МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 05 01 Информационные системы

и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных железнодорожного вокзала с использованием технологии репликация данных между серверами СУБД»

Выполнил студент Злобин Р.Ю.

(Ф.И.О.)

Руководитель проекта доцент Е.А Блинова

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: доцент Е.А Блинова.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: доцент Е.А Блинова.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2021

**Содержание**

[Введение 4](#_Toc72834704)

[1. Аналитический обзор аналогов 5](#_Toc72834705)

[1.1 Web-приложение «БЧ» 5](#_Toc72834706)

[1.2 Web-приложение «ticketsby» 6](#_Toc72834707)

[2. Анализ требований к программному средству и разработка функциональных требований 7](#_Toc72834708)

[3. Разработка архитектуры проекта 8](#_Toc72834709)

[3.1 Обобщенная структура управления 8](#_Toc72834710)

[3.2 Проектирование модели базы данных 9](#_Toc72834711)

[4. Использование Oracle 12с 13](#_Toc72834712)

[4.1 Создание таблиц 13](#_Toc72834713)

[4.2 Создание ролей для разграничения доступа 13](#_Toc72834714)

[4.3 Создание пакетов и процедур для работы с базой данных 13](#_Toc72834715)

[4.4 Создание автоматизированных заданий 19](#_Toc72834716)

[4.5 Описание процедур экспорта и импорта 19](#_Toc72834717)

[4.6 Тестирование производительности базы данных 20](#_Toc72834718)

[5. Описание технологии репликации 23](#_Toc72834719)

[6. Руководство по использованию программного средства 27](#_Toc72834720)

[6.1 Область клиента 27](#_Toc72834721)

[6.2 Область менеджера 27](#_Toc72834722)

[Заключение 28](#_Toc72834723)

[Cписок источников 29](#_Toc72834724)

[Приложение А 30](#_Toc72834725)

[Приложение Б 31](#_Toc72834726)

[Приложение В 37](#_Toc72834727)

[Приложение Г 39](#_Toc72834728)

[Приложение Д 44](#_Toc72834729)

[Приложение Е 50](#_Toc72834730)

[Приложение Ж 54](#_Toc72834731)

[Приложение З 56](#_Toc72834732)

[Приложение И 59](#_Toc72834733)

[Приложение И 60](#_Toc72834734)

[Приложение К 61](#_Toc72834735)

# **Введение**

Данный курсовой проект посвящён разработке базы данных для службы железнодорожного вокзала и технологии репликации данных между серверами СУБД.

Цель разработки базы данных – спроектировать максимально удобную для встраивания в программное средство базу данных, внутри которой будет осуществлена вся работа с таблицами с помощью пакетов и хранимых процедур.

Данная база данных должна находится на удалённом сервере и быстро разворачиваться на любом другом сервере, посредством последовательного выполнения скриптов.

В базе данных должны присутствовать разграничения для различных пользователей, таких как администратор, менеджер и пользователь. Это было сделано для того, чтобы пользователь не смог повредить базу данных.

Технология репликации данных используется для синхронизации состояния объектов баз данных. Среди задач, решаемых репликацией, можно отметить: поддержку резервной базы данных на случай потери основной, снижение нагрузки на базу за счёт переноса части запросов на реплики, перенос данных в архивные или аналитические системы.

# **1. Аналитический обзор аналогов**

В наши дни множество аналогов программных средств можно найти на различных веб-ресурсах, связанных с данной тематикой. Перед тем как приступить к разработке требовалось провести анализ существующих аналогов.

1.1 Web-приложение «БЧ»

На рисунке 1.1 предствлена главная страница web-приложения «БЧ».

