Новосибирский государственный университет

|  |
| --- |
| УТВЕРЖДАЮ |
|  |
|  |
| (ФИО) |
|  |
| «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_ г. |

Отчет о научно-исследовательской работе:

“**Информационная система авиастроительного предприятия**”

**Выполнил: Бречка Антон**

Новосибирск 2020

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc458005623)

[Глава 1. Анализ проекта 4](#_Toc458005624)

[Глава 2. Проектирование системы 6](#_Toc458005624)

[Глава 3. Реализация системы 8](#_Toc458005624)

[Глава 4. Тестирование 8](#_Toc458005624)

[ЗАКЛЮЧЕНИЕ 9](#_Toc458005625)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc458005626)

[ПРИЛОЖЕНИЯ 11](#_Toc458005627)

ВВЕДЕНИЕ

Проект «Информационная система авиастроительного предприятия» включает в себя:

База данных, в которой хранится информация о структурном представлении предприятия, а также данные работников, участвующих в процессе выполнения работы;

Приложение, предоставляющее интерфейс, взаимодействующее с базой данных, для получения информации по запросам, возможным для роли зарегистрированного пользователя

Имеются следующие требования для приложения:

Получить перечень видов изделий отдельной категории и в целом, собираемых указанным цехом, предприятием.

Получить число и перечень изделий отдельной категории и в целом, собранных указанным цехом, участком, предприятием в целом за определенный отрезок времени.

Получить данные о кадровом составе цеха, предприятия в целом и по указанным категориям инженерно-технического персонала и рабочих.

Получить число и перечень участков указанного цеха, предприятия в целом и их начальников.

Получить перечень работ, которые проходит указанное изделие.

Получить состав бригад указанного участка, цеха.

Получить список мастеров указанного участка, цеха.

Получить перечень изделий отдельной категории и в целом, собираемых в настоящий момент указанным участком, цехом, предприятием.

Получить состав бригад, участвующих в сборке указанного изделия.

Получить перечень испытательных лабораторий, участвующих в испытаниях некоторого конкретного изделия.

Получить перечень изделий отдельной категории и в целом, проходивших испытание в указанной лаборатории за определенный период.

Получить список испытателей, участвующих в испытаниях указанного изделия, изделий отдельной категории и в целом в некоторой лаборатории за определенный период.

Получить состав оборудования, использовавшегося при испытании указанного изделия, изделий отдельной категории и в целом в некоторой лаборатории за определенный период.

Получить число и перечень изделий отдельной категории и в целом, собираемых указанным цехом, участком, предприятием в целом в настоящее время

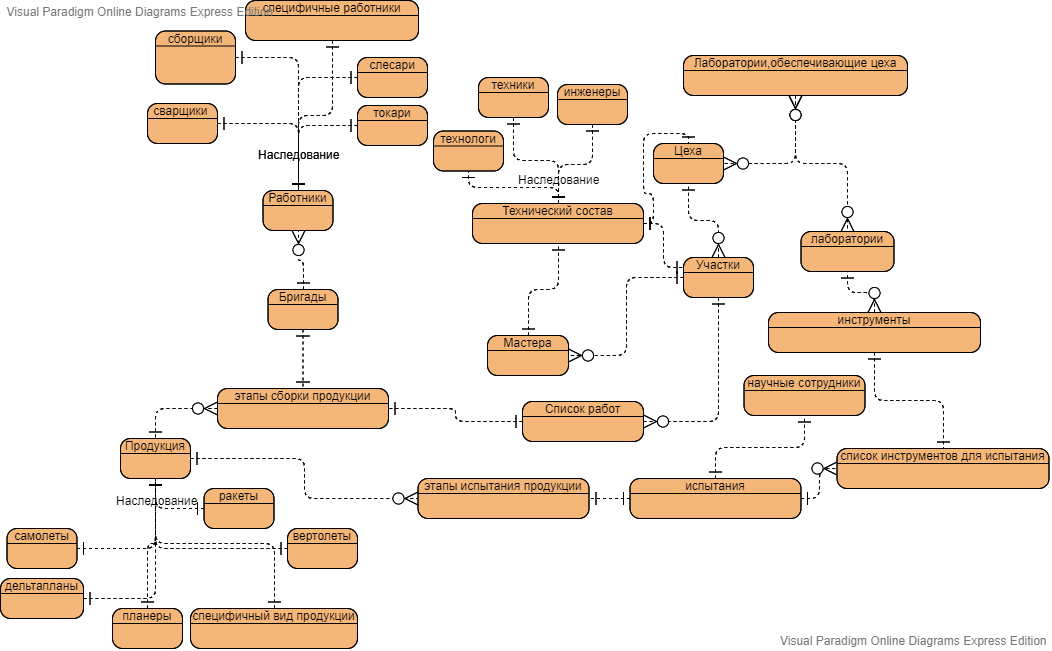
Цель: реализовать проект в виде двухмодульной системы база данных-клиент, удовлетворяющий требованиям

Задачи: произвести анализ требований к проекту, спроектировать систему, реализовать систему, произвести тестирование

Глава 1. Анализ проекта

Исходный анализ проекта заключался в построении ER-модели, представленной на следующей диаграмме [Рисунок 1]

ER-диаграмма



В отношениях тип-подтип участвуют следующие сущности:

Супертип: работники;

Подтипы: слесари, токари, сварщики, сборщики, специфичные работники.

Супертип: технический состав;

Подтипы: техники, технологи, инженеры.

