

配置

在使用 SQLE 提供的 Online DDL 功能前，需要进行下述配置。

配置规则

Online DDL 对应的规则在**规则列表 -> 全局配置**中。对应的规则名为『**改表时，表空间超过指定大小(MB)时使用gh-ost上线**』。通过规则的 value 值，可以动态调整触发 Online DDL 的阈值（调整方式见规则模板管理）。

配置 gh-ost 参数

SQLE 根据 gh-ost 的官方推荐，使用了如下默认配置（相应配置说明见官方文档）：

```
mysql_timeout=0.0
assume_master_host=""
master_user=""
master_password=""
exact_rowcount=false
concurrent_rowcount=true
allow_on_master=false
allow_master_master=false
allow_nullable_unique_key=false
approve_renamed_columns=false
skip_renamed_columns=false
tungsten=false
discard_foreign_keys=false
skip_foreign_key_checks=false
;skip_strict_mode=false //fixme
aliyun_rds=false
gcp=false
azure=false
test_on_replica=false
test_on_replica_skip_replica_stop=false
migrate_on_replica=false
ok_to_drop_table=false
initially_drop_old_table=false
initially_drop_ghost_table=false
timestamp_old_table=false
cut_over=""
force_named_cut_over=false
force_named_panic=false
switch_to_rbr=false
assume_rbr=false
cut_over_exponential_backoff=false
exponential_backoff_max_interval=64
chunk_size=1000
dml_batch_size=10
default_retries=120
cut_over_lock_timeout_seconds=3
nice_ratio=0
max_lag_millis=1500
throttle_control_replicas=""
throttle_query=""
throttle_http=""
ignore_http_errors=false
heartbeat_interval_millis=100
throttle_flag_file=""
throttle_additional_flag_file="/tmp/gh-ost.throttle"
postpone_cut_over_flag_file=""
panic_flag_file=""
initially_drop_socket_file=false
serve_socket_file="/tmp/gh-ost.${库名}.${表名}.sock"
serve_tcp_port=0
replica_server_id=${内部生成随机数防止重复}
max_load="Threads_running=80,Threads_connected=1000"
critical_load="Threads_running=80,Threads_connected=1000"
critical_load_interval_millis=0
critical_load_hibernate_seconds=0
force_table_names=""
```

可以根据自己的需求自定义配置。配置文件的位置在 `/opt/sqle/etc/gh-ost.ini` 。注意单词间使用下划线分割。

其他配置

1. 开启数据源 binlog，并设置 binlog 格式为 ROW
2. 如果数据源为单实例或者从实例，需要在配置文件里指定**allow_on_master=true**
3. 数据源的用户权限至少需要：
    i. ALTER, CREATE, DELETE, DROP, INDEX, INSERT, LOCK TABLES, SELECT, TRIGGER, UPDATE ${迁移表所在的数据库}.*
    ii. SUPER, REPLICATION SLAVE on *.* 或者 REPLICATION CLIENT, REPLICATION SLAVE on *.*

限制

当然 gh-ost 也有一些使用限制，请参考官方文档。

诊断

如果一条 SQL 正在通过 Online DDL 上线，通常可以通过以下方式诊断上线过程。

SQLE 日志

通过 SQLE 的日志文件可以看到 gh-ost 执行完整流程，如下面是 dry-run 流程的部分输出日志：

```
time="2021-10-26T11:01:27+08:00" level=info msg="dry-run gh-ost" task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="--alter: ADD COLUMN `c10` INT" alter="alter table sbtest1 add column c10 int;
" host=127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Migrating `test`.`sbtest1`" alter="alter table sbtest1 add column c10 int;" h
ost=127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
...
time="2021-10-26T11:01:27+08:00" level=info msg="Done migrating `test`.`sbtest1`" alter="alter table sbtest1 add column c10 in
t;" host=127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Removing socket file: /tmp/gh-ost.test.sbtest1.sock" alter="alter table sbtes
t1 add column c10 int;" host=127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Tearing down inspector" alter="alter table sbtest1 add column c10 int;" host=
127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Tearing down applier" alter="alter table sbtest1 add column c10 int;" host=12
7.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Tearing down streamer" alter="alter table sbtest1 add column c10 int;" host=1
27.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="Tearing down throttler" alter="alter table sbtest1 add column c10 int;" host=
127.0.0.1 onlineddl=gh-ost port=4406 task_id=12 thread_id=50N
time="2021-10-26T11:01:27+08:00" level=info msg="dry-run OK!" task_id=12 thread_id=50N
```

Systemd 日志

通过 Systemd 的日志管理器可以看到 gh-ost 执行迁移的进度：

```
journalctl -u sqled
```

索引优化

一、背景

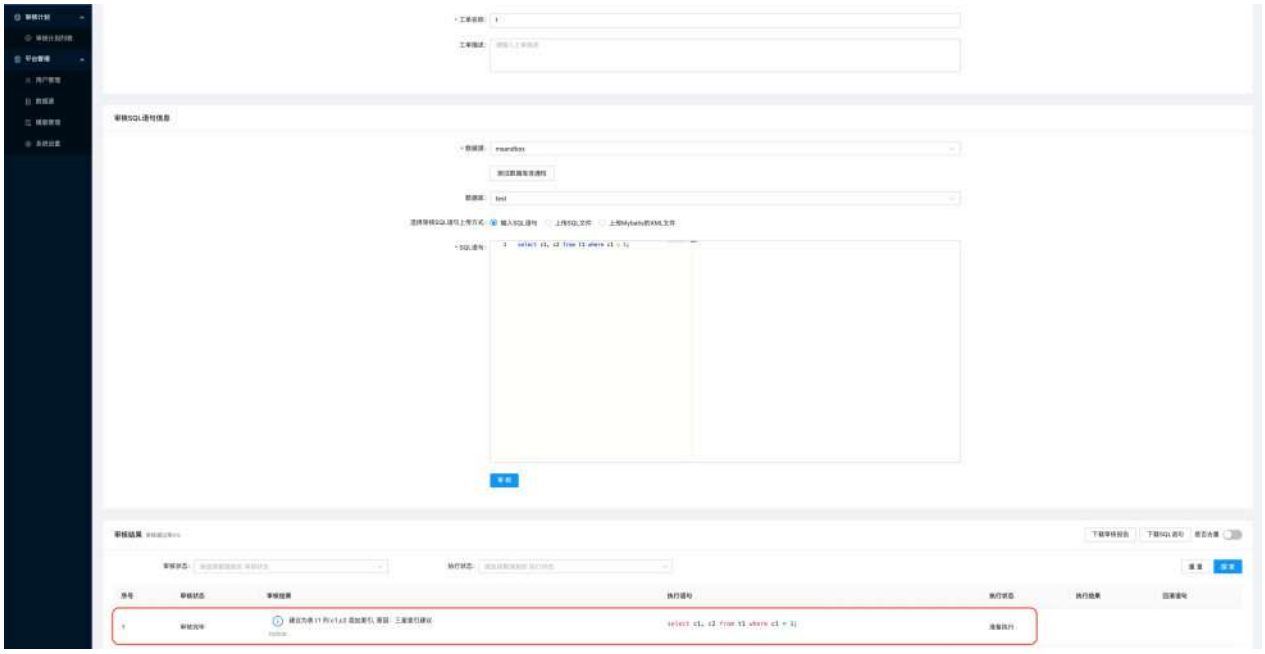
大多数生产环境上对数据库的操作都是读多写少。对数据库的查询操作经常容易造成应用的性能瓶颈。其中大概率原因是以下两个：

1. SQL 语句不合理，导致查询效率低下；
2. 数据库索引不合理，导致查询效率低下。

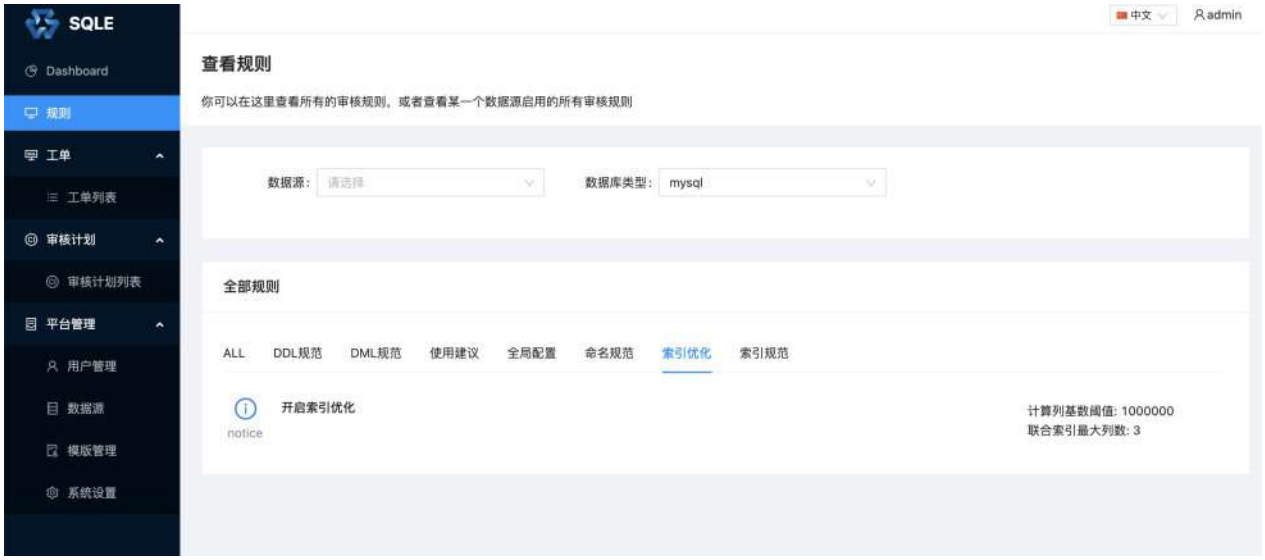
针对第二个问题，我们可以通过优化数据库索引来提高查询效率。在 SQLE 中提供了索引优化的功能。