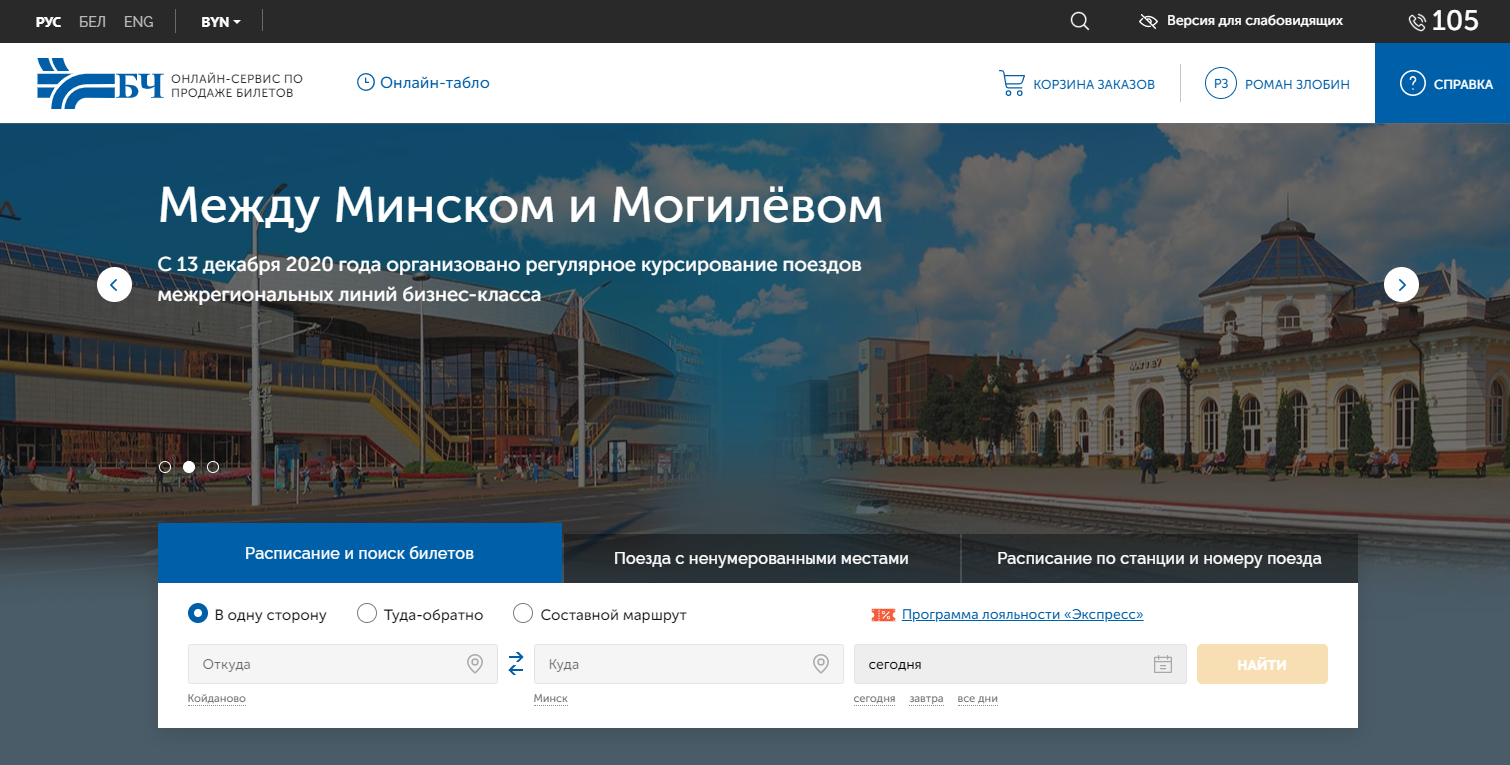


Рисунок 1.1 – Страница аналога web-приложения «БЧ»

Ознакомившись с программных средством, был сделан вывод об основных функциональных особенностях:

* возможность просматривать расписание;
* возможность осуществлять заказы билетов;
* возможность просматривать онлайн-табло станции;
* возможность регистрации и авторизации;
* возможность добавлять маршруты в избранное;
* возможность добавлять поезда в избранное;
* возможность добавлять пассажиров.

1.2 Web-приложение «ticketsby»

На рисунке 1.2 представлено главное окно приложения «системы автоматизации сервисного центра».

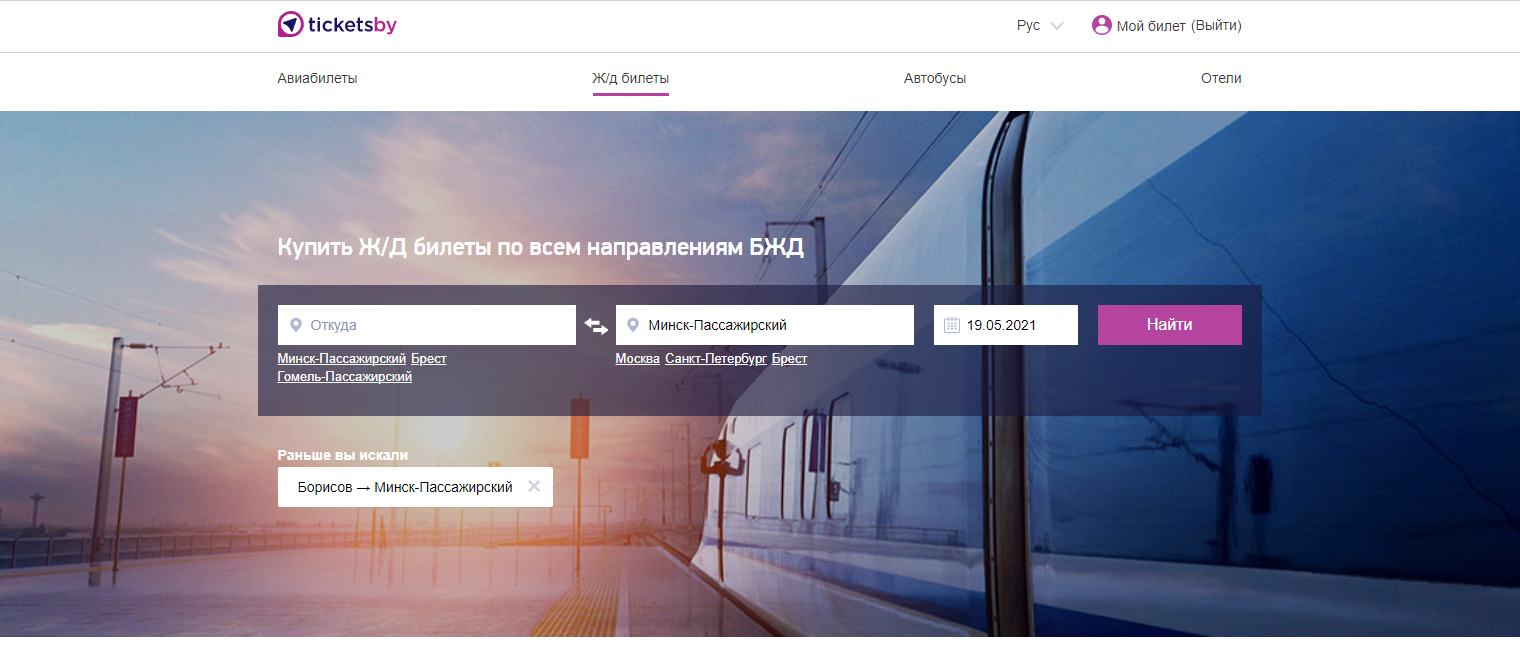


Рисунок 1.2 – Страница аналога web-приложения «ticketsby»

Ознакомившись с программных средством, был сделан вывод об основных функциональных особенностях:

* возможность просматривать расписание;
* возможность осуществлять заказы билетов;
* возможность регистрации и авторизации;
* возможность добавлять пассажиров;
* промокоды и система бонусов.

# **2. Анализ требований к программному средству и разработка функциональных требований**

Проанализировав соответствующие аналоги по теме, были выдвинуты следующие требования:

* разрабатываемая база данных должна иметь авторизацию для работников и клиентов;
* просмотр расписания пользователем;
* оформление заказа пользователем;
* онлайн-табло определённых железнодорожных вокзалов;
* создание маршрутов менеджерами;
* уведомления для пользователей.

В качестве СУБД была выбрана Oracle Database 12c.

Для обеспечения повышенной отказоустойчивости и повышения надёжности системы было принято решение реплицировать базу данных на дополнительный сервер. Данное решение позволит поддерживать актуальную резервную копию базы данных, снизить нагрузку на основную систему, в случае необходимости, переносить данные в архивные и аналитические системы.

# **Разработка архитектуры проекта**

## **Обобщенная структура управления**

В структуре управления имеется разделение пользователей на клиентов и менеджеров. Роль администратора как таковая не реализуется, но её наличие необходимо и возлагается на администратора базы данных.

Для наглядности демонстрации возможностей пользователей воспользуемся UML-диаграммами. UML – уникальный язык моделирования (Unified Modeling Language)-это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем [2].

Диаграмма вариантов использования программного средства клиентом представлена на рисунке 3.1.