Супертип: продукция;

Подтипы: планеры, ракеты, вертолеты, специфичный вид продукции, самолеты, дельтапланы.

Требования к целостности данных: при реализации базы данных необходимо создать все сущности и отношения между ними, представленные выше, а также следующие проверки:

Уникальное имя продукции между всеми подтипами

При добавлении работника в бригаду проверка на принадлежность к другой бригаде

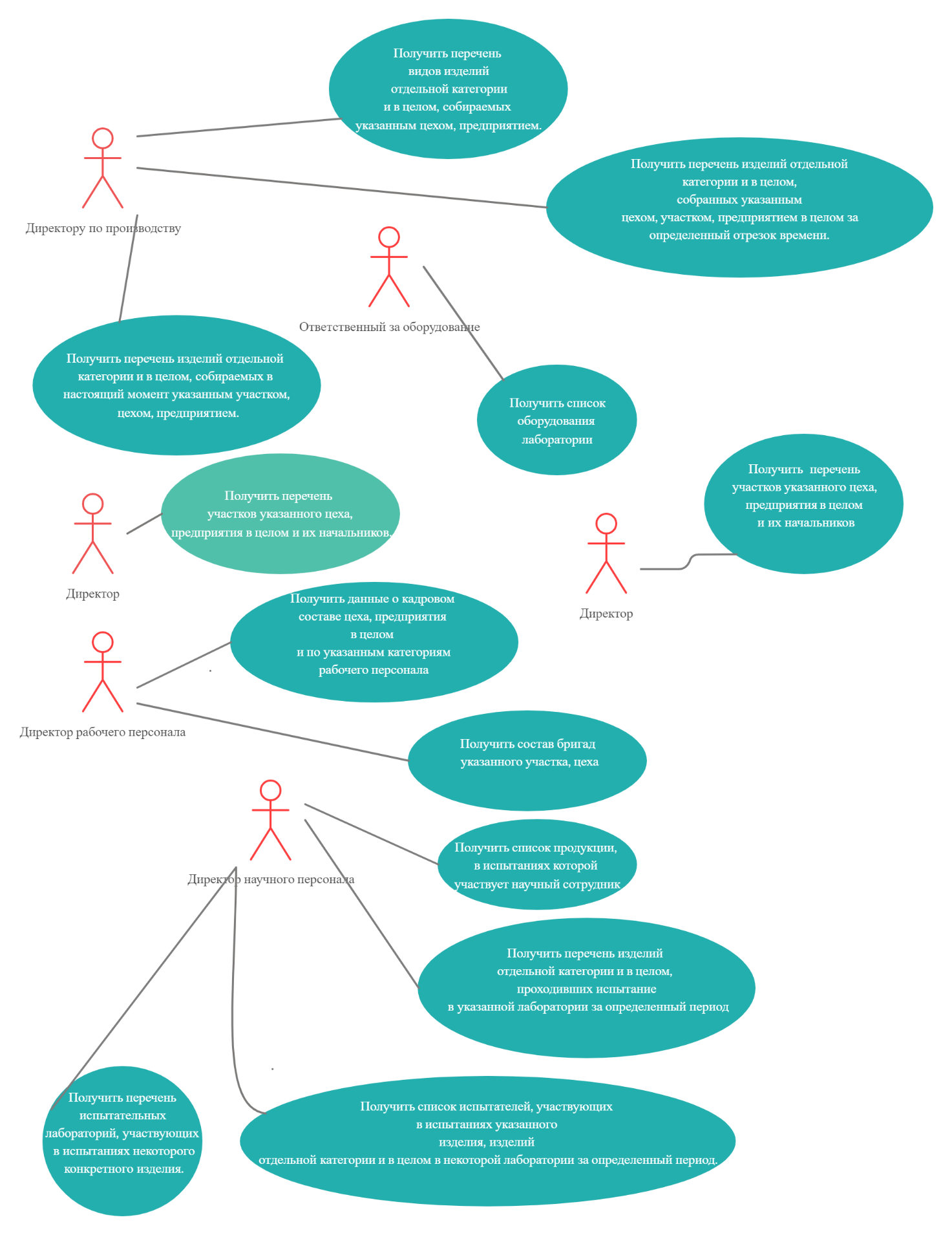
При назначении человека технического персонала на должность мастера, начальника участка, начальника цеха, проверять наличие должности у человека из двух оставшихся вариантов.

При добавлении этапа разработки изделия проверка уникальности цеха для списка работ и выбранного изделия

При добавлении этапа испытания изделия проверка на обслуживание цеха, в котором происходила сборка изделия, лабораторией

Требования, освещенные выше, планируется реализовать непосредственно с помощью базы данных с помощью использования trigger, constraint, primary key, foreign key.

Далее, были выведены роли пользователей приложения и основные возможности взаимодействия с приложением.