二、使用方式

对 SQL 的索引优化结果通过 SQL 审核页面展示出来。如下图：



在 SQLE 中索引优化默认是关闭状态。通过配置规则开启索引优化：



这个规则上有以下参数可以配置：

参数 1：联合索引最大列数

SQLE 给出的索引建议可能是多列联合索引。当联合索引的列数太多，可能会增加数据库对索引的维护成本。所以通过一个配置来控制联合索引的最大列数，默认值是 3。

参数 2：计算列基数阈值

在建立联合索引时，字段的顺序非常重要。将基数较高的列放到前面，可以走索引的过程中筛选掉更多的不需要的记录。

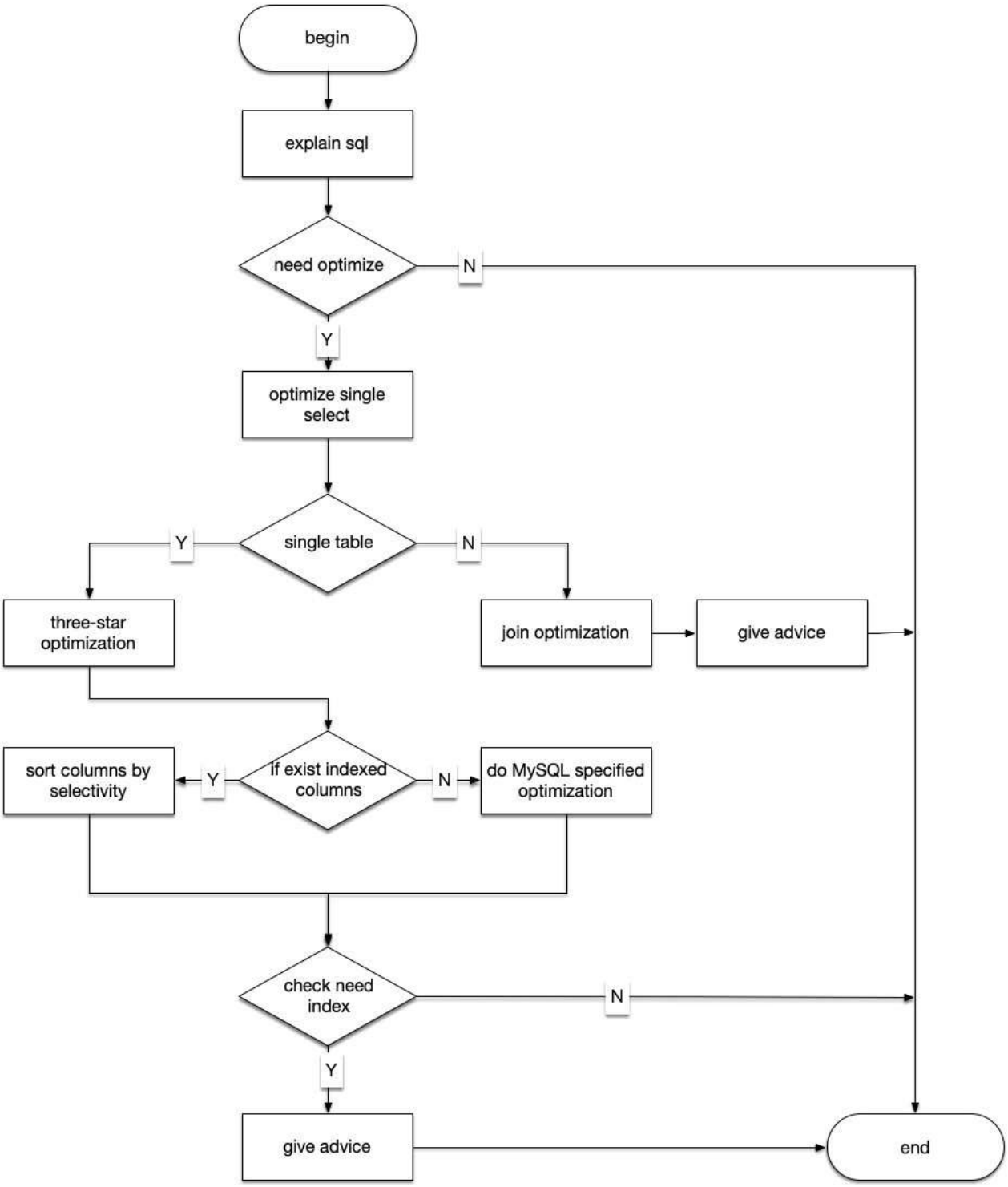
但是完全按照基数顺序建立联合索引，在一些场景下可能也不那么完美：

1. 表的记录较多，计算基数对性能影响较大；
2. 表的数据分布可能经常发生变化，列的基数大小有可能因此发生变化。

所以 SQLE 给出了一个阈值参数，来控制联合索引是否按照基数从大到小的顺序建立，默认值是 1000000。

三、优化策略

整个索引优化的流程图如下：



下面简单介绍一下部分的流程模块里做的事情：

1. explain SQL

在索引优化的过程中，会借助 MySQL 的 EXPLAIN 命令，来查询 SQL 执行计划。通过执行计划，可以更准确的知道当前 SQL 的索引使用情况。

2. need optimize

无论多么复杂的查询，最终都会被解析成一个个单表查询。一个单表查询对应执行计划的一行记录。通过对每一行记录的分析，可以知道是否需要对表进行索引优化。

通过判断单表的访问方式，可以判断是否需要对表进行索引优化。单表的访问方式在执行计划记录中对应 `type` 列。`type` 的具体值就不一一介绍了，详情可以参考文档 EXPLAIN Output Format。

SQLE 会对访问方式为 `ALL` 和 `index` 的表做索引优化。

3. optimize single select

对于复杂的如带有子查询的 SQL 语句，SQLE 会对每一个子查询进行索引优化。

4. three-star optimization/join optimization

对于多表的 select 语句，SQLE 会根据 explain 的结果，对被驱动表进行索引优化，优化方式是建议在被驱动表的关联字段上建立索引。

对于单表的 select 语句，SQLE 则参考「Relational database index design and the optimizers」一书中的三星索引算法，尽量给出三星索引建议。

5. if exist indexes columns

通过前面的步骤，如果给出了一个索引建议。例如建议 c1 上添加索引。这时还需要判断：

1. 数据库中是否存在该索引
2. 数据库中是否存在以 c1 为前缀的联合索引

如果满足上面两个条件之一，则放弃该索引建议。

审核任务

目录

审核任务介绍

一、背景

在审核工单中我们介绍了如何通过 SQLE 进行 SQL 审核并上线的流程。在这流程中，通常是审核一些 DDL，如建库（create database）、建表（create table）和改表（alter table）等语句。 审核工单管理，主要解决 SQL 上线的规范化流程化的问题，它能够帮助 DBA自动化处理整个 SQL 上线过程中一些重复繁琐的工作。

审核工单管理也有它的局限性。

- 第一，通常**审核工单管理** 中流转的是 DDL 一类的 SQL，它们的上线通常是一次性操作。DDL 上线后，通常还会有业务 SQL（通常是 DML）访问数据库。这时可能会遇到一些执行效率较低的业务 SQL 造成数据库的性能问题。这类 SQL 是**审核工单管理**无法覆盖到的。
- 第二，通常在使用**审核工单管理**审核需要上线的 SQL 时，项目已经临近发版，功能都已经实现完毕。如果这时审核出 SQL 存在一些问题，是否修复这些问题，可能会收到很多因素的影响（如 SQL 问题的影响面大小，项目发版的紧急程度等）。虽然这类 SQL **审核工单管理**可以覆盖到，但因为外部因素，会导致问题 SQL 的存在。

SQLE 使用**审核任务**来解决上面的两个问题。

审核任务是基于 Cron 的 SQL 自动审核系统。一个**审核任务**的成功运行需要两方参与，分别是 SQLE Server 和 **Scanner**。在 SQLE Server 中创建审核任务。Scanner 将 SQL 上传到 SQLE Server 对应的审核任务的 **SQL 池**，SQLE 根据配置对 SQL 池触发审核，并生成**审核报告**。