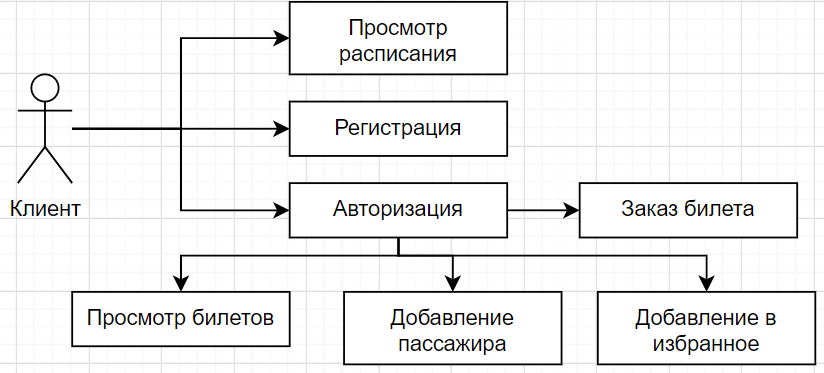


Рисунок 3.1 – Диаграмма использования пс клиентом

Диаграмма вариантов использования приложения для работника представлена на рисунке 3.2.

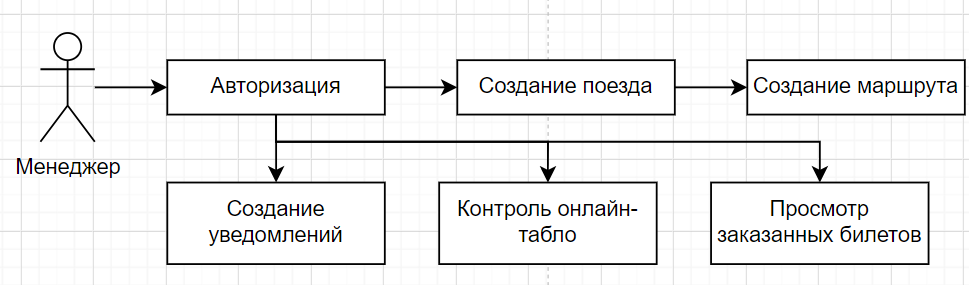


Рисунок 3.2 – Диаграмма использования пс для работника

Клиенту сразу предоставляется возможность авторизоваться, зарегистрироваться и просмотреть расписание. Авторизированный клиент может оформлять заказы билетов, регистрировать пассажиров, просматривать свои билеты, взаимодействовать со своей коллекцией избранного.

Менеджеру необходимо авторизоваться для взаимодействия с системой. После ему предоставляется возможность работать со списком поездов и маршрутов, заказанных билетов. Так же на менеджера ложиться ответственность контролировать онлайн табло и уведомления для клиентов.

Область работы администратора – Sql Developer или SqlPlus. Он ответственен за инициализацию, контроль и мониторинг информационной системы.

## **Проектирование модели базы данных**

Первым этапом выполнения курсовой работы будет создание логически взаимосвязанных таблиц. Чтобы составить визуальную взаимосвязанную структуры нашей базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

База данных состоит из 16 таблиц, схема который представлена в приложении А.

Далее приведена их структура и описание.

Таблица 3.1 – Структура таблицы «User\_Table»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | Id пользователя |
| NAME | nvarchar2 | Имя пользователя |
| SURNAME | nvarchar2 | Фамилия пользователя |
| FATHERNAME | nvarchar2 | Отчество пользователя |
| EMAIL | nvarchar2 | Электронная почта пользователя |
| SEX | nchar | Пол пользователя |
| BIRTHDAY | date | Дата рождения пользователя |
| PASSWORD | raw | Пароль пользователя, хеширован |

Таблица «Tokens» содержит уникальный идентификатор сессии пользователя, представленный шестнадцати битным токеном. Описание ее полей приведено в таблице 3.2.

Таблица 3.2 – Структура таблицы «Tokens»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | ID токена |
| USERID | nvarchar2 | ID пользователя |
| TOKEN | nvarchar2 | Токен |

Таблица «Doc\_Type» содержит информацию о типе документа пассажира, на которого оформляется билет. Структура представлена в таблице 3.3.

Таблица 3.3 – Структура таблицы «Doc\_Type»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | ID документа |
| TITLE | nvarchar2 | Наименование документа |
| REGEX | nvarchar2 | Регулярное выражение для проверки номера документа |

Таблица «Passenger» содержит описание пассажира, на которого оформляется билет. Структура представлена в таблице 3.4.

Таблица 3.4 – Структура таблицы «Equipment»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | Id пассажира |
| NAME | nvarchar2 | Имя пассажира |
| SURNAME | nvarchar2 | Фамилия пассажира |
| FATHERNAME | nvarchar2 | Отчество пассажира |
| SEX | nchar | Пол пассажира |

Таблица 3.4 – Структура таблицы «Equipment

|  |  |  |
| --- | --- | --- |
| BIRTHDAY | date | Дата рождения пассажира |
| PHONENUMBER | nvarchar2 | Номер телефона пассажира |
| DOCTYPE | NUMBER | ID типа документа |
| DOCNUM | nvarchar2 | Номер документа |

Таблица «Order\_Status» содержит информацию об статусе заказа. Структура представлена в таблице 3.5.

Таблица 3.5 – Структура таблицы «Order\_Status»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | ID типа заказа |
| STATUS\_NAME | nvarchar2 | Наименование статуса |
| STATUS\_DECRIPTION | nvarchar2 | Описание статуса |

Таблица «Order\_Table» содержит информацию о заказах пользователя. Структура представлена в таблице 3.6.

Таблица 3.6 – Структура таблицы «Order\_Table»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | raw | ID заказа |
| USERID | number | Id исполнителя |
| ORDERDATE | date | Дата оформления заказа |
| STATUS | number | ID статуса заказа |

Таблица «Destpoint» содержит информацию о пунктах остановки. Структура представлена в таблице 3.7.

Таблица 3.7 – Структура таблицы «Destpoint»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | NUMBER | Id пункта остановки |
| NAME | nvarchar2 | Наименование пункта остановки |
| DESCRIPTION | nvarchar2 | Описание пункта остановки |

Таблица «Route» содержит информацию о маршрутах. Структура представлена в таблице 3.8.

Таблица 3.8 – Структура таблицы «Route»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | Id маршрута |
| ADDITIONAL\_INFO | nvarchar2 | Информация о маршруте |

Таблица «Train\_Type» содержит основную информацию о типах поездов. Структура представлена в таблице 3.9.