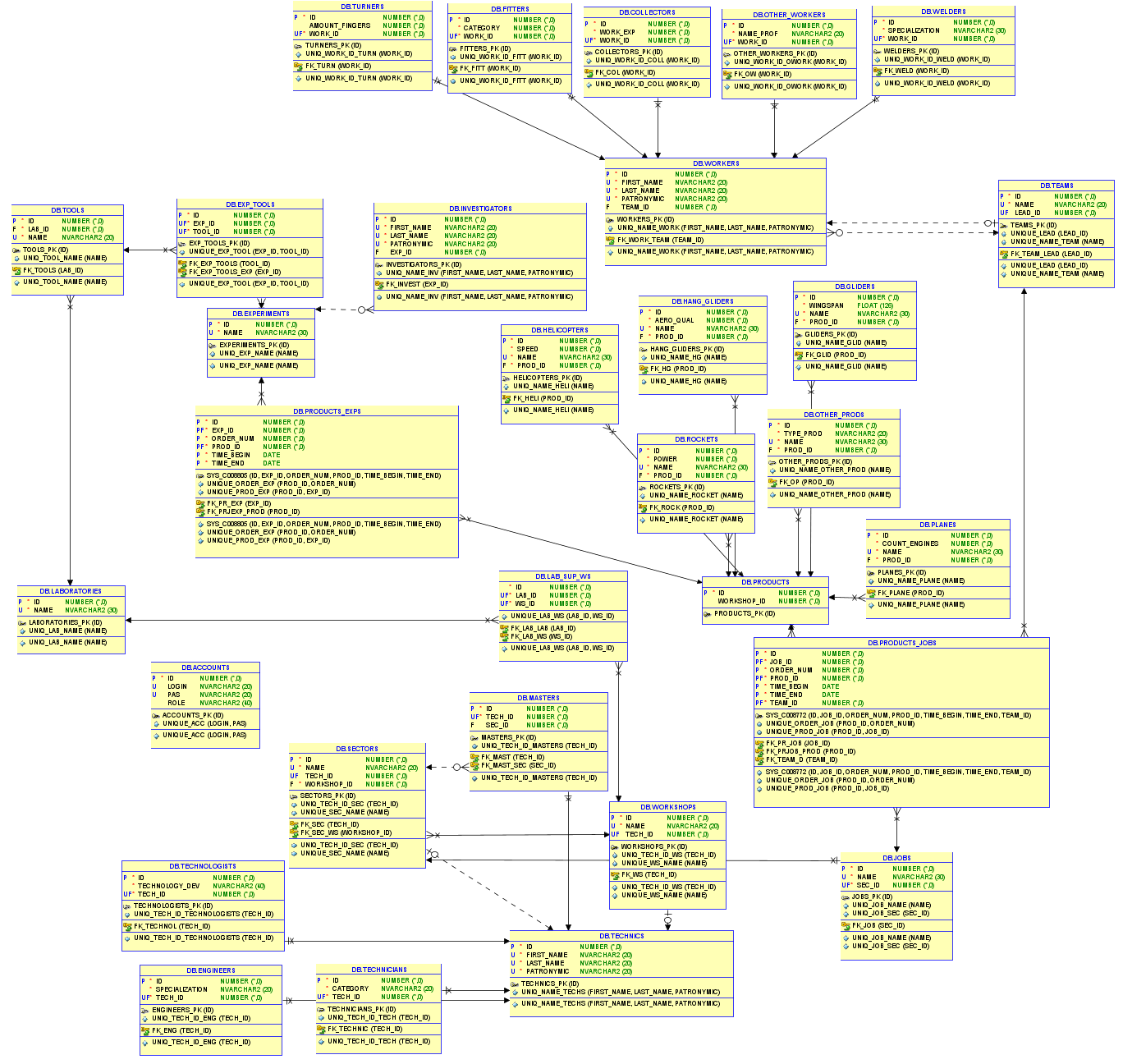
Диаграмма возможностей пользователей

Глава 2. Проектирование системы

Для реализации базы данных использовалась Oracle database XE, поскольку данная СУБД имеет большой спектр возможностей, а также удобна для взаимодействия с клиент-приложением.

Основываясь на функциональных требованиях предложения, была спроектирована схема, показанная ниже.

Схема базы данных



Как было показано в ER модели, реализованы соответствующие таблицы для реализации отношения супертип-подтип:

Для супертипа «ПРОДУКЦИЯ» была создана таблица «PRODUCTS», где уникальный идентификатор соответствует атрибуту prod\_id, существующему в таблицах, соответствующих подтипам

Для супертипа «РАБОЧИЙ» была создана таблица «WORKERS», где уникальный идентификатор соответствует атрибуту work\_id, существующему в таблицах, соответствующих подтипам

Для супертипа «ТЕХНИЧЕСКИЙ ПЕРСОНАЛ» была создана таблица «TECHNICS», где уникальный идентификатор соответствует атрибуту tech\_id, существующему в таблицах, соответствующих подтипам

Для задания работ и испытаний для изделия были реализованы таблицы «PRODUCTS\_JOBS» и «PRODUCTS\_EXPS», которые включают в себя следующие атрибуты:

Общие:

Time\_begin: Дата начала выполнения этапа

Time\_end: Дата окончания выполнения этапа

Prod\_id: Уникальный идентификатор продукта

Order\_num: Порядковый номер этапа

«PRODUCTS\_JOBS»:

job\_id: Уникальный идентификатор работы по сборке

«PRODUCTS\_EXPS»:

exp\_id: Уникальный идентификатор эксперимента

Чтобы была возможность учитывать обслуживает ли лаборатория цех, была добавлена таблица «LAB\_SUP\_WS», которая содержит атрибуты уникального идентификатора цеха и лаборатории.

При разработке базы данных возникли трудности при реализации циклической зависимости между таблицами «TEAMS» и «WORKERS». Для разрешения данного конфликта был использован триггер, который после добавления новой записи в «TEAMS» обновляет таблицу «WORKERS», аналогичный триггер использовался для добавления новой записи в «WORKERS».