二、审核任务介绍

分类

类型	支持的数据库	SQL采集模式
库表元数据	MySQL	SQLE 自动抓取
TopSQL	Oracle	SQLE 自动抓取
慢日志	MySQL	Scanner 抓取
Mybatis扫描	ALL	Scanner 抓取
应用程序SQL抓取	ALL	OpenAPI 推送
自定义	ALL	OpenAPI 推送

SQL采集模式

1. SQLE 自动抓取

由 SQLE 后台服务定期抓取，创建完审核任务后无需再做其他操作，即可再界面看到已经采集的SQL并进行审核。

2. Scanner 抓取

scanner 是SQLE支持的实现对特定场景进行SQL采集的一个客户端工具，通过调用 SQLE 的OpenAPI将SQL推送到SQLE，例如：scanner支持扫描MySQL慢日志 并实时解析文件中的慢SQL将SQL推送到SQLE。具体的使用参考：Scanner 使用说明

3. OpenAPI 推送

SQLE 对外提供标准的OpenAPI接口，任何人都可以通过调用接口来将特定场景的SQL收集并推送到SQLE，来使用审核任务的审核能力。用户可以自行实现scanner（可以是任何形式，bash脚本、python程序等）将SQL推送到审核任务类型为“自定义”的审核任务中并进行审核。

审核任务管理

创建审核任务

参数说明

- 审核任务名称：略；
- 数据源名称：填写需要审核的目标库，若不填则仅进行静态分析不会连库；
- 审核任务类型：参考审核任务介绍的审核任务分类；
- 任务审核周期：配置的是SQLE对审核任务进行自动审核的周期。

特殊参数说明

不同的审核任务类型可能会配置不同的参数，选中对应的审核类型后会自动展示

库表元数据

- 采集周期（分钟）：默认60分钟，代表SQLE server间隔多久采集一次库表元数据；
- 是否采集视图信息：开启后会采集视图。

Top SQL

- 采集周期（分钟）: 默认60分钟，代表SQLE server间隔多久采集一次Oracle Top SQL；
- top_n: 展示前多少条记录，默认3;
- V$SQLAREA中的排序字段：采用那个字段作为Top SQL 的排序字段
  - "executions"： 按总执行次数排序
  - "elapsed_time"：按总执行时间排序（默认值）
  - "cpu_time"：按CPU消耗时间排序

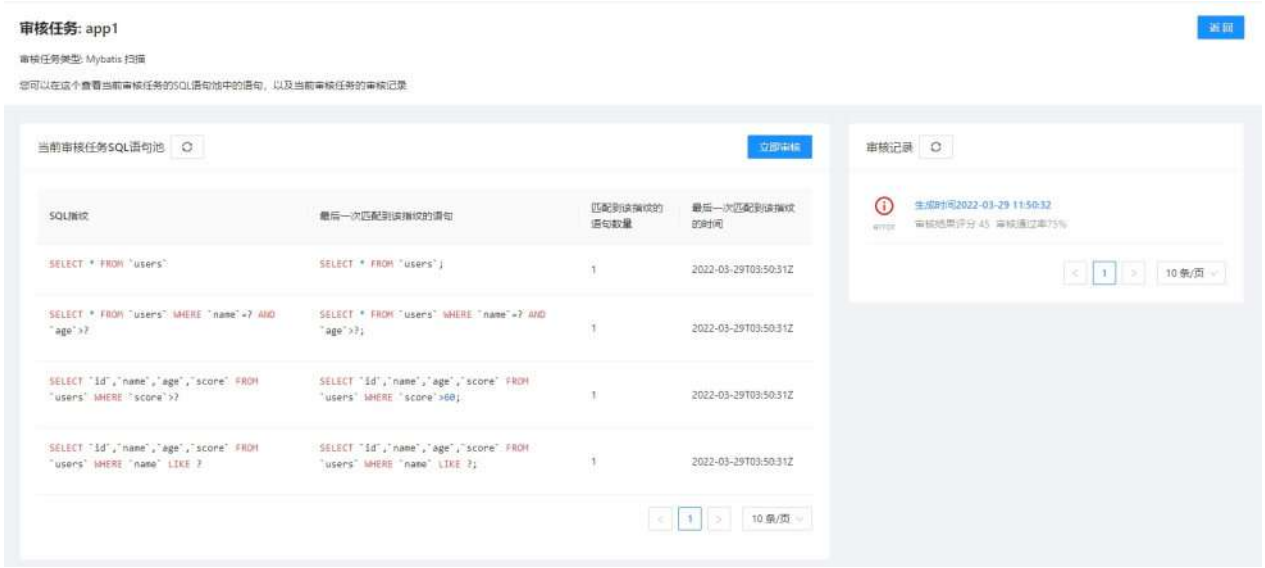
- "disk_reads"：按物理读次数排序
- "buffer_gets"：按逻辑读次数排序
- "user_io_wait_time"：按 IO 等待时间排序

审核任务列表

列表页面展示了当前 SQLE 中正在运行的所有审核任务。点击审核任务的**编辑**按钮修改这个审核任务。另外可以看到**访问凭证**一列，这是给 Scanner 使用的，使用方式请参考Scanner一章。

审核任务详情

点击**审核任务**，进入审核任务的详情页面：



在**审核任务**的详情页面，我们可以看到它的 SQL 池列表和对应产生的审核报告列表。上图中展示的 SQL 池列表中的 SQL 是通过 MyBatis Scanner 扫描代码仓库中文件得到的。

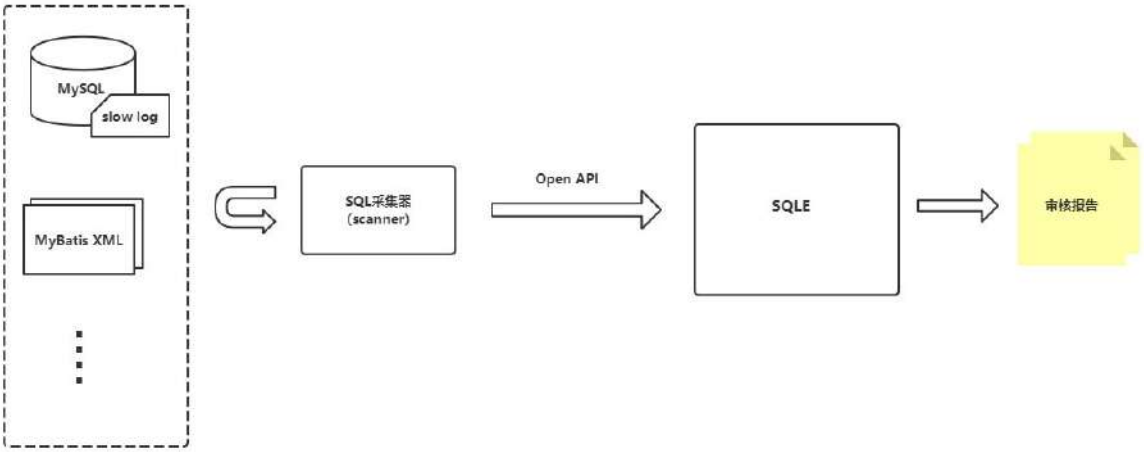
审核任务审核报告

点击**审核记录**，可以查看审核报告

Scanner 使用说明

一、简介

Scanner 是**审核任务管理**中负责解析并上传 SQL 的组件。 不同场景下的 **Scanner** 会有不同的解析行为，但它们最终都需要通过统一的 API 接口将 SQL 上传到 SQLE。由于 Scanner 是**审核任务**功能对应的组件，所以在使用前需要创建审核任务，具体操作步骤见审核任务管理。



二、Scanner 存放位置

Scanner 打包在 SQLE RPM 中。在部署完 SQLE Server后（部署方式见安装部署），Scanner 的二进制放在 `${SQLE的工作目录}/bin` 目录下。如下：

```
[root@sqle-server bin]# pwd
/opt/sqle/bin
[root@sqle-server bin]# ll
total 62816
-rwxr-x--- 1 actiontech-universe actiontech 25064378 Sep 27 13:16 scannerd
-rwxr-x--- 1 actiontech-universe actiontech 39254854 Sep 27 13:16 sqled
```

三、Scanner 介绍

1. MyBatis Scanner

概述

MyBatis Scanner 通过指定代码目录，扫描并解析目录中的 MyBatis XML 文件得到 SQL 语句。**MyBatis Scanner** 将解析出的 SQL 上传至 SQLE Server 后，触发审核并得到审核结果。如果审核结果中包含 Error 级别的错误（指触发了 Error 级别的审核规则），则将退出码（Exit Code）置为非 0。

参数说明

```
[root@sqle-server bin]# ./scannerd mybatis --help
Parse MyBatis XML file

Usage:
  SQLE mybatis [flags]

Flags:
  -D, --dir string   xml directory
  -h, --help         help for mybatis

Global Flags:
  -H, --host string    sqle host (default "127.0.0.1")
  -N, --name string    audit plan name
  -P, --port string    sqle port (default "10000")
  -A, --token string   sqle token
```

- host：SQLE Server 所在的主机 IP 地址（默认是当前主机）
- port：SQLE Server 提供 HTTP 服务的端口（默认是 10000）
- name：审核任务名（表示 Scanner 将 SQL 上传至哪个审核任务的 **SQL 池**）
- token：审核任务上传凭证（具体值可到审核任务列表页中**访问凭证**列中获取）
- dir: Mybatis XML 对应的文件夹