Таблица 3.9 – Структура таблицы «Train\_Type»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | ID типа поезда |
| DESCRIPTION | number | Описание типа поезда |

Таблица «Train» содержит основную информацию о типах поездов. Структура представлена в таблице 3.10.

Таблица 3.10 – Структура таблицы «Train»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | ID поезда |
| “NUMBER” | decimal(5, 0) | Номер поезда |
| TRAIN\_TYPE | number | ID типа поезда |
| ROUTE | number | ID маршрута |

Таблица «DS\_Links» содержит основную информацию о переходах между остановочными пунктами между остановочными пунктами в маршруте, представляет собой связный список. Структура представлена в таблице 3.11.

Таблица 3.11 – Структура таблицы «DS\_Links»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | ID перехода |
| ROUTE | number | ID маршрута |
| NEXT | number | ID следующего перехода |
| DESTPOINT | number | ID следующего остановочного пункта |
| COST | decimal(5,2) | Стоимость перехода |
| TIMEMINUT | number | Время перехода |

Таблица «Ticket» содержит основную информацию о приобретаемом пользователем билете. Структура представлена в таблице 3.12.

Таблица 3.12 – Структура таблицы «Ticket»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | raw | ID билета |
| ORDERID | raw | ID заказа |
| PASSENGER | number | ID пассажира |
| TRAIN | number | ID поезда |
| “FROM” | number | ID пункта выезда |
| “TO” | number | ID пункта прибытия |
| COST | decimal(5,2) | Стоимость билета |
| CARIGENUM | decimal(2,0) | Номер вагона |
| SEATNUM | decimal(3,0) | Номер места |

Таблица «Favorite\_Path» содержит информацию о избранных маршрутах пользователя. Структура представлена в таблице 3.13.

Таблица 3.13 – Структура таблицы «Favorite\_Path»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | raw | ID билета |
| USERID | raw | ID пользователя |
| “FROM” | number | ID пункта выезда |
| “TO” | number | ID пункта прибытия |

Таблица «Favorite\_Train» содержит информацию о избранных маршрутах пользователя. Структура представлена в таблице 3.14.

Таблица 3.14 – Структура таблицы «Favorite\_Train»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | raw | ID билета |
| USERID | raw | ID пользователя |
| TRAINID | number | ID поезда |

Таблица «Notification» содержит информацию об оповещениях пользователя. Структура представлена в таблице 3.14.

Таблица 3.15 – Структура таблицы «Notification»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | ID оповещения |
| MESSAGE | nvarchar2 | Текст оповещения |
| START\_DATE | date | Начало оповещения |
| END\_DATE | date | Конец оповещения |

Таблица «Online\_Table» содержит информацию об онлайн табло для пользователя. Структура представлена в таблице 3.14.

Таблица 3.16 – Структура таблицы «Online\_Table»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| ID | number | ID строки табло |
| DESTPOINT | number | ID пункта остановки |
| TRAIN | number | ID поезда |
| ARRIVE | date | Время прибытия |
| DEPART | date | Время отбытия |

# **Использование Oracle** **12с**

## **Создание таблиц**

Для работы в базе данных мы должны создать таблицы и организовать между ними связи, которые будут представлять логическую структуру БД. В нашу базу будут включены такие таблицы:

* DESTPOINT;
* DOC\_TYPE;
* FAVORITE\_PATH;
* NOTIFICATION;
* ONLINE\_TABLE;
* ORDER\_STATUS;
* ORDER\_TABLE;
* PASSENGER;
* ROUTE;
* TICKET;
* TOKENS;
* TRAIN;
* TRAIN\_TYPE;
* USER\_TABLE.

Создание этих таблиц проходит исполнение команд блока PL/SQL.

Все таблицы создаются в отдельном файле CreateTables.sql, для дальнейшей возможности запускать скрипт в целом файле в новой базе данных.

## **Создание ролей для разграничения доступа**

Для правильной работы десктоп приложения с базой данных необходимо разграничение ролей. Для выполнения данной задачи в СУБД Oracle предусмотрены встроенные функции, такие как пользователи, роли и привилегии.

В данном случае были созданы два пользователя и им было выдано право на выполнение процедур из пакета (рисунок 4.1).

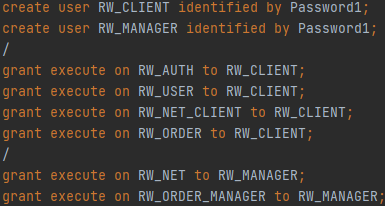


Рисунок 4.1 – Создание пользователей и выдача им прав

## **Создание пакетов и процедур для работы с базой данных**

После добавления ролей и создания всех необходимых таблиц приступаем к созданию процедур, позволяющих работать с данными.

Пакет Oracle PL/SQL – это объект схемы, который группирует логически связанные типы, элементы и подпрограммы. Пакеты обычно состоят из двух частей: спецификации и тела, хотя иногда тело не нужно.

Хранимая процедура (stored procedure) – это программа, которая выполняет некоторые действия с информацией в базе данных и при этом сама хранится в базе данных.

Для логического разделения процедур были созданы следующие пакеты:

* RW\_AUTH – пакет процедур авторизации для клиента;
* RW\_USER – пакет процедур работы с профилем для клиента;
* RW\_NET – пакет процедур работы с железнодорожной сетью для менеджера;
* RW\_NET\_CLIENT – пакет процедур работы с железнодорожной сетью для клиента;
* RW\_ORDER – пакет процедур для работы с заказами для клиента;
* RW\_ORDER\_MANAGER – пакет процедур для работы с заказами для менеджера;
* RW\_TYPES – пакет процедур для работы с таблицами констант для администратора;
* RW\_NOTIFICATION – пакет процедур для работы с оповещениями клиентов для менеджера;
* RW\_NOTIFICATION\_CLIENT – пакет процедур для работы с оповещениями клиентов для клиента;
* RW\_ONLINE\_TABLE – пакет процедур для работы с онлайн табло для менеджера;
* RW\_ONLINE\_TABLE\_CLIENT – пакет процедур для работы с онлайн табло для клиента;
* RW\_XML – пакет процедур для импорта и экспорта данных из xml документов для администратора.

Все пакеты процедур находятся во владении схемы администратора RW\_ADMIN.

Заголовок пакета, содержащий авторизации для клиента представлен на рисунке 4.3.1.