В качестве инструмента для создания форм был выбран интерфейс программирования приложений Windows Forms, позволив сократить время разработки пользовательского интерфейса приложения-клиента. Ниже приведена таблица, описывающая требуемые графические формы для реализации функциональных требований от приложения.

Прецеденты – формы – описание формы

|  |  |  |
| --- | --- | --- |
| Выбрать лабораторию по имени;  Выбрать научного сотрудника | Базовая упрощенная форма выбора | Форма с выпадающим списком из существующих элементов выбираемой категории |
| Войти в приложение в соответствии с логином и паролем | Форма входа | Форма, содержащая два текстовых поля и кнопки войти и выйти |
| Получить информацию из запроса select | Базовая информационная форма | Базовая форма для получения данных из запроса |
| Ошибка в приложении | Форма для отображения ошибок | Форма, отображающая подробную информацию об ошибках в приложении |
| Выбрать место выборки:  Цех, предприятие, участок | Базовая форма выбора | Форма с переключаемыми режимами выборки и двумя выпадающими списками, содержащие информацию о участках и цехах предприятия |

Для реализации авторизации была добавлена таблица «ACCOUNTS», содержащая логины, пароли и роли пользователей.

Глава 3. Реализация системы

Общий объем работ по программированию:

Реализация предложенной схемы данных средствами базы данных Oracle XE (в частности, с использованием утилиты SQL Plus)

Настройка удаленного репозитория GitHub

Создание приложения клиента с помощью языка программирования C#

Реализация модуля подключения к базе данных посредством Oracle data access

Реализация модуля графического интерфейса клиентского приложения

Создание форм посредством Windows forms

Реализация классов-контроллеров, отвечающих за формирование, наполнение и функционал форм

Для создания документа, описывающего схему базы данных, было выбрано приложение SQL Developer. Результатом является SQL скрипт, содержащий описание всех таблиц, триггеров и встроенных функций на языке DDL.

Пример создания таблицы, соответствующей профессии сборщик

Пример наполнения тестовыми данными таблицы «WORKERS»



Ниже будет приведена реализация подключения посредством использования Oracle data access



Целостность данных производилась посредством использования constraints и triggers на серверной части, а также генерации exception на части приложения, ниже будут представлены некоторые из решений, помогающих обеспечить целостность

Пример использования триггера для проверки принадлежности продукта к определённому цеху



Пример ограничения уникальной тройки имя фамилия отчество для рабочего посредством constraints



Пример генерации exception на стороне приложения



Обработка исключений выполнялась в блоке try, и перехватывались посредством иерархичных блоков catch для отлавливания разных видов ошибок, ниже приведен фрагмент кода демонстрирующий данный процесс



Глава 4. Тестирование

В целях тестирования был создан набор данных со следующими параметрами:

Занесены роли пользователей в таблицу «ACCOUNTS» для полной проверки программных возможностей

Все таблицы были наполнены данными, с непересекающимися связанными данными из других сущностей для вариативности

Тестирование проходило с помощью сверки правильного результата выполнений запросов с полученными в ходе выполнения программы.

Заключение

В ходе работы был выполнен анализ требований к проекту, построена ER диаграмма, являющаяся шаблоном для реализации базы данных, выполнено разделение ролей пользователей посредством предоставления программой различного функционала, а также с помощью реализации авторизации, была спроектирована база данных и написано клиентское приложение, предоставляющее удобный интерфейс для взаимодействия с данными.

Изначально был произведен анализ требований, и выделены наиболее важные сущности и атрибуты для хранения информации. Далее с помощью визуализации ER диаграммой возникла надобность в перестройке дизайна базы данных чтобы удовлетворять всем требуемым компетенциям от программы.

Следующим шагом стала написание sql скриптов для проектирования базы данных. Выполнено это было с помощью удобного сайта по созданию таблиц dbdesigner. После реализации дизайна серверной части, были добавлены ограничения целостности данных на серверной части с помощью использования триггеров и constraints.

Далее была выполнена реализация клиента. Изначально была реализована логика взаимодействия базы данных с клиентской частью приложения. После было выполнено создание форм с помощью удобного интерфейса по созданию форм Windows Forms встроенного в IDE Visual Studio.

Последующим этапом было выполнено заполнение тестовых данных с помощью приложения sql developer и выполнено тестирования, которое показало положительные результаты работы программы.

Перспективы дальнейшей работы: реализация добавления/удаления данных с помощью клиентского приложения, добавления интерактивной составляюще к интерфейсу для более дружелюбного взаимодействия с пользователем.

Источники

<https://docs.microsoft.com/ru-ru/dotnet/>

https://docs.oracle.com/en/

Приложение

Sql скрипты для создания сущностей

create table workers (

id int not null primary key,

first\_name nvarchar2(20) not null,

last\_name nvarchar2(20) not null,

patronymic nvarchar2(20) not null,

team\_id int,

constraint uniq\_name\_work unique(first\_name,last\_name,patronymic)

);

/

create table teams (

id int not null primary key,

name nvarchar2(20) not null,

lead\_id int ,

constraint unique\_lead unique(lead\_id),

constraint unique\_name\_team unique(name),

constraint fk\_team\_lead foreign key(lead\_id) references workers(id) on delete set null

);

/

create table turners (

id int not null primary key,

amount\_fingers int default 10 check(amount\_fingers>0),

work\_id int not null,

constraint fk\_turn foreign key(work\_id) references workers(id) on delete cascade,

constraint uniq\_work\_id\_turn unique(work\_id)