常见场景

通过集成到 CI/CD 进行开发阶段 SQL 审核， 通过持续的审核代码仓库中的 SQL 以及变更，可以提早发现问题。

2. SlowQuery Scanner（企业版功能）

概述

SlowQuery Scanner 通过指定慢日志文件，扫描并解析文件中的慢 SQL 语句，并定时将解析出的 SQL 上传至 SQLe Server。在 SQLe Server 侧，通过审核任务中配置的 Cron 定时（或人工）触发审核。 **SlowQuery Scanner** 是一个常驻进程，它会持续的监控 Scanner 启动时指定的慢日志文件，一旦产生新的慢 SQL，它会增量解析并上传这些 SQL，可以将scanner启动为后台进程。

参数说明

```
[root@sqle-server bin]# ./scannerd slowquery --help
Parse slow query

Usage:
  SQLE slowquery [flags]

Flags:
  -h, --help              help for slowquery
      --log-file string   log file absolute path

Global Flags:
  -H, --host string    sqle host (default "127.0.0.1")
  -N, --name string    audit plan name
  -P, --port string    sqle port (default "10000")
  -A, --token string   sqle token
```

- host：SQLE Server 所在的主机 IP 地址（默认是当前主机）
- port：SQLE Server 提供 HTTP 服务的端口（默认是 10000）
- name：审核任务名（表示 Scanner 将 SQL 上传至哪个审核任务的 **SQL 池**）
- token：审核任务上传凭证（具体值可到审核任务列表页中**访问凭证**列中获取）
- log-file：慢日志文件的绝对路径

数据库审核插件

目录

背景

将 SQLE 的数据库审核插件化主要有以下目的：

- 将审核流程（业务）的代码和具体审核实现的代码进行分离，支持多数据源类型的审核
- 审核插件在满足基本规范情况下，与 SQLE 独立开发

概述

SQLE 使用 go-plugin 来实现审核插件化。插件只需实现两个接口，即可实现与 SQLE 主进程通信（详情见审核插件开发）。

数据库审核插件使用

一、配置插件

创建插件二进制目录，通常创建在 SQLE 的工作目录下：

```
bash-4.2$ ls -l
total 24
drwxr-x--- 2 actiontech-universe actiontech 4096 Oct  8 09:30 bin
drwxrwx--- 1 actiontech-universe actiontech 4096 Oct 12 13:40 etc
drwxr-x--- 1 actiontech-universe actiontech 4096 Oct 12 13:40 logs
drwxr-xr-x 1 actiontech-universe actiontech 4096 Oct  8 09:30 plugins # 创建插件目录
drwxr-x--- 2 actiontech-universe actiontech 4096 Oct  8 09:30 scripts
drwxr-x--- 3 actiontech-universe actiontech 4096 Oct  8 09:30 ui
```

修改配置文件 sqle.yml：

```
server:
  sqle_config:
    server_port: 10000
    auto_migrate_table: true
    debug_log: false
    log_path: './logs'
    plugin_path: './plugins'   # 此处填写插件目录，也可以填写绝对路径
  db_config:
    mysql_cnf:
      mysql_host: '127.0.0.1'
      mysql_port: '3306'
      mysql_user: 'root'
      mysql_password: 'mysqlpass'
      mysql_schema: 'sqle'
```

二、集成插件

将插件的二进制文件放在 `一、配置插件` 中创建的目录 `plugins/` 内，其中插件的可以由第三方提供或者自行编译，参考：sqle-pg-plugin

ps：在重启 SQLE 前，需要确保 SQLE 的运行用户拥有插件二进制的执行权限，如果没有，可以通过以下命令来设置：

```
chmod +x /opt/sqle/plugins/sqle-pg-plugin
```

三、重启 SQLE Server

rpm 安装

```
systemctl restart sqled
```

docker

```
docker restart sqle-server
```

四、确认插件生效

通过日志

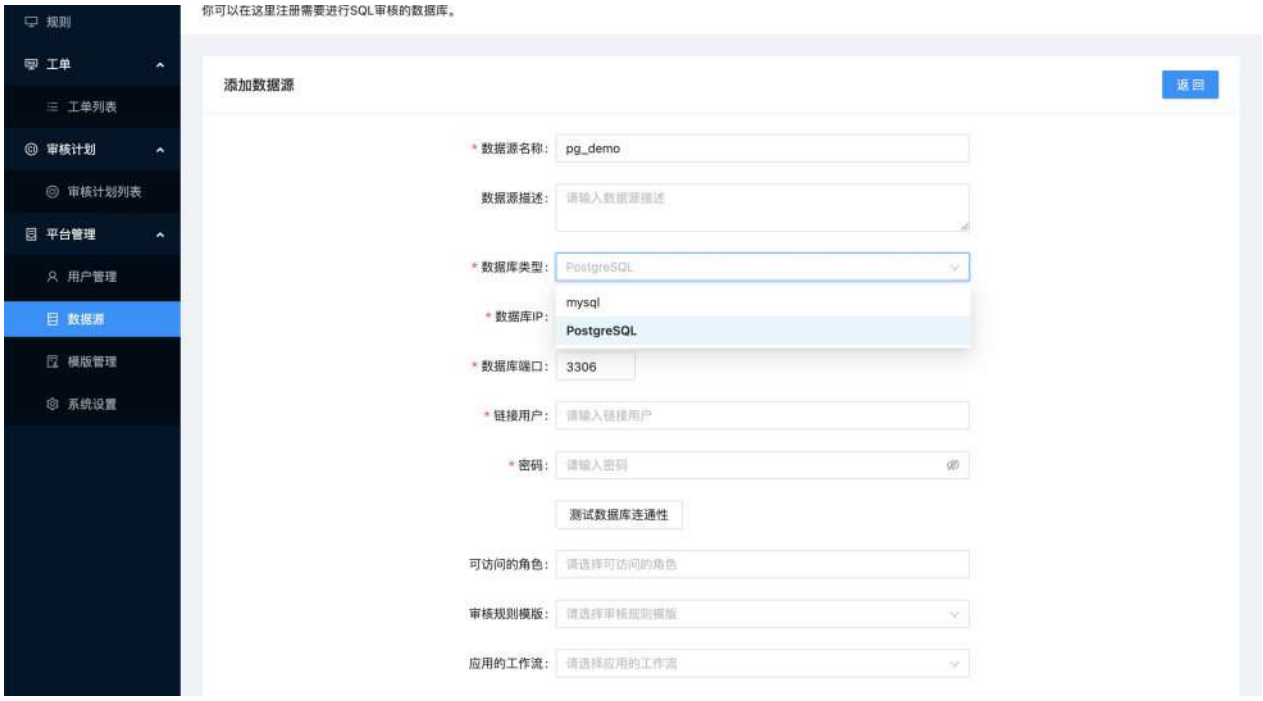
重启后，若成功加载插件，日志中会打印当前加载的审核插件：

```
time="2021-10-12T14:06:42+08:00" level=info msg="starting sqled server"
time="2021-10-12T14:06:42+08:00" level=info msg="plugin inited" plugin_name=PostgreSQL
...
time="2021-10-12T14:06:43+08:00" level=info msg="starting http server on :10000"
```

通过页面

- 查看 SQLE 规则页面，PostgreSQL 插件支持的审核规则已经加载到 SQLE Server：

- 添加数据源，显示已经可以添加 PostgreSQL 类型的数据源。



数据库审核插件开发

在本篇文档中，会介绍如何开发一个数据库审核插件，分为**快速开始部分**和**详细部分**。如果你是一个对 Go 语言不太了解的人，可以先通过**快速开始部分**的文档，实现一个简单的数据库审核插件。当你对插件开发有了一定的了解之后，可以通过**详细部分**的文档，通过更多自定义的方式，实现更加复杂的数据库审核插件。

快速开始部分

一、前置

1. 语法

由于 SQLE 是一个用 Go 语言开发的开源项目，如果你对 Go 语言完全不了解，需要先了解 Go 语言的基础语法，建议使用官方的 Go 语言快速开始文档（Go Tour 或者 Go 语言之旅），如果你已经了解了 Go 语言的基础语法，可以直接跳过本部分。

2. 包管理

SQLE 插件是一个独立于 SQLE 的进程，所以编写插件的方式与开发一个新的 Go 语言项目并没有什么差异。Go 语言项目使用 Go Modules 来管理包，所以在开始之前你需要先了解一下 Go Modules 的使用方法。建议参考这篇文档，文档中大致介绍了如果开始一个新的 Go 语言项目，并通过 Go Modules 的方式来调用其他项目的包，如果你已经了解了 Go Modules 的使用方法，可以直接跳过本部分。

3. Go 语言项目构建

二、编写插件

这一小节会假设你已经创建了一个由 Go Modules 管理的审核插件项目。下面开始介绍插件核心代码的开发。

1. 插件库介绍

SQLE 为了方便插件的开发，在自身的插件层之上做了一层封装（Adaptor），插件开发者可以使用这个封装库来快速的开发一个数据库审核插件。

在开发之前，你需要先引入 SQLE：

```
go get github.com/actiontech/sqle@v1.2111.0-pre3 # 此版本为该文档编辑时的最新版本
```