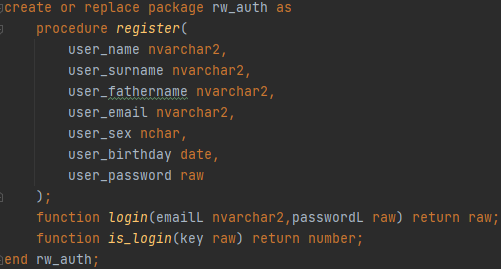


Рисунок 4.3.1 – Заголовок пакета RW\_AUTH

Данный пакет содержит процедуру register, отвечающая за регистрацию клиента в системе, функцию login для авторизации пользователя в системе и функцию is\_login проверяющая ключ авторизации пользователя.

Заголовок пакета, содержащий процедуры работы с железнодорожной сетью для менеджера представлен на рисунке 4.3.2.

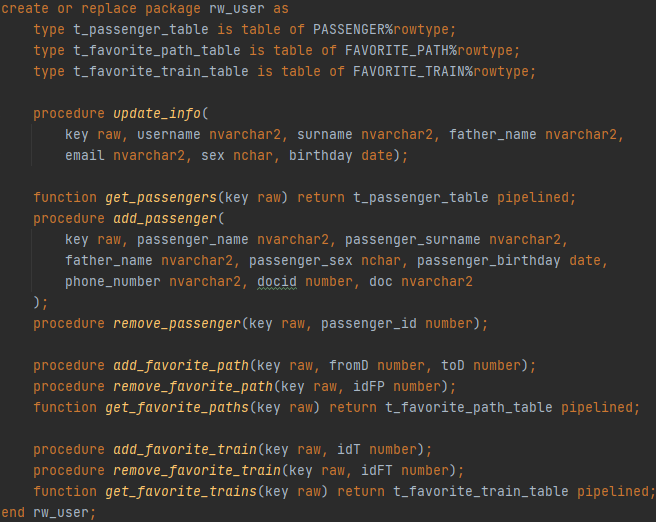


Рисунок 4.3.2 – Заголовок пакета RW\_USER

Пакет RW\_USER содержит процедуру update\_info для обновления информации о клиенте, процедуры add\_passenger и remove\_passenger для работы с пассажирами клиента, add\_favorite\_path и remove\_favorite\_path для работы с избранными маршрутами пользователя. Для получения информации в пакете предусмотрены конвейерные функции (pipeline functions).

Конвейерными функциями называются табличные функции, которые возвращают данные в виде коллекции, но делают это асинхронно, то есть получена одна запись коллекции и сразу же эта запись отдается в вызывающий код, в котором она сразу же обрабатывается.

Пакет содержит конвейерные функции get\_passengers, get\_favorite\_path, get\_fovorite\_trains для получения информации о пассажирах, избранных маршрутах и поездах клиента.

Следует обратить внимание, что все операции в пакете требуют ключ авторизации key, который можно получить только функцией login пакета RW\_AUTH.

Заголовок пакета, содержащий процедуры для работы администратора БД представлен на рисунке 4.3.3.

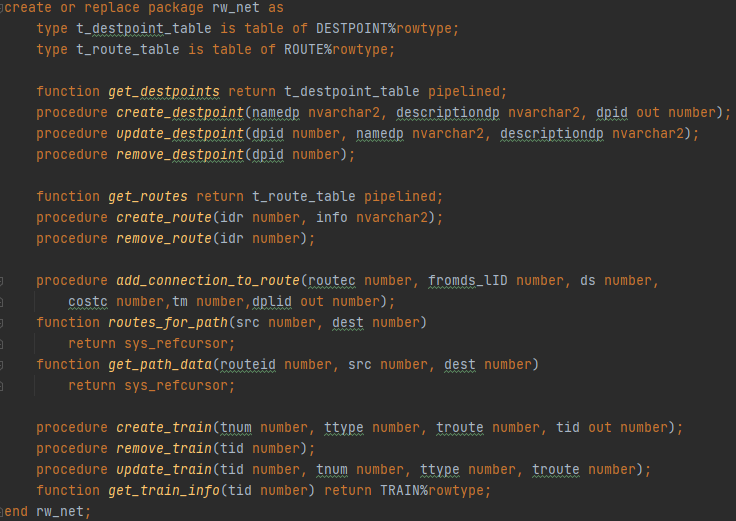


Рисунок 4.3.3 – Заголовок пакета RW\_NET

Пакет RW\_NET содержит полный набор операций для пунктов остановки, поездов и маршрутов, реализованных схожим образом как для пакета RW\_USER. Отдельно следует отметить функции routes\_for\_path и get\_path\_data, который производят поиск маршрутов в сети и информации об отрезке конкретного маршрута. Так как данный пакет предназначен исключительно для менеджера, был создан пакет RW\_NET\_CLIENT, который содержит только операции для получения информации, заголовок которого представлен на рисунке 4.3.4.

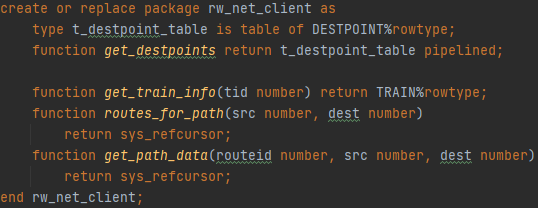


Рисунок 4.3.4 – Заголовок пакета RW\_NET\_CLIENT

Заголовок пакета, содержащий процедуры для работы с заказами для клиента представлен на рисунке 4.3.5.

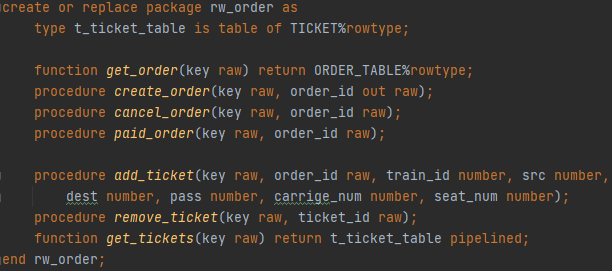


Рисунок 4.3.5 – Заголовок пакета RW\_ORDER

Пакет RW\_ORDER содержит методы создания, отмены, оплаты и получения заказа. Подразумевается, что активным будет только один заказ. Кроме того, оплаченные заказы не подлежат удалению. Пакет также содержит методы добавления, удаления билетов из заказа, а также конвейерную функцию получения всех билетов клиента.

Так как данный пакет предназначен только для клиентов было принято решение создать пакет RW\_ORDER\_MANAGER для менеджера, заголовок которого представлен на рисунке 4.3.6.

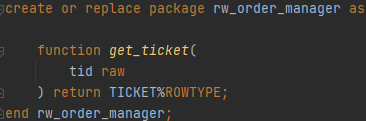


Рисунок 4.3.6 – Заголовок пакета RW\_ORDER\_MANAGER

Данный пакет содержит метод get\_ticket для получения информации о конкретном билете.

Заголовок пакета, содержащий процедуры для таблицами-константами для администратора представлен на рисунке 4.3.7.

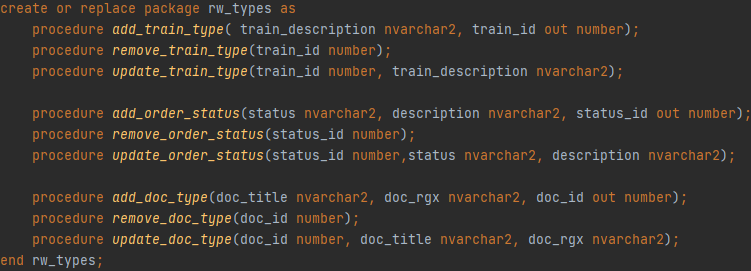


Рисунок 4.3.7 – Заголовок пакета RW\_TYPES

Пакет RW\_TYPES содержит полный набор операций для таблиц-констант.

Заголовок пакета, содержащий процедуры для работы с оповещениями клиентов для менеджера представлен на рисунке 4.3.8.

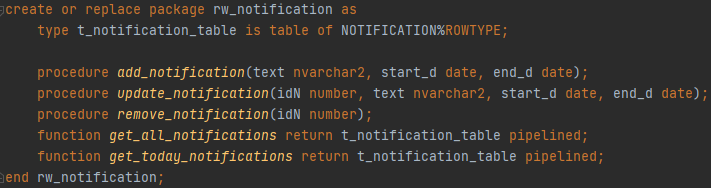


Рисунок 4.3.8 – Заголовок пакета RW\_NOTIFICATION

Пакет RW\_NOTIFICATION содержит методы создания, отмены, обновления оповещения клиента. Так же объявлены конвейерные функции для получения всех и актуальных оповещений.

Так как данный пакет предназначен только для менеджера было принято решение создать пакет RW\_NOTIFICATION\_CLIENT для клиента, заголовок которого представлен на рисунке 4.3.9.

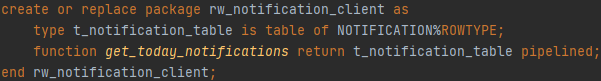


Рисунок 4.3.9 – Заголовок пакета RW\_NOTIFICATION\_CLIENT

Данный пакет содержит единственный метод, получение актуальных оповещений.

Заголовок пакета, содержащий процедуры для работы с онлайн табло для менеджера представлен на рисунке 4.3.10.

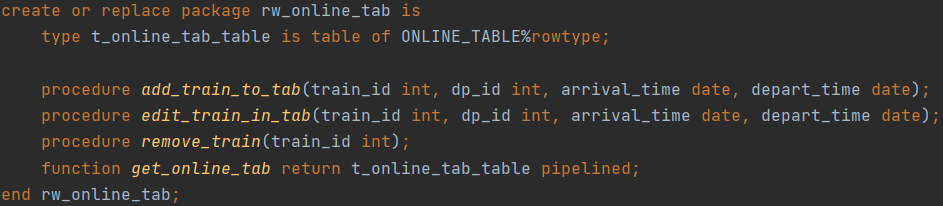


Рисунок 4.3.10 – Заголовок пакета RW\_NOTIFICATION\_CLIENT

Пакет RW\_ONLINE\_TAB содержит методы добавления, редактирования, удаления и чтения поездов в онлайн табло. Так как данный пакет предназначен только для менеджера, по аналогии с пакетом RW\_NOTIFICATION\_CLIENT был создан пакет RW\_ONLINE\_TAB\_CLIENT содержащий только методы для чтения данных.

## **Создание автоматизированных заданий**

Для поддержания актуальности информации в базе данный необходимо выполнять некоторые задания:

* очищать таблицу ключей авторизации TOKENS;
* архивировать неактуальные билеты в таблице TICKETS.

Первую задачу необходимо выполнять ежемесячно, а вторую в два часа ночи ежедневно. Для автоматизации задач Oracle Database 12c предлагает такое средство как Scheduler (Планировщик). Все необходимые методы расположены в пакете DBMS\_SCHEDULER. Используя данный пакет, автоматизируем необходимые задачи и создадим файл CreateSchduler.sql. PL/SQL блок представлен на рисунке 4.4.1.

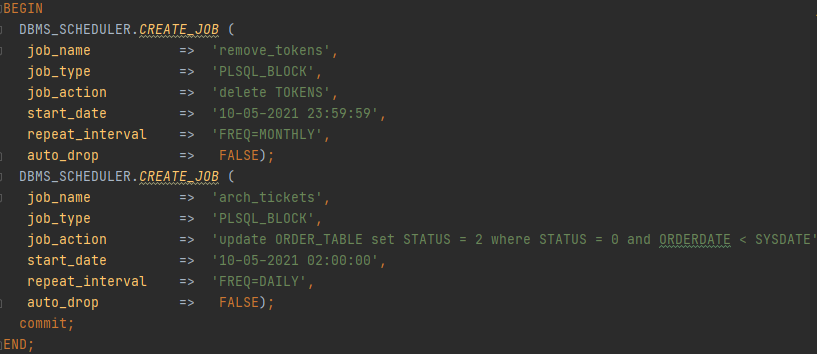


Рисунок 4.4.1 – Заголовок пакета RW\_NOTIFICATION\_CLIENT

## **Описание процедур экспорта и импорта**

Для комфортного администрирования базы данных в ней необходимо реализовать возможность экспорта и импорта данных в удобный для администратора формат. В нашем случае это формат xml.

Для доступа к папке из СУБД Oracle нужно должен быть создан Directory.

Объект Directory является логической ссылкой в базе данных на каталог файловой системы сервера, где установлена БД Oracle. Владельцем всех объектов Directory в базе данных является пользователь SYS, даже если объект Directory создан другим пользователем. Имена объектов Directory уникальны внутри всей БД. (рисунок 4.4.1).