);

/

create table fitters (

id int not null primary key,

category int not null check(category>0),

work\_id int not null,

constraint uniq\_work\_id\_fitt unique(work\_id),

constraint fk\_fitt foreign key(work\_id) references workers(id) on delete cascade

);

/

create table welders (

id int not null primary key,

specialization nvarchar2(30) not null,

work\_id int not null,

constraint uniq\_work\_id\_weld unique(work\_id),

constraint fk\_weld foreign key(work\_id) references workers(id) on delete cascade

);

/

create table other\_workers (

id int not null primary key,

name\_prof nvarchar2(20) not null,

work\_id int not null,

constraint uniq\_work\_id\_owork unique(work\_id),

constraint fk\_ow foreign key(work\_id) references workers(id) on delete cascade

);

/

create table collectors (

id int not null primary key,

work\_exp int not null check(work\_exp>-1),

work\_id int not null,

constraint uniq\_work\_id\_coll unique(work\_id),

constraint fk\_col foreign key(work\_id) references workers(id) on delete cascade

);

/

create table products(

id int not null primary key,

workshop\_id int

);

/

create table gliders(

id int not null primary key,

wingspan float not null check(wingspan>0),

name nvarchar2(30) not null,

prod\_id int not null,

constraint uniq\_name\_glid unique(name),

constraint fk\_glid foreign key(prod\_id) references products(id) on delete cascade

);

/

create table rockets(

id int not null primary key,

power int not null check(power>0),

name nvarchar2(30) not null,

prod\_id int not null,

constraint uniq\_name\_rocket unique(name),

constraint fk\_rock foreign key(prod\_id) references products(id) on delete cascade

);

create table planes(

id int not null primary key,

count\_engines int not null check(count\_engines>0),

name nvarchar2(30) not null,

prod\_id int not null,

constraint fk\_plane foreign key(prod\_id) references products(id) on delete cascade,

constraint uniq\_name\_plane unique(name)

);

/

create table hang\_gliders(

id int not null primary key,

aero\_qual int not null check(aero\_qual>0),

name nvarchar2(30) not null,

prod\_id int not null,

constraint uniq\_name\_hg unique(name),

constraint fk\_hg foreign key(prod\_id) references products(id) on delete cascade

);

/

create table helicopters(

id int not null primary key,

speed int not null check(speed>0),

name nvarchar2(30) not null,

prod\_id int not null,

constraint uniq\_name\_heli unique(name),

constraint fk\_heli foreign key(prod\_id) references products(id) on delete cascade

);

/

create table other\_prods(

id int not null primary key,

type\_prod nvarchar2(20) not null,

name nvarchar2(30) not null,

prod\_id int not null,

constraint uniq\_name\_other\_prod unique(name),

constraint fk\_op foreign key(prod\_id) references products(id) on delete cascade

);

/

create table technics (

id int not null primary key,

first\_name nvarchar2(20) not null,

last\_name nvarchar2(20) not null,

patronymic nvarchar2(20) not null,

constraint uniq\_name\_techs unique(first\_name,last\_name,patronymic)

);

/

create table engineers (

id int not null primary key,

specialization nvarchar2(20) not null ,

tech\_id int not null,

constraint uniq\_tech\_id\_eng unique(tech\_id),

constraint fk\_eng foreign key(tech\_id) references technics(id) on delete cascade

);

/

create table technicians (

id int not null primary key,

category nvarchar2(20) not null ,

tech\_id int not null,

constraint uniq\_tech\_id\_tech unique(tech\_id),

constraint fk\_technic foreign key(tech\_id) references technics(id) on delete cascade

);

/

create table technologists (

id int not null primary key,

technology\_dev nvarchar2(40) not null ,

tech\_id int not null,

constraint uniq\_tech\_id\_technologists unique(tech\_id),

constraint fk\_technol foreign key(tech\_id) references technics(id) on delete cascade

);

/

create table workshops(

id int not null primary key,

name nvarchar2(20) not null ,

tech\_id int,

constraint fk\_ws foreign key(tech\_id) references technics(id) on delete set null,

constraint uniq\_tech\_id\_ws unique(tech\_id),

constraint unique\_ws\_name unique(name)

);

/

create table sectors(

id int not null primary key,

name nvarchar2(20) not null ,

tech\_id int,

workshop\_id int not null,

constraint fk\_sec foreign key(tech\_id) references technics(id) on delete set null,

constraint fk\_sec\_ws foreign key(workshop\_id) references workshops(id) on delete cascade,

constraint uniq\_tech\_id\_sec unique(tech\_id),

constraint unique\_sec\_name unique(name)

);

/

create table masters(

id int not null primary key,

tech\_id int not null,

sec\_id int,

constraint uniq\_tech\_id\_masters unique(tech\_id),

constraint fk\_mast foreign key(tech\_id) references technics(id) on delete cascade,

constraint fk\_mast\_sec foreign key(sec\_id) references sectors(id) on delete set null

);

/

create table jobs(

id int not null primary key,

name nvarchar2(30) not null,

sec\_id int not null,

constraint fk\_job foreign key(sec\_id) references sectors(id) on delete cascade,

constraint uniq\_job\_sec unique(sec\_id),

constraint uniq\_job\_name unique(name)

);

/

create table products\_jobs(

id int not null primary key,

job\_id int not null,

order\_num int not null check(order\_num>0),

prod\_id int not null,

time\_begin date,

time\_end date,

team\_id int not null,

constraint check\_time\_job check(time\_end>=time\_begin),

constraint unique\_prod\_job unique(prod\_id,job\_id),

constraint unique\_order\_job unique(prod\_id,order\_num),

constraint fk\_pr\_job foreign key(job\_id) references jobs(id) on delete cascade,

constraint fk\_prjob\_prod foreign key(prod\_id) references products(id) on delete cascade,

constraint fk\_team\_d foreign key(team\_id) references teams(id) on delete cascade

);

/

create table experiments(

id int not null primary key,

name nvarchar2(30) not null,

constraint uniq\_exp\_name unique(name)

);

/

create table laboratories(

id int not null primary key,

name nvarchar2(30) not null,

constraint uniq\_lab\_name unique(name)

);

/

create table tools(

id int not null primary key,

lab\_id int not null,

name nvarchar2(20) not null,

constraint uniq\_tool\_name unique(name),

constraint fk\_tools foreign key(lab\_id) references laboratories(id) on delete cascade

);

/

create table exp\_tools(

id int not null primary key,

exp\_id int not null,

tool\_id int not null,

constraint fk\_exp\_tools\_exp foreign key(exp\_id) references experiments(id) on delete cascade,

constraint unique\_exp\_tool unique(exp\_id,tool\_id),

constraint fk\_exp\_tools foreign key(tool\_id) references tools(id) on delete cascade

);

/

create table products\_exps(

id int not null primary key,

exp\_id int not null,

order\_num int not null check(order\_num>0),

prod\_id int not null,

time\_begin date,

time\_end date,

constraint check\_time\_exp check(time\_end>=time\_begin),

constraint unique\_prod\_exp unique(prod\_id,exp\_id),

constraint unique\_order\_exp unique(prod\_id,order\_num),

constraint fk\_prjexp\_prod foreign key(prod\_id) references products(id) on delete cascade,

constraint fk\_pr\_exp foreign key(exp\_id) references experiments(id) on delete cascade

);

/

alter table workers add constraint fk\_work\_team foreign key(team\_id) references teams(id) on delete set null;

/

create table investigators (

id int not null primary key,

first\_name nvarchar2(20) not null,

last\_name nvarchar2(20) not null,

patronymic nvarchar2(20) not null,

exp\_id int,

constraint fk\_invest foreign key(exp\_id) references experiments(id) on delete set null,

constraint uniq\_name\_inv unique(first\_name,last\_name,patronymic)

);

/

create table lab\_sup\_ws (

id int not null,

lab\_id int not null,

ws\_id int not null,

constraint fk\_lab\_ws foreign key(ws\_id) references workshops(id) on delete cascade,

constraint unique\_lab\_ws unique(lab\_id,ws\_id),

constraint fk\_lab\_lab foreign key(lab\_id) references laboratories(id) on delete cascade

);

/

create table Accounts(

id int not null primary key,

login NVARCHAR2(20) not null,

pas NVARCHAR2(20) not null,

role NVARCHAR2(40) not null,

constraint unique\_acc unique(login,pas)

);

Создание триггеров и авто инкрементов

CREATE OR REPLACE TRIGGER unique\_welders

before

UPDATE OR insert

ON welders

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.work\_id<>:old.work\_id or :old.work\_id is null then

select count(\*) into c from collectors where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from turners where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from fitters where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from other\_workers where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_col

before

UPDATE OR insert

ON collectors

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.work\_id<>:old.work\_id or :old.work\_id is null then

select count(\*) into c from welders where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from turners where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from fitters where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from other\_workers where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_turners

before

UPDATE OR insert

ON turners

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.work\_id<>:old.work\_id or :old.work\_id is null then

select count(\*) into c from welders where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from collectors where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from fitters where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from other\_workers where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_fitters

before

UPDATE OR insert

ON fitters

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.work\_id<>:old.work\_id or :old.work\_id is null then

select count(\*) into c from welders where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from collectors where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from turners where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from other\_workers where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_other\_workers

before

UPDATE OR insert

ON other\_workers

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.work\_id<>:old.work\_id or :old.work\_id is null then

select count(\*) into c from welders where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from collectors where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from turners where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

select count(\*) into c from fitters where work\_id=:new.work\_id;

if c!=0 then

raise\_application\_error(-20000,'worker has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER ins\_other\_prods

before update or insert

ON other\_prods

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :old.name<>:new.name then

select count(\*) into c from gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from rockets where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from planes where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from hang\_gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from helicopters where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

cREATE OR REPLACE TRIGGER ins\_gliders

before update or insert

ON gliders

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :old.name<>:new.name then

select count(\*) into c from other\_prods where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from rockets where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from planes where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from hang\_gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from helicopters where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

cREATE OR REPLACE TRIGGER ins\_rockets

before update or insert

ON rockets

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :old.name<>:new.name then

select count(\*) into c from gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from other\_prods where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from planes where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from hang\_gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from helicopters where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

cREATE OR REPLACE TRIGGER ins\_planes

before update or insert

ON planes

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :old.name<>:new.name then

select count(\*) into c from gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from other\_prods where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from rockets where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from hang\_gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from helicopters where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

CREATE OR REPLACE TRIGGER ins\_hang\_gliders

before update or insert

ON hang\_gliders

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :new.name<>:new.name then

select count(\*) into c from gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from other\_prods where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from rockets where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from planes where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from helicopters where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

CREATE OR REPLACE TRIGGER ins\_helicopters

before update or insert

ON helicopters

FOR EACH ROW

DECLARE

c int;

id int;

BEGIN

if :old.name<>:new.name then

select count(\*) into c from gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from other\_prods where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from rockets where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from planes where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

select count(\*) into c from hang\_gliders where name=:new.name;

if c!=0 then

raise\_application\_error(-20000,'not unique name');

end if;

end if;

if :new.prod\_id is null then

select seq\_products.nextval into id from dual;

insert into products(id) values(id);

:new.prod\_id:=id;

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_other\_prods

before update

ON other\_prods

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_gliders

before update

ON gliders

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_rockets

before update

ON rockets

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_planes

before update

ON planes

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_hang\_gliders

before update

ON hang\_gliders

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_ls\_helicopters

before update

ON helicopters

FOR EACH ROW

BEGIN

if :old.prod\_id<>:new.prod\_id then

raise\_application\_error(-20000,'could not update prod\_id');

end if;

end;

/

CREATE OR REPLACE TRIGGER add\_team\_id

after insert

ON teams

FOR EACH ROW

DECLARE

c int;

BEGIN

select count(\*) into c from workers where id=:new.lead\_id and team\_id is not null;

if c<>0 then

raise\_application\_error(-20000,'worker has other team');

else

update workers set team\_id=:new.id where id=:new.lead\_id;

end if;

end;

/

CREATE OR REPLACE TRIGGER add\_lead\_id

after insert

ON workers

FOR EACH ROW

DECLARE

c int;

BEGIN

select count(\*) into c from teams where id=:new.team\_id and lead\_id is null;

if c<>0 then

update teams set lead\_id=:new.id where id=:new.team\_id;

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_team\_id

after update

ON teams

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.lead\_id<>:old.lead\_id then

select count(\*) into c from workers where id=:new.lead\_id and team\_id is not null and team\_id<>:new.id;

if c<>0 then

raise\_application\_error(-20000,'worker has other team');

else

update workers set team\_id= case id

when :new.lead\_id then :new.id

when :old.lead\_id then null

else team\_id end

where id in(:new.lead\_id,:old.lead\_id);

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER update\_lead\_id

before update

ON workers

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.team\_id<>:old.team\_id then

select count(\*) into c from teams where id=:new.team\_id and lead\_id is null;

if c<>0 then

update teams set lead\_id=:new.id where id=:new.team\_id;

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_technicians

before

UPDATE OR insert

ON technicians

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from engineers where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

select count(\*) into c from technologists where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_engineers

before

UPDATE OR insert

ON engineers

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from technicians where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

select count(\*) into c from technologists where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_technologists

before

UPDATE OR insert

ON technologists

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from technicians where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

select count(\*) into c from engineers where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_masters

before

UPDATE OR insert

ON masters

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from workshops where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has other job');

end if;

select count(\*) into c from sectors where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has job');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_workshops\_lead

before

UPDATE OR insert

ON workshops

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from masters where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

select count(\*) into c from sectors where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_sectors\_leads

before

UPDATE OR insert

ON sectors

FOR EACH ROW

DECLARE

c int;

BEGIN

if :new.tech\_id<>:old.tech\_id or :old.tech\_id is null then

select count(\*) into c from workshops where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

select count(\*) into c from masters where tech\_id=:new.tech\_id;

if c!=0 then

raise\_application\_error(-20000,'technic has profession');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER unique\_ws\_prod

before

UPDATE OR insert

ON products\_jobs

FOR EACH ROW

DECLARE

id1 int;

id2 int;

BEGIN

if :new.prod\_id<>:old.prod\_id or :new.job\_id<>:old.job\_id or :old.job\_id is null then

select workshop\_id into id1 from sectors where id in (select sec\_id from jobs where id=:new.job\_id);

select workshop\_id into id2 from products where id in :new.prod\_id;

if id1<>id2 then

raise\_application\_error(-20000,'cannot define workshop for product');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER ver\_lab\_sup\_ws

before

UPDATE OR insert

ON products\_exps

FOR EACH ROW

DECLARE

id1 int;

BEGIN

if :new.prod\_id<>:old.prod\_id or :new.exp\_id<>:old.exp\_id or :old.prod\_id is null then

select count(\*) into id1 from

(select workshop\_id from products where id = :new.prod\_id)

where workshop\_id in

(select ws\_id from lab\_sup\_ws

where lab\_id in

(select lab\_id from tools

where id in

(select tool\_id from exp\_tools

where exp\_id=:new.exp\_id)));

if id1=0 then

raise\_application\_error(-20000,'workshop does not supports with laboratories');

end if;

end if;

end;

/

CREATE OR REPLACE TRIGGER check\_ws

before

UPDATE OR insert

ON products\_jobs

FOR EACH ROW

DECLARE

id1 int;

BEGIN

select count(\*) into id1 from

products where id=:new.prod\_id and workshop\_id is not null;

if id1=0 then

raise\_application\_error(-20000,'please setup workshop for product before');

end if;

end;

/

CREATE sequence seq\_workers START WITH 1;

CREATE sequence seq\_turners START WITH 1;

CREATE sequence seq\_fitters START WITH 1;

CREATE sequence seq\_welders START WITH 1;

CREATE sequence seq\_collectors START WITH 1;

CREATE sequence seq\_products START WITH 1;

CREATE sequence seq\_gliders START WITH 1;

CREATE sequence seq\_rockets START WITH 1;

CREATE sequence seq\_planes START WITH 1;

CREATE sequence seq\_hang\_gliders START WITH 1;

CREATE sequence seq\_helicopters START WITH 1;

CREATE sequence seq\_other\_prods START WITH 1;

CREATE sequence seq\_other\_workers START WITH 1;

CREATE sequence seq\_engineers START WITH 1;

CREATE sequence seq\_technics START WITH 1;

CREATE sequence seq\_technicians START WITH 1;

CREATE sequence seq\_workshops START WITH 1;

CREATE sequence seq\_sectors START WITH 1;

CREATE sequence seq\_masters START WITH 1;

CREATE sequence seq\_technologists START WITH 1;

CREATE sequence seq\_jobs START WITH 1;

CREATE sequence seq\_products\_jobs START WITH 1;

CREATE sequence seq\_experiments START WITH 1;

CREATE sequence seq\_laboratories START WITH 1;

CREATE sequence seq\_tools START WITH 1;

CREATE sequence seq\_exp\_tools START WITH 1;

CREATE sequence seq\_products\_exps START WITH 1;

CREATE sequence seq\_lab\_sup\_ws START WITH 1;

CREATE sequence seq\_teams START WITH 1;

CREATE sequence seq\_investigators START WITH 1;

CREATE sequence seq\_accounts START WITH 1;

CREATE OR REPLACE TRIGGER next\_account

BEFORE INSERT ON accounts

FOR EACH ROW

BEGIN

SELECT seq\_accounts.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_gliders

BEFORE INSERT ON gliders

FOR EACH ROW

BEGIN

SELECT seq\_gliders.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_workers

BEFORE INSERT ON workers

FOR EACH ROW

BEGIN

SELECT seq\_workers.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_turners

BEFORE INSERT ON turners

FOR EACH ROW

BEGIN

SELECT seq\_turners.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_fitters

BEFORE INSERT ON fitters

FOR EACH ROW

BEGIN

SELECT seq\_fitters.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_welders

BEFORE INSERT ON welders

FOR EACH ROW

BEGIN

SELECT seq\_welders.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_collectors

BEFORE INSERT ON collectors

FOR EACH ROW

BEGIN

SELECT seq\_collectors.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_rockets

BEFORE INSERT ON rockets

FOR EACH ROW

BEGIN

SELECT seq\_rockets.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_hang\_gliders

BEFORE INSERT ON hang\_gliders

FOR EACH ROW

BEGIN

SELECT seq\_hang\_gliders.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_planes

BEFORE INSERT ON planes

FOR EACH ROW

BEGIN

SELECT seq\_planes.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_helicopters

BEFORE INSERT ON helicopters

FOR EACH ROW

BEGIN

SELECT seq\_helicopters.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_other\_prods

BEFORE INSERT ON other\_prods

FOR EACH ROW

BEGIN

SELECT seq\_other\_prods.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_other\_workers

BEFORE INSERT ON other\_workers

FOR EACH ROW

BEGIN

SELECT seq\_other\_workers.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_engineers

BEFORE INSERT ON engineers

FOR EACH ROW

BEGIN

SELECT seq\_engineers.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_technicians

BEFORE INSERT ON technicians

FOR EACH ROW

BEGIN

SELECT seq\_technicians.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_technics

BEFORE INSERT ON technics

FOR EACH ROW

BEGIN

SELECT seq\_technics.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_workshops

BEFORE INSERT ON workshops

FOR EACH ROW

BEGIN

SELECT seq\_workshops.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_sectors

BEFORE INSERT ON sectors

FOR EACH ROW

BEGIN

SELECT seq\_sectors.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_masters

BEFORE INSERT ON masters

FOR EACH ROW

BEGIN

SELECT seq\_masters.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_technologists

BEFORE INSERT ON technologists

FOR EACH ROW

BEGIN

SELECT seq\_technologists.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_jobs

BEFORE INSERT ON jobs

FOR EACH ROW

BEGIN

SELECT seq\_jobs.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_products\_jobs

BEFORE INSERT ON products\_jobs

FOR EACH ROW

BEGIN

SELECT seq\_products\_jobs.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_experiments

BEFORE INSERT ON experiments

FOR EACH ROW

BEGIN

SELECT seq\_experiments.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_laboratories

BEFORE INSERT ON laboratories

FOR EACH ROW

BEGIN

SELECT seq\_laboratories.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_tools

BEFORE INSERT ON tools

FOR EACH ROW

BEGIN

SELECT seq\_tools.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_exp\_tools

BEFORE INSERT ON exp\_tools

FOR EACH ROW

BEGIN

SELECT seq\_exp\_tools.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_products\_exps

BEFORE INSERT ON products\_exps

FOR EACH ROW

BEGIN

SELECT seq\_products\_exps.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_lab\_sup\_ws

BEFORE INSERT ON lab\_sup\_ws

FOR EACH ROW

BEGIN

SELECT seq\_lab\_sup\_ws.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_teams

BEFORE INSERT ON teams

FOR EACH ROW

BEGIN

SELECT seq\_teams.NEXTVAL

INTO :new.id

FROM dual;

end;

/

CREATE OR REPLACE TRIGGER next\_investigators

BEFORE INSERT ON investigators

FOR EACH ROW

BEGIN

SELECT seq\_investigators.NEXTVAL

INTO :new.id

FROM dual;

end;