选择 SQLE 中两个库插件相关的库：

1. github.com/actiontech/sqle/sqle/driver
2. github.com/actiontech/sqle/sqle/pkg/driver

第一个库定义了插件规则的结构体，在你编写插件规则时需要用到这个库。

第二个库中实现了一些默认的插件，在引入这个库后，只需要实现相应的规则与规则处理函数即可。其中三个默认的插件为：

1. PostgreSQL
2. Oracle
3. SQL Server

如果这三个默认的插件不能满足需求，可以自己实现一个插件，参考本篇文档「**自定义插件**」小节。

2. 选择插件

下面假设你想要实现一个 SQL Server 的审核插件。在 main 函数中创建一个空的 SQL Server 审核插件，这时你的 main 文件应该是这样的：

```
package main

import (
    "github.com/actiontech/sqle/sqle/driver"
    adaptor "github.com/actiontech/sqle/sqle/pkg/driver"
)

func main() {
    plugin := adaptor.NewAdaptor(&adaptor.MssqlDialector{})
}
```

3. 编写插件规则与规则处理函数

假设你需要实现一个规则，该规则检查 SQL 是否使用了 select *。

定义规则如下：

```
Rule{
    Name:     "aviod_select_all_column", # 规则ID，该值会与插件类型一起作为这条规则在 SQLE 的唯一标识
    Desc:     "避免查询所有的列", # 规则描述
    Category: "DQL规范", # 规则分类，用于分组，相同类型的规则会在 SQLE 的页面上展示在一起
    Level:    driver.RuleLevelError, # 规则等级，表示该规则的严重程度。在插件注册阶段，会使用所有 RuleLevelError 级别的规则创建一个默认的规则模板。
}
```

规则的处理函数如下：

```
func(ctx context.Context, rule *driver.Rule, sql string) (string, error) {
    if strings.Contains(sql, "select *") {
        return rule.Desc, nil
    }
    return "", nil
}
```

这里为了演示，这个处理函数只是简单的使用了字符串匹配的方式，你也可以使用正则或者 AST 语法树的方式来检查 SQL 语句（AST 的方式会在「**自定义 SQL 解析器**」一小节中介绍）。

最后将插件规则与规则处理函数通过 `plugin.AddRule()` 函数注册到 SQLE 中，注册完成后，你的 main 文件应该是这样的：

```
package main

import (
    "github.com/actiontech/sqle/sqle/driver"
    adaptor "github.com/actiontech/sqle/sqle/pkg/driver"
)

func main() {
    plugin := adaptor.NewAdaptor(&adaptor.MssqlDialector{})
    aviodSelectAllColumn := &driver.Rule{
        Name:     "aviod_select_all_column",
        Desc:     "避免查询所有的列",
        Category: "DQL规范",
        Level:    driver.RuleLevelError,
    }
    aviodSelectAllColumnHandler := func(ctx context.Context, rule *driver.Rule, sql string) (string, error) {
        if strings.Contains(sql, "select *") {
            return rule.Desc, nil
        }
        return "", nil
    }
    plugin.AddRule(aviodSelectAllColumn, aviodSelectAllColumnHandler)

    ////////////////////////////////////////
    // ... 编写更多规则并通过 AddRule 注册到 SQLE 中
    ////////////////////////////////////////

    // 最后关键一步，调用 `plugin.Serve()` 启动插件:
    plugin.Serve()
}
```

4. 构建并使用插件

和通常的程序编写流程一样，编写完插件代码后，需要将其构建成二进制文件，然后才能将其注册到 SQLE 中。执行 `go build -o ${二进制名} main.go` 将插件代码构建成二进制文件。最后参考数据库审核插件使用 来使用你的自定义插件。

二、 自定义部分

1. 自定义 SQL 解析器

前面介绍的审核规则都是通过字符串匹配的方式来解析 SQL 的内容。这种方式适合规则较少且 SQL 简单的情况下使用。

如果需要对 SQL 进行更复杂的解析匹配，恰好你选择的数据库插件又有相应的 SQL 解析器，这时可以使用自定义 SQL 解析器的方式来编写插件。

首先通过调用 `WithSQLParser()` 注册自己的 SQL 解析器。在添加规则时则使用 `plugin.AddRuleWithSQLParser()` 添加带有解析器的处理函数。在处理函数中，将 interface{} 断言成具体的 AST 语法树，通过语法树级别的操作来更加精细的处理 SQL。PostgreSQL 提供了基于 cgo 调用的解析器（见：PostgreSQL SQL 解析器），下面的代码展示了如何写一个 SQL 解析器的插件：

```go
func main() {
    plugin := adaptor.NewAdaptor(&adaptor.PostgresDialector{})

    // 依然是定义规则
    aviodSelectAllColumn := &driver.Rule{
        Name:     "aviod_select_all_column",
        Desc:     "避免查询所有的列",
        Category: "DQL规范",
        Level:    driver.RuleLevelError,
    }

    // 依然是定义处理函数，这时处理函数的参数是 interface{} 类型，需要将其断言成 AST 语法树。
    aviodSelectAllColumnHandler := func(ctx context.Context, rule *driver.Rule, ast interface{}) (string, error) {
        node, ok := ast.(*parser.RawStmt)
        if !ok {
            return "", errors.New("ast is not *parser.RawStmt")
        }

        switch stmt := node.GetStmt().GetNode().(type) {
        case *parser.Node_SelectStmt:
            for _, target := range stmt.SelectStmt.GetTargetList() {
                column, ok := target.GetResTarget().GetVal().GetNode().(*parser.Node_ColumnRef)
                if !ok {
                    continue
                }
                for _, filed := range column.ColumnRef.GetFields() {
                    _, ok = filed.GetNode().(*parser.Node_AStar)
                    if ok {
                        return rule.Desc, nil
                    }
                }
            }
        }
        return "", nil
    }

    // 依然是注册规则，与前面的例子不同的是，这时使用的是 `plugin.AddRuleWithSQLParser()` 。
    plugin.AddRuleWithSQLParser(aviodSelectAllColumn, aviodSelectAllColumnHandler)

    // 依然是启动插件，与前面的例子不同的是，需要将 SQL 解析的方法注册到插件中。
    plugin.Serve(adaptor.WithSQLParser(func(sql string) (ast interface{}, err error) {
        // parser.Parse 使用 PostgreSQL 的解析器，将 sql 解析成 AST 语法树。
        result, err := parser.Parse(sql)
        if err != nil {
            return nil, errors.Wrap(err, "parse sql error")
        }
        if len(result.Stmts) != 1 {
            return nil, fmt.Errorf("unexpected statement count: %d", len(result.Stmts))
        }

        // 将 SQL 的语法树返回。
        return result.Stmts[0], nil
    }))
}
```

2. 自定义插件

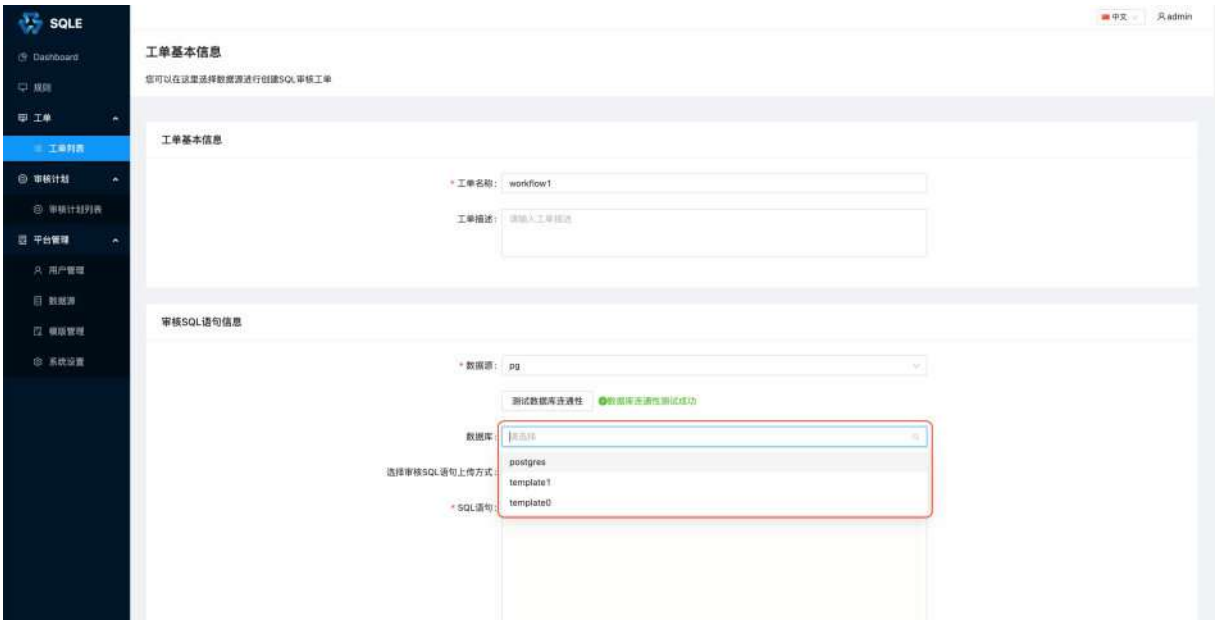
如果 driver 包中默认的 PostgreSQL、Oracle 与 SQL Server 插件不能满足你的需求的话，你也可以自定义一个数据库插件。方法就是实现一个接口：

```go
type Dialector interface {
    Dialect(dsn *driver.DSN) (driverName string, dsnDetail string)
    ShowDatabaseSQL() string
    String() string
}
```

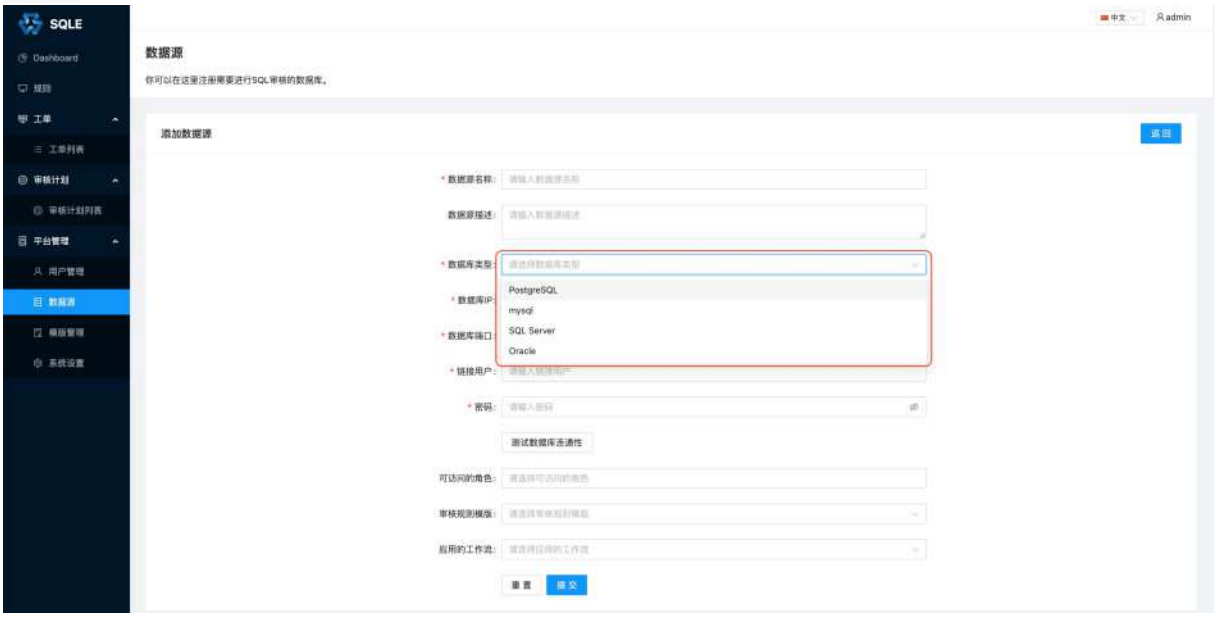
在实现这个接口前，你需要先了解一下 Go 语言原生 Driver 的概念（以 MySQL Driver 为例）。

下面介绍 Dialector 接口的含义：

1. `Dialect`：实现该方法，通过 DSN 提供的 Host Port User Password Database 信息和你选择的数据库 driver，构造出 driverName 与 dsnDetail。driverName 是你引入的数据库 driver 名称；dsnDetail 是连接数据库驱动的必要信息。这两个参数最终会通过 `sql.Open()` 来创建一个数据库连接。
2. `ShowDatabaseSQL`：实现该方法，可以自定义你数据源中默认展示的数据库列表，该数据库列表最终会展示在工单审核列表的数据库下拉框中，如下图：



3. `String`：实现该方法，该方法的返回值会作为你实现的数据库审核插件名展示在 SQLE 的相关下拉框中，如下图：



将你的实现作为 NewAdaptor 的参数传入即可，后续的步骤与前面规则相关的介绍一致。

二、详细部分

1. SQLE与插件的交互图

2. 插件接口说明

2.1 注册接口说明

该接口定义了该插件的名称和实现的规则，SQLE启动的时候会调用该接口获取插件名称和该插件支持的规则列表。

```
type Registerer interface {
    Name() string
    Rules() []*model.Rule
}
```

1. `Name`：插件名，最终会展示在 SQLE 页面的数据源类型的下拉框中；
2. `Rules`：插件支持的规则，在启动 SQLE 时，会调用插件获取这些规则，你将在规则模板内看到它们。

2.2 审核接口说明

该接口定义了SQLE进行审核时，由插件完成的和具体数据库底层交互的操作

```
type Driver interface {
    Close(ctx context.Context)
    Ping(ctx context.Context) error
    Exec(ctx context.Context, query string) (driver.Result, error)
    Tx(ctx context.Context, queries ...string) ([]driver.Result, error)
    Schemas(ctx context.Context) ([]string, error)
    Parse(ctx context.Context, sqlText string) ([]Node, error)
    Audit(ctx context.Context, sql string) (*AuditResult, error)
    GenRollbackSQL(ctx context.Context, sql string) (string, string, error)
}
```

1. `Close`：关闭审核插件使用的相关资源，通常是完成一次审核后，关闭数据库连接等资源;
2. `Ping`：检测数据库的连接性,通常在添加数据源时，为了检测填写的数据是否正确，会调用此方法;
3. `Exec`：执行 SQL 上线时执行此方法；
4. `Tx`：执行 SQL 上线时执行此方法，一般当SQL是DML时且需要事务执行时会批量执行SQL；
5. `Schemas`：返回审核插件展示给用户的 Schema 列表；
6. `Parse`：解析审核插件支持的 SQL 格式；
7. `Audit`：根据指定的SQL语句生成审核建议；
8. `GenRollbackSQL`：生成 SQL 的回滚语句。

2.3 插件的配置信息说明

```
type Config struct {
    DSN           *DSN
    Rules          []*Rule
}
```

1. `DSN`：数据源信息, 待审核的数据库；
2. `Rules`：本次审核制定的规则列表。

2.4 初始化函数说明

插件的主进程入口，由插件的 main 函数调用即可实现插件

```
func ServePlugin(r Registerer, newDriver func(cfg *Config) Driver)
```

1. `r`：传入 Registerer 的接口实现, 由插件侧实现；
2. `newDriver`：传入 Driver 的初始化函数，该函数的入参是 `Config` 是由 SQLE 向插件传递的配置信息，函数的出参是 Driver 的接口实现，由插件侧实现。

系统设置

目录

背景

SQLE 将部分需要手动配置后 才能正常工作的功能 集中放在系统设置页面配置, **管理员**在系统设置界面可以方便的修改相关参数, 启停相关功能

LDAP设置

背景

LDAP 简介

企业内部需要认证的服务很多，员工需要记住很多的密码， 即使对这些服务进行相同的密码设置，也存在很大的安全隐患。每一个新员工的到来管理员都要初始化很多密码，而这些密码都被设置成了“888888”等弱密码，由于各种软件的认证机制之间没有使用一个统一的标准，员工无法一次性修改所有服务的密码，这导致很多即使是入职很久的员工都还在使用这个“众所周知”的密码。

另外一个比较严重的问题出现在公司增加内部服务的时候，例如领导要在公司内部提供邮件服务或把现有的Proftpd 换成更高效的Vsftpd，管理员需要重新为所有的员工初始化新的账户信息，对于一个有上千员工的企业来说这将是一个“灾难”。

如果可以为各种软件提供一个标准的认证机制，所有软件就可以不再用独有的用户管理方法， 而是通过这种统一的认证机制进行用户认证，这样就解决了目前很多企业遇到的问题。LDAP正是这样一种标准的协议，LDAP的历史可以追溯到1988年，之后诞生的很多软件基本上都支持这个协议。近年随着企业对LDAP需求的不断增加，绝大多数有认证机制的软件都会首先提供对LDAP的支持。本文将介绍通过LDAP统一身份认证的方法，以简化这种复杂的管理过程。

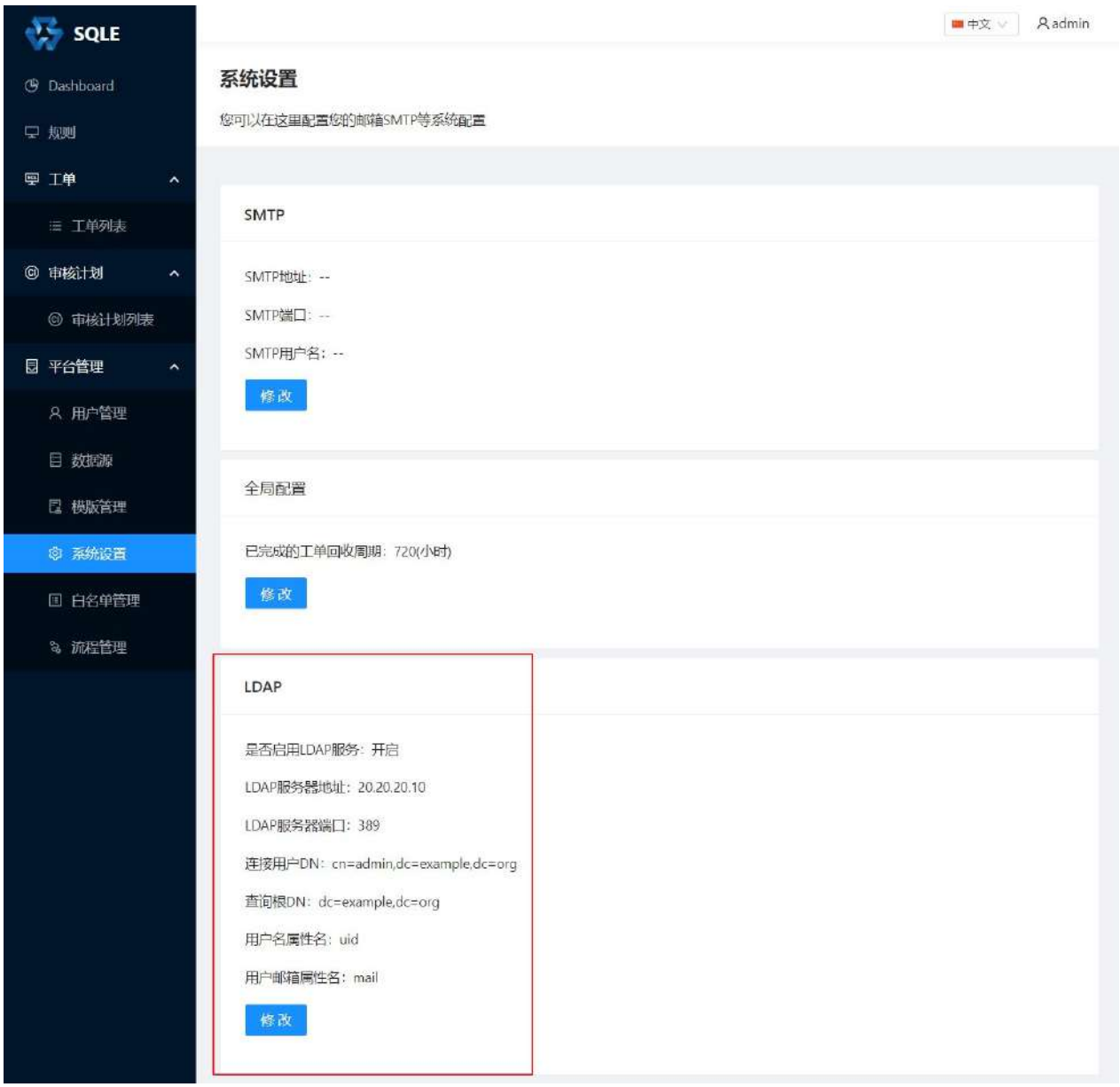
LDAP的内部数据结构为树状结构, 所有的用户和部门都作为这棵树的一个节点, 部门名称,用户名,密码,邮箱等会作为节点的一个属性存在

SQLE 中的LDAP登录功能

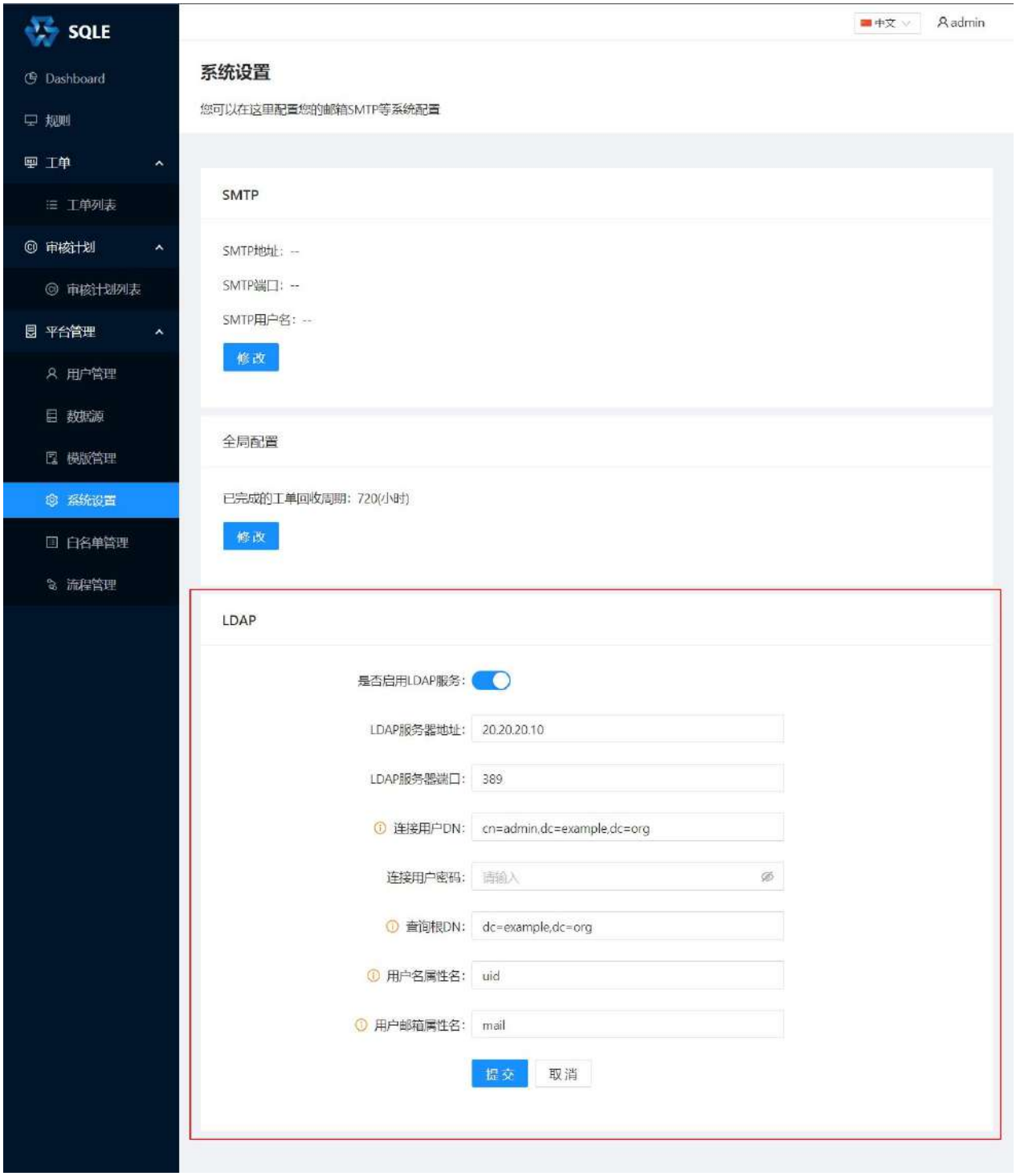
SQLE中支持通过配置的方式接入LDAP系统, 如同其他系统一样接受统一的用户管理, 以便于管理员管理SQLE中的用户, 也使得员工无需在使用一套公司内通用账号密码的同时单独记录SQLE的账号密码

在SQLE中, 如果正确配置并启用了LDAP功能, 用户就可以在登录界面使用LDAP中记录的账号密码进行登录, SQLE会在员工第一次登录时同步此员工的账号和邮箱, 并在以后的登录中均通过LDAP校验此账户

LDAP配置修改方式



LDAP的配置展示在如图的位置, 点击对应 **[修改]** 按钮可以进行配置修改, 点击按钮后效果如下图



可根据需要修改对应条目, 修改后点击 **[提交]** 保存配置, 也可点击 **[取消]** 取消本次修改

修改后ldap用户即可通过sqle登录界面进行登录


Action SQLe
SQL审核平台
ldap_user1
登录

LDAP配置各参数释义

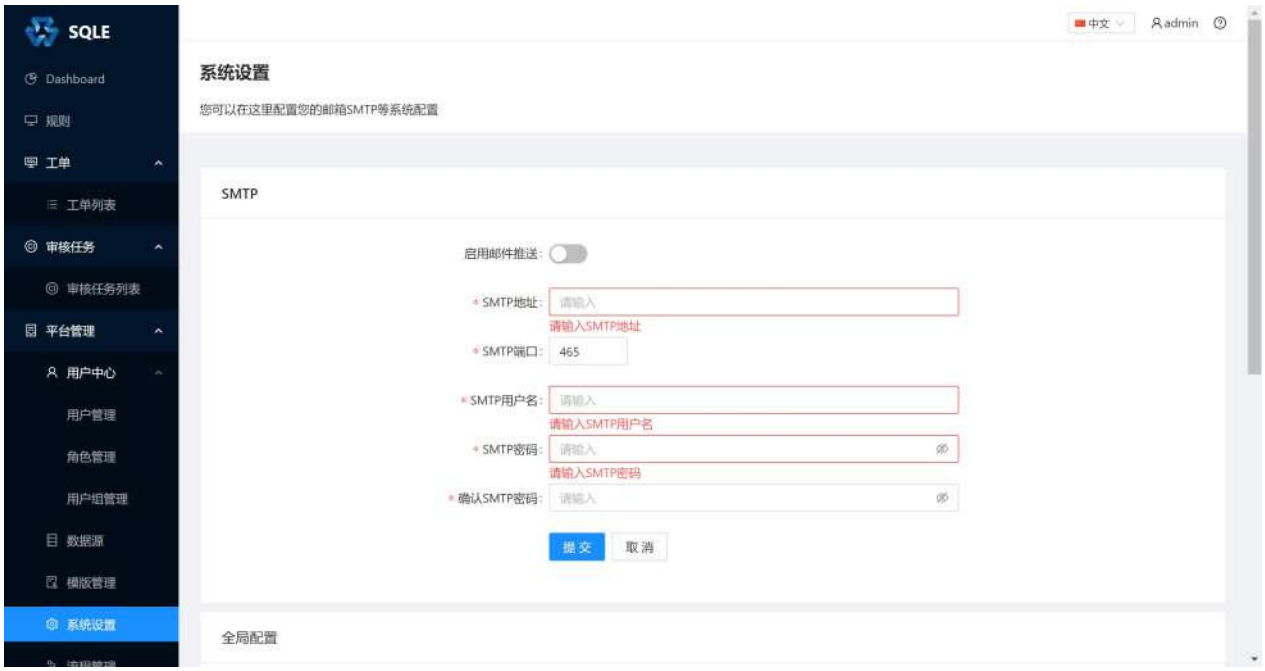
1. 是否启用LDAP服务
   - 填写说明: 此选项为开关
   - 作用: 只有当此选项被打开时LDAP功能才会生效. 此选项关闭时仅保存LDAP相关配置, 不会使用此功能
2. LDAP服务器地址
   - 填写说明: 填写LDAP服务器的IP地址(不含端口号)
   - 作用: 用于LDAP登陆时SQLE找到LDAP服务器的位置
3. LDAP服务器端口
   - 填写说明: 填写LDAP服务器上LDAP服务的监听端口
   - 作用: 用于LDAP登陆时SQLE找到LDAP服务的工作端口
4. 连接用户DN:
   - 填写说明: 填写管理用户的 DN(Distinguished Name) ,类似于 'cn=admin,dc=example,dc=org' , 不是管理用户的用户名, 且管理用户需要有查询其他可能登录用户的权限
   - 作用: 当用户登录时需要先使用管理用户登录到LDAP中查找出登录用户在LDAP中的DN, 再使用查找出的DN和登录用户填写的密码进行LDAP登录校验
5. 连接用户密码:
   - 填写说明: 此项不会进行展示, 如果不希望修改之前设置的密码可以不填这一项, 点击保存后会继续使用原密码而不会使用空值覆盖原密码
   - 作用: 连接用户DN登录时的密码
6. 查询根DN
   - 填写说明: 查询时只会查找以此DN为根节点的树, 如两个部门在LDAP中对应的节点互相不在对方的子树上, 此时如果查询根DN设置成其中一个部门的DN, 则另一个部门将无法通过LDAP登录SQLE
   - 作用: 用于限定SQLE中LDAP登录的作用范围
7. 用户属性名
   - 填写说明: 填写用户名在LDAP中对应的节点属性名, 一般设置为uid
   - 作用: 用于LDAP用户登录时查询登录用户的DN, LDAP中此属性对应的值也将用作此用户第一次登录SQLE自动注册时的用户名

8. 用户邮箱属性名

   - 填写说明: 填写用户邮箱在LDAP中对应的节点属性名, 一般设置为mail
   - 作用: SQLE将根据此属性名从LDAP中获取到用户的邮箱, 并将此邮箱用于此用户第一次登录SQLE自动注册时的用户邮箱

邮箱配置

SQLE 支持在工单流转过程中，通过邮件通知工单下一步操作人。在系统配置中配置发件邮箱服务器的信息，如下：

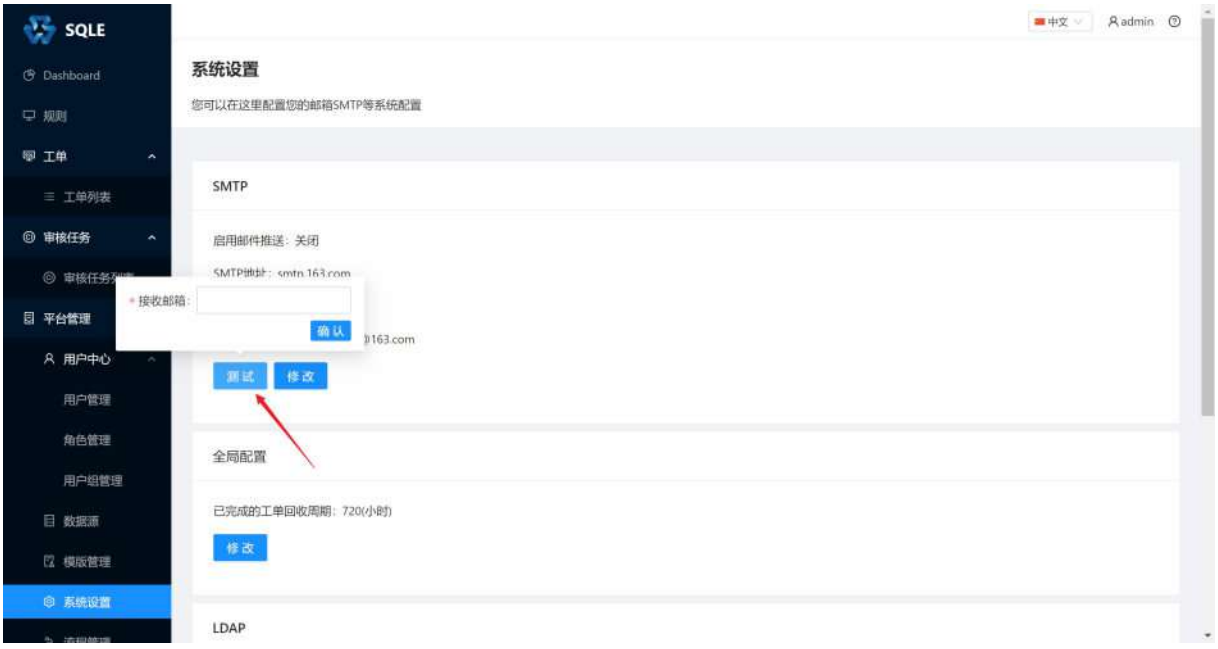


邮箱配置项：

- 是否启用邮件通知: 启用后才会发送邮件通知, 邮件通知启停不会影响其他通知方式
- SMTP 地址：SMTP 服务器地址，如 smtp.163.com
- SMTP 端口：SMTP 服务器端口，如 25
- SMTP 用户名：完整的邮箱用户名
- SMTP 密码：该用户名的授权码（非邮箱用户密码）

邮箱测试方式:

- 配置好相关参数
- 点击测试按钮, 输入邮箱地址点击测试



- 查看测试邮箱是否收到测试邮件

FAQ

如果上面的内容仍然无法解答你的问题，欢迎上 GitHub 提交 issue。

一、诊断手法

1. 查看systemd日志

如果是 centos7 下 systemctl 启动，通过 `journalctl` 查看启动日志

```
journalctl -u sqled|grep -v systemd # 如果未筛选出内容可执行 journalctl -u sqled 查看全部内容
```

二、常见问题

1. `sqled.yml` 配置文件权限不足

诊断

检查 SQLE 安装路径下 `sqle/etc/sqled.yml` 对于用户 `actiontech-universe` 是否有可读权限。

修复

```
chown actiontech-universe:actiontech sqled.yml
```

2. 程序启动后打开页面白屏

原因1：未使用 systemd 启动程序

使用命令行直接启动 SQLE， 需要在 SQLE 主目录下启动，不然找不到前端文件，**建议使用systemd**，如果必须使用命令行，可以参考：

```
cd /opt/sqle # 假设安装在/opt目录下
./bin/sqled --config /opt/sqle/etc/sqled.yml --pidfile=/opt/sqle/sqled.pid
```

原因2：缺少前端文件

确认 sqle/ui 目录是否存在或是否有内容。如果损坏，建议重新安装。

3. 执行 make docker_rpm 报错

SQLE 的 RPM 打包使用的是 Docker 打包方式。当执行 `make docker_rpm` 报错时，大多数情况需要到 Docker 打包容器里查看报错日志。具体步骤如下：

1. 在 docker_rpm target 中添加一个 sleep，如下图：

2. 进入 Docker 打包容器
3. cat /tmp/build.log 查看报错信息

回滚语句常见问题

1. DML 语句没有回滚语句生成

由于一条 DML 可能影响的行数是不确定的，所以需要限制回滚语句的生成。即影响行数超过阈值则不生成回滚语句，对应的规则名为「在 DML 语句中预计影响行数超过指定值则不回滚」。

原因1：未配置相关规则

如果没有配置上面的规则，默认不会走生成回滚语句的流程（即使影响行数很少）。

原因2：影响行数超过了规则阈值