Рисунок 4.4.1 – Создание объекта Directory

Для удобства администратора все процедуры импорта экспорта помещены в пакет RW\_XML.

На рисунке 4.4.2 представлена процедура импорта данных из БД в файл TICKET.XML из БД.

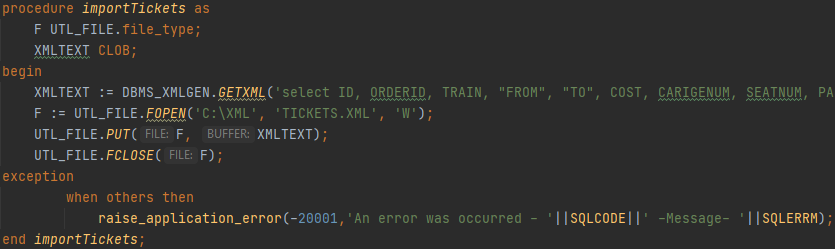


Рисунок 4.4.2 – Процедура импорта данных

На рисунке 4.4.3 представлена процедура экспорта данных из файла TICKET.XML в БД.

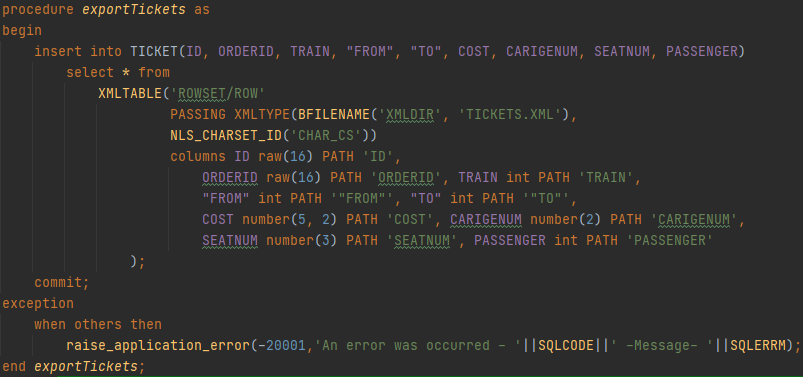


Рисунок 5.4.3 – Процедура экспорта данных

## **Тестирование производительности базы данных**

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. Для этого с помощью PL/SQL блока в таблицы было добавлено 100000 строк (рисунок 4.6.1).

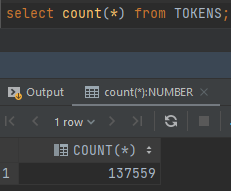


Рисунок 4.6.1 – Количество записей в таблице

Наиболее критическими являются методы получения данных и те, которых пользователь не видит, a именно is\_login таблицы TOKENS, get\_passengers таблицы PASSANGERS, get\_tickets таблицы TICKETS, get\_today\_notifications таблицы NOTIFICATIONS.

Построим планы по основным запросам каждого из методов.

Для метода is\_login основным запросом будет поиск определённой строки в таблице TOKENS с определённым значением ключа доступа. План данного запроса показан на рисунке 4.6.2.

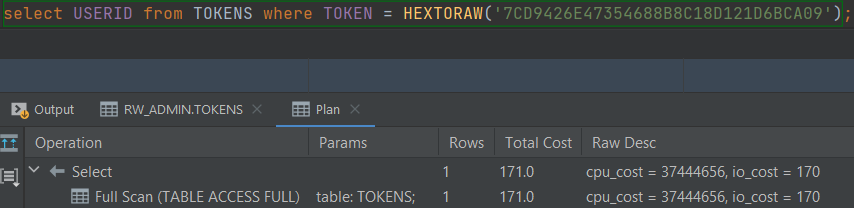


Рисунок 4.6.1 – План запроса для метода is\_login

Видно, что данный запрос проводит полное сканирование таблицы и большими затратами по вычислительным ресурсам. Данный метод вызывается горазда чаше, чем родственный ему метод login производящий вставку данных в таблицу TOKENS, поэтому будет целесообразно создать покрывающий индекс для колонок TOKEN, USERID таблицы TOKENS. Результат показан на рисунке 4.6.2.

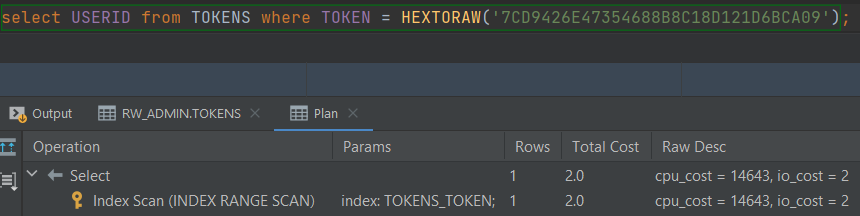


Рисунок 4.6.2 – План запроса для метода is\_login после создания индекса

Положительный результат очевиден, значительное уменьшение стоимости запроса, как по вычислительным, так и по дисковым ресурсам.

Для метода get\_passangers основным запросом будет поиск строк в таблице PASSANGERS с определённым значением ID клиента. План данного запроса показан на рисунке 4.6.3.

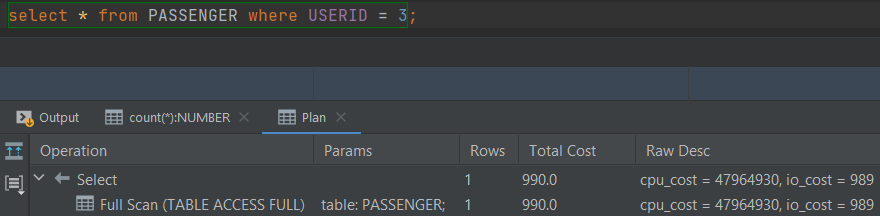


Рисунок 4.6.3 – План запроса для метода get\_ passangers

Видно, что данный запрос проводит полное сканирование таблицы и большими затратами по вычислительным ресурсам. Данный метод вызывается горазда чаше, чем родственный ему метод add\_passanger производящий вставку данных в таблицу PASSANGER, поэтому будет целесообразно создать индекс для колонки USERID таблицы PASSANGER. Результат показан на рисунке 4.6.4.

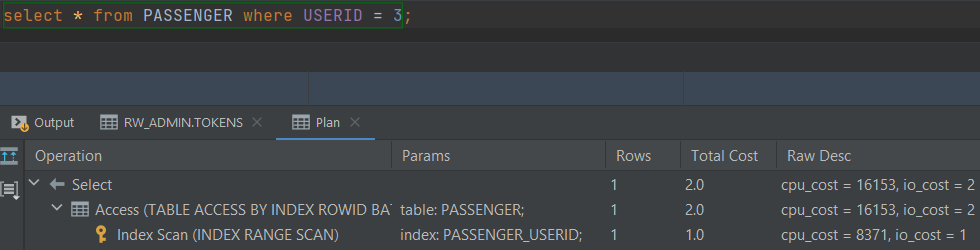
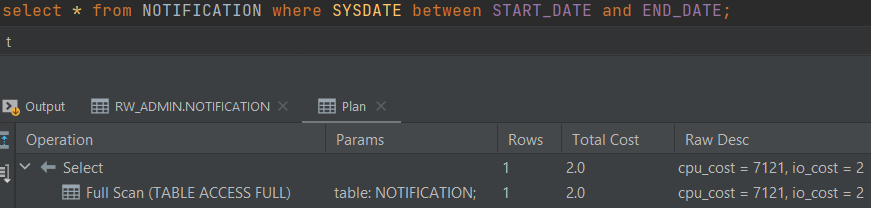


Рисунок 4.6.4 – План запроса для метода get\_passangers после создания индекса

Создание данного индекса позволило избежать полного сканирования таблицы, что значительно сократило использование вычислительных и дисковых ресурсов.

Метод get\_tickets логически полностью совпадает с методом get\_passangers поэтому создание индекса для этого метода тоже оправдано.

Для метода get\_today\_notifications основным запросом будет поиск строк в таблице NOTIFICATION с датами начала и окончания оповещения расположенными между текущей даты. План данного запроса показан на рисунке 4.6.5.



Поиск осуществляется полным сканирование таблицы, но с очень маленькой стоимостью запроса. Попытка улучшить результат индексом показана на рисунке 4.6.6.

Рисунок 4.6.5 – План запроса для метода get\_today\_notifications

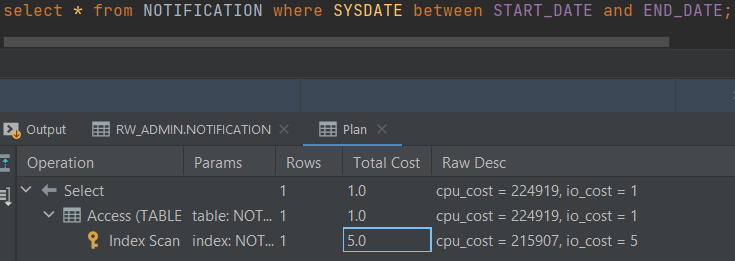


Рисунок 4.6.6 – План запроса для метода get\_today\_notifications после создания индекса

Как показал результат, индекс только ухудшил результат и его использование в данном случае не целесообразно.

# **Описание** **технологии репликации**

Репликация – это механизм синхронизации содержимого нескольких копий объекта (например, содержимого баз данных). Репликация — это процесс, под которым понимается копирование данных из одного источника на другой (или на множество других) и наоборот. При репликации изменения, сделанные в одной копии объекта, могут быть распространены в другие копии

Oracle Database обладает множеством механизмов репликации, один из них это Multi-Master Replication (MMR). Данная технология позволяет реплицировать:

* таблицы;
* индексы;
* процедуры, функции и триггеры;
* пакеты;
* пользовательские типы.

У использования MMR есть свои плюсы и минусы.

Положительные стороны MMR следующее:

* репликация большого количества типов объектов, включая объекты, определенные пользователем.
* обновления или изменения объектов реплицируемые. Добавление столбца в таблицу на главном сервере могут быть реплицированы на другие сервера.
* репликация с любым количеством других баз данных. Любой сервер может реплицировать с другими серверами.

Однако есть и недостатки, такие как:

* потенциально большие требования к пропускной способности сети. Множество серверов не только отправляют и извлекают изменения между ними, но и отправляют подтверждения и административные данные.
* понижение производительности. Сложность и надежность имеют свою цену. MMR предполагает использование триггеров и процедур, что может привести к снижение производительности базы данных. В зависимости от того, сколько у вас данных при репликации это снижение производительности может быть значительным.
* значительное увеличение требований к администрированию. Когда проблемы появляются в базе данных, администратор базы данных должен убедиться, что репликация не является причиной или что причина не реплицируется в другие базы данных. Настройка производительности базы данных и решение проблем становятся более сложными на порядок.
* изменения в базе данных требуют дополнительного планирования. Развертывание новой версии приложения может быть намного сложнее. Каждая новая версия будет требуют пересмотра дизайна репликации.

Исходя из вышеописанных аргументов, возникает главная рекомендация. Не реализовывать более сложную репликацию чем нужно.

Прежде чем приступать к репликации вы должны выполнить следующие:

* установите первичные ключи. Что бы реплицировать таблицу база данных должна определять уникальные строки. Это отлично решается первичными ключами;
* установите параметры базы данных. В таблице 5.1 дан синопсис всех необходимых изменений для поддержки MMR;
* проверьте, что пакеты репликации установлены.

Таблица 5.1 – Параметры базы данных

|  |  |  |
| --- | --- | --- |
| Название | Значение по умолчанию | Рекомендуемое значение |
| COMPATIBLE | Зависит от используемой вами версии Oracle | Установите в соответствии с наиболее старой версией Oracle участвующей в репликации |
| GLOBAL\_NAMES | false | True |
| OPEN\_LINKS | 4 | 4 + 2 за каждый сервер участвующий в репликации. |
| PARALLE\_AUTOMATIC\_ TURING | false | true |
| PROCESSES | Наследуется от параметра parallen\_max\_servers | Добавьте 12 к текущему значению |
| SHARED\_POO L\_SIZE | Зависит от ОС | Добавьте, как минимум 80МБ к текущему значению |

После настройки сервера СУБД, следующим шагом будет создание пользователя администрирования репликации. PL/SQL блок создающий данного показан на рисунке 5.1.

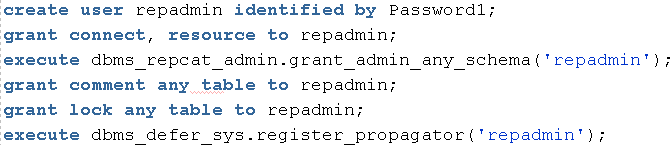


Рисунок 5.1 – Создание администратора репликации

Следует заметить, что пакеты dbms\_repcat\_admin и dbms\_defer\_sys находятся во владении пользователя SYS, