

# 数据库附加说明文档

## 1.1) 部署策略

### 部署选择

*Docker* 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux或Windows操作系统的机器上，也可以实现虚拟化。

modified

阿里云->腾讯云

本次课设中，我们采用腾讯云ubuntu服务器，使用docker部署oracle数据库，理由如下：

1. docker安装快速，效率高
2. docker隔离性好，可以安装多个oracle实例，互不干扰，只要管理好对主机端口的映射即可
3. 卸载管理更方便和干净，直接删除容器和镜像即可
4. 数据备份、迁移功能更加方便强大
5. 性能与直接安装接近，但是启动速度远快于直接安装

### 部署流程

docker的安装较为简单，直接apt安装即可，使用

```
$ docker search oracle
```

可以搜索所有支持的oracle镜像，在此选择使用oracle12c版本

使用

```
$ docker pull truevolly/oracle-12c
```

拉取镜像后，使用

```
$ docker run -d -p 1521:1521 truevolly/oracle-12c
```

即可将镜像实例化。通过truevolly/oracle-12c镜像实例化了一个容器，同时指定容器的1521端口（oracle的默认端口）映射到了宿主服务器的1521端口。实际上在此省略了一个容器卷的部署，在后面将提及

部署成功后，使用

```
$ docker exec -it ${镜像名} bash
```

即可进入容器，使用

```
$ORACLE_HOME/bin/sqlplus
```

可以进入数据库实例的sqlplus控制台

## 1.2) 库设计

### 表设计

- 保持规范性

- 为了保证数据库中数据的完整性，在表格设计时考虑了一些缺省、非空约束、check约束等例如在News表中

```
create table News (  
    news_id number not null,  
    title varchar2(128) not null,  
    publishDateTime date not null,  
    summary varchar2(512) not null,  
    contains varchar2(3999) not null,  
    matchTag varchar2(20) not null,  
    propertyTag varchar2(20) not null,  
    primary key (news_id),  
    check (matchTag in ('中超', '西甲', '意甲', '法甲', '德甲', '英超'))  
);
```

对于新闻的id、标题、发布时间、简略、内容、比赛tag、类型tag都进行了非空约束，对比赛tag还额外进行了check in 约束，保证数据的合法性。

而在Posts表中

```
modified  
修改三行sql
```

```
create table Posts (  
    post_id number not null,  
    publishDateTime date not null,  
    contains varchar2(1000) not null,  
    isBanned number(1,0) default on null 0 not null,  
    approvalNum number default on null 0 not null,  
    favouriteNum number default on null 0 not null,  
    title varchar2(50) not null,  
    primary key (post_id)  
);
```

除了一些非空约束以外，还设定了缺省值。例如封禁标志isBanned就缺省为0

- 为了数据库的可维护性，还需要对部分外码设置级联。如果发生进行删除用户等操作时发现由于完整性约束使得操作不可进行等事件，数据库的可维护性显然不足

例如在用户发表帖子的关系集PublishPost中

```
create table PublishPost (  
    post_id number not null,  
    user_id number not null,  
    primary key (post_id, user_id),  
    foreign key (post_id) references Posts on delete cascade,  
    foreign key (user_id) references Usr on delete cascade  
);
```

对Posts(post\_id)和Usr(user\_id)的外码引用设计了级联删除，这保证了在删除帖子或者删除用户时，对应的PublishPost关系也会被移除，数据库事务保持一致性

而在球队球员联系集TeamOwnPlayer中

```
create table TeamOwnPlayer (  
    team_id number,  
    player_id number,  
    primary key (team_id, player_id),  
    foreign key (team_id) references team on delete cascade,  
    foreign key (player_id) references players on delete cascade  
);
```

对team(team\_id)和players(player\_id)的外码引用也设计了级联删除，在队伍或者球员信息被删除时，对应的TeamOwnPlayer关系也会被安全移除

- 在项目的设计中，有一些数据是经常同时需要，但又不能通过简单逻辑查询的，例如在获取Posts的数据时，倾向于同时获取其点赞数、收藏数，严格按照3NF设计的数据库中，点赞数、收藏数需要Posts表与likePost、collectPost表进行联表查询，这会造成性能的浪费以及后端逻辑的复杂性。

为了解决这个问题，我们尝试在数据库中增加了一些冗余字段，但造成数据库设计的不规范，破坏数据库的一致性问题，因此我们通过触发器对其进行维护。下面对触发器进行说明：

- incrFollowedNumber、decrFollowedNumber

```
create or replace TRIGGER decrFollowedNumber  
BEFORE DELETE ON follow  
FOR EACH ROW  
BEGIN  
    UPDATE usr  
    SET followedNumber = followedNumber - 1  
    WHERE USER_ID = :OLD.FOLLOW_ID;  
END;
```

```
create or replace TRIGGER incrFollowedNumber  
AFTER INSERT ON follow  
FOR EACH ROW  
BEGIN  
    UPDATE usr  
    SET followedNumber = followedNumber + 1  
    WHERE USER_ID = :NEW.FOLLOW_ID;  
END;
```

这两个触发器对用户的被关注数进行维护，当follow表单被进行插入或者删除操作时，对follow\_id对应的用户的followedNumber字段进行自增、自减操作

- incrFollowNumber、decrFollowNumber

```
create or replace TRIGGER decrFollowNumber
BEFORE DELETE ON follow
FOR EACH ROW
BEGIN
    UPDATE usr
    SET followNumber = followNumber - 1
    WHERE USER_ID = :OLD.FOLLOWER_ID;
END;
```

```
create or replace TRIGGER incrFollowNumber
AFTER INSERT ON follow
FOR EACH ROW
BEGIN
    UPDATE usr
    SET followNumber = followNumber + 1
    WHERE USER_ID = :NEW.FOLLOWER_ID;
END;
```

这两个触发器对用户的关注数进行维护，当follow表单被进行插入或者删除操作时，对follower\_id对应的用户的followNumber字段进行自增、自减操作

- incrPostApprovalNum、decrPostApprovalNum

```
create or replace TRIGGER decrPostApprovalNum
BEFORE DELETE ON LIKEPOST
FOR EACH ROW
BEGIN
    UPDATE posts
    SET approvalnum = approvalnum - 1
    WHERE post_id = :OLD.post_id;
END;
```

```
create or replace TRIGGER incrPostApprovalNum
AFTER INSERT ON LIKEPOST
FOR EACH ROW
BEGIN
    UPDATE posts
    SET approvalnum = approvalnum + 1
    WHERE post_id = :NEW.post_id;
END;
```

这两个触发器对帖子的赞同数进行维护，当LIKEPOST表单被进行插入或者删除操作时，对post\_id对应的帖子的approvalnum字段进行自增、自减操作

modified

修改

- update\_game\_score\_after\_insert、update\_game\_score\_after\_delete

```
create or replace trigger update_game_score_after_insert
after insert on playerjoingame
for each row
declare
```

```

this_team_id number;
this_hometeam number;
this_guestteam number;
begin

    -- 获取球员所属队伍的team_id
    select team_id into this_team_id
    from teamownplayer top
    where top.player_id=:new.player_id;

    -- 获取赛事的主客队id
    select hometeam into this_hometeam
    from gameteam gt
    where gt.game_id=:new.game_id;

    select guestteam into this_guestteam
    from gameteam gt
    where gt.game_id=:new.game_id;

    --更新主队得分
    update game g
    set g.homescore=g.homescore+:new.goal
    where g.game_id=:new.game_id and this_team_id=this_hometeam;

    --更新客队得分
    update game g
    set g.guestscore=g.guestscore+:new.goal
    where g.game_id=:new.game_id and this_team_id=this_guestteam;

end;

```

```

create or replace trigger update_game_score_after_delete
after delete on playerjoingame
for each row
declare
    this_team_id number;
    this_hometeam number;
    this_guestteam number;
begin

    -- 获取球员所属队伍的team_id
    select team_id into this_team_id
    from teamownplayer top
    where top.player_id=:old.player_id;

    -- 获取赛事的主客队id
    select hometeam into this_hometeam
    from gameteam gt
    where gt.game_id=:old.game_id;

    select guestteam into this_guestteam
    from gameteam gt
    where gt.game_id=:old.game_id;

    --更新主队得分

```

```

update game g
set g.homescore=g.homescore-:old.goal
where g.game_id=:old.game_id and this_team_id=this_hometeam;

--更新客队得分
update game g
set g.guestscore=g.guestscore-:old.goal
where g.game_id=:old.game_id and this_team_id=this_guestteam;

end;

```

这两个触发器对比赛的得分进行维护，当playerJoinGame表单被进行插入或者删除操作时，对game中对应的帖子的homeScoe、guestScore字段进行自增、自减操作

- incrPostCollect、decrPostCollect

```

create or replace TRIGGER decrPostCollect
BEFORE DELETE ON COLLECTPOST
FOR EACH ROW
BEGIN
    UPDATE posts
    SET favouritenum = favouritenum - 1
    WHERE post_id = :OLD.post_id;
END;

```

```

create or replace TRIGGER incrPostCollect
BEFORE INSERT ON COLLECTPOST
FOR EACH ROW
BEGIN
    UPDATE posts
    SET favouritenum = favouritenum + 1
    WHERE post_id = :NEW.post_id;
END;

```

这两个触发器对帖子的收藏数进行维护，当COLLECTPOST表单被进行插入或者删除操作时，对post\_id对应的帖子的favouritenum字段进行自增、自减操作

- unique\_report\_trigger

```

create or replace TRIGGER unique_report_trigger
BEFORE INSERT ON reports
FOR EACH ROW
DECLARE
    existing_report_count NUMBER;
BEGIN
    IF :NEW.status = 'unhandled' THEN
        -- 检查是否存在其他相同的记录
        SELECT COUNT(*)
        INTO existing_report_count
        FROM reports
        WHERE reporter_id = :NEW.reporter_id
        AND post_id = :NEW.post_id
        AND status = 'unhandled'
        AND report_time != :NEW.report_time; -- 排除当前记录
    END IF;
END;

```

```
-- 如果存在其他相同的记录，阻止插入或更新操作
IF existing_report_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, '同一举报者对同一帖子的举报只能存在一个。');
END IF;
END IF;
END;
```

这个触发器是辅助完成业务逻辑设置的。

业务逻辑包含：同一个用户对同一个帖子的状态为'unhandled'的举报在同一时间最多只能存在一个，而使用触发器可以很好的在违反约束的事务进行前进行阻止。

当数据被插入reports时，先计数表格中同一个用户对同一个帖子的状态为'unhandled'的举报，当此种数据已经存在时，阻止插入事务并抛出报错信息。

## 优化尝试

- 对赛事通过时间进行排序的操作比较多，而在sql developer中进行该操作大概需要1.6s 尝试优化加速一下，因此在startTime上创建索引

```
create index game_time_index on game(starttime)
```

创建后对赛事通过时间进行排序大概需要1.4-1.5秒，优化不是非常显著，这是由于在数据的爬取时基本是按照时间进行的，导致数据处于有序状态。

## 1.3) 配置管理

### 数据持久化

容器通常是临时的、易于销毁和重新创建的。尤其是为了应对数据的不慎销毁问题，我们需要进行数据持久化处理来保证数据库的持久性和可靠性。

选择docker部署的另一个考虑就基于此。docker官方提供了一个叫做数据卷(Volume)的数据持久化方案。

数据卷是宿主机中的一个目录或者文件，当容器目录和数据卷目录绑定后，对容器的修改会立即同步。

一个数据卷可以被多个容器同时挂载，一个容器也可以被挂载多个数据卷。

简单来说，数据卷本质其实是共享文件夹，是宿主机与容器之间数据共享的桥梁。

根据原理，数据卷将一个docker容器中的数据挂载到服务器中的文件上，在之后通过对该文件的操作可以轻松实现数据的备份、共享以及修改。

数据卷的创建并不复杂，创建数据卷并在实例化容器时指定数据卷挂载即可

```
$ docker volume create dbdata
$ docker run -d -p 1521:1521 -v dbdata:${path} truevolly/oracle-12c
```

之后可以查看容器的具体信息

```
$ sudo docker volume inspect dbdata
```

可以看到下列输出：

```
ubuntu@vm-4-9-ubuntu:~$ sudo docker volume inspect dbdata
[
  {
    "CreatedAt": "2023-08-16T17:28:57+08:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/dbdata/_data",
    "Name": "dbdata",
    "Options": {},
    "Scope": "local"
  }
]
```

对服务器路径 `/var/lib/docker/volumes/dbdata/_data` 下的文件进行定时备份即可完成数据的持久化处理。当不慎删除了容器时，数据卷并不会被删除，只需要重新实例化容器并指定挂载的数据卷即可恢复数据

## 参数文件

Oracle中的参数文件是一个包含一系列参数以及参数对应值的操作系统文件。它们是在数据库实例启动第一个阶段时候加载的，决定了数据库的物理结构、内存、数据库的限制及系统大量的默认值、数据库的各种物理属性、指定数据库控制文件名和路径等信息，是进行数据库设计的重要文件。

更旧版本的oracle只采用了pfile一种参数文件格式，在本项目采用的12c版本中已经加入了spfile文件。

pfile可以通过文本编辑器直接修改参数值，并且需要手动重新启动数据库实例才能使更改生效。

spfile是二进制文件格式，也包含了Oracle数据库实例的配置参数，它不能直接通过文本编辑器进行修改，而是通过Oracle提供的ALTER SYSTEM语句或图形界面工具进行更改。

oracle实例默认是根据spfile文件进行启动的。由于希望修改一下会话时长的限制，使用了ALTER SYSTEM语句对spfile文件进行修改，并根据spfile文件与pfile文件进行了相互生成

```
create spfile from pfile
create pfile from spfile
```

由于疏忽并没有指定生成的文件的路径以及文件名，导致了文件的覆盖。当我关闭了数据库实例想要重启时候，无论是通过缺省方式还是指定pfile，都会给我报错参数错误

spfile无法查看，通过

```
vim /u01/app/oracle/product/12.1.0/xe/dbs/init.ora
```

发现内容为空，根据其生成的spfile也显然无法使用

根据官方文档，oracle有一个默认的配置文件的init，理解为pfile文件的例子即可。尝试根据其启动数据库。首先在目录中找到该文件

```
/u01/app/oracle/admin/xe/pfile/init.ora.716202394632
```

对其进行备份后修改会话时长限制



尝试在sqlplus中启动数据库实例:

```
SQL> startup pfile="/u01/app/oracle/admin/xe/pfile/init.ora.716202394632"
...
Database mounted.
```

成功启动, 之后根据其创建spfile、根据spfile生成pfile并进行备份, 保证数据库配置文件的安全。

## 日志信息

日志信息是记录了数据库的变更, 包括数据插入、更新、删除以及数据定义语言 (DDL) 操作等信息的文件, Oracle 数据库的日志是确保数据库可恢复性、高可用性和一致性的关键组成部分。对其的定期备份也是需要考虑的部分。

首先进入sqlplus停止监听服务

```
$ lsnrctl set log_status off;
```

进入监听文件的目录下,查看文件信息

```
$ cd /u01/app/oracle/diag/tnslsnr/604dc6ad6ec8/listener/trace
/u01/app/oracle/diag/tnslsnr/604dc6ad6ec8/listener/trace$ ls
listener.log
```

查看一下文件内容

```
$ vi listener.log
...
31-AUG-2023 03:40:32 * (CONNECT_DATA=(SERVICE_NAME=xe)(CID=
(PROGRAM=/home/ubuntu/DataBase/DBwebAPI/DBwebAPI/bin/Debug/net6.0/DBwebAPI.dll)
(HOST=VM-4-11-ubuntu)(USER=ubuntu))) * (ADDRESS=(PROTOCOL=tcp)
(HOST=xxx.xx.xxx.xxx)(PORT=xxxxxx)) * establish * xe * 0
...
31-AUG-2023 03:41:32 * service_update * xe * 0
...
```

一切正常。将listener.log文件更名备份后, 创建一个空文件listener.log放在原目录下,最后进入sqlplus启动监听服务

```
$ lsnrctl set log_status on;
```

modified

增加

## 表空间设置

在oracle官方的图形化工具sql developer中查看一下表空间的使用情况

	TABLESPACE_NAME	PERCENT_USED	PCT_USED	ALLOCATED	USED	FREE	DATAFILES
1	SYSTEM		99.47	800	795.75	4.25	1
2	SYSAUX		93.85	820	769.56	50.44	1
3	USERS		85.45	13.75	11.75	2	1
4	UNDOTBS1		8.75	130	11.38	118.63	1
5	TESTSPACE		2	50	1	49	1
6	TEMP		0.51	197	1	196	1

其中项目中的用户文件是存放在USERS空间的。可以看到其空间占用较大，而SYSTEM和SYSAUX也几乎占满。这是由于SYSTEM和SYSAUX空间本就是自动增长的，在占满并自动增长之后数据占用就一直有着较高比例。

在本此设计中，同一个表空间的所属数据文件都已有一个，对表空间的修改可以简化为对数据文件的修改。

查看一下数据文件设置

```
SELECT * FROM dba_data_files;
```

发现其自动增长AUTOEXTENSIBLE都为YES，但是拓展大小NEXT都为空。

默认情况下，Oracle 数据库的 NEXT 大小通常会被设置为一个相对较小的固定值，通常是数兆字节。每次自动增长时，数据文件的大小只会增加一小部分。如果数据库经常执行大型操作或具有高数据增长率，这可能导致频繁的文件增长，从而导致额外的磁盘 I/O 和管理开销。数据文件也会快速达到最大限制，从而引发数据库性能问题和操作中斷。

因此对表空间进行管理

```
alter database datafile '/u01/app/oracle/oradata/xe/users01.dbf' resize 200M ;
```

利用此语句可以将数据文件大小设置为200M。根据上述信息所示，USERS表空间分配100M的空间是完全足够的。







```
alter database datafile '/u01/app/oracle/oradata/xe/users01.dbf' autoextend on
next 20M ;
```

之后执行以上语句使数据文件自动增长步长为20M。

对系统级的文件操作希望更加慎重，因此进行RESIZE操作了。但是应该设置自动拓展步长。

```
alter database datafile '/u01/app/oracle/oradata/xe/system01.dbf' autoextend on
next 100M ;
alter database datafile '/u01/app/oracle/oradata/xe/sysaux01.dbf' autoextend on
next 100M ;
```

现在再次查看表空间

	TABLESPACE_NAME	PERCENT_USED	PCT_USED	ALLOCATED	USED	FREE	DATAFILES
1	SYSTEM		99.47	800	795.75	4.25	1
2	SYSAUX		93.85	820	769.56	50.44	1
3	TESTSPACE		10	10	1	9	1
4	UNDOTBS1		8.75	130	11.38	118.63	1
5	USERS		5.87	200	11.75	188.25	1
6	TEMP		0.51	197	1	196	1

用户表空间十分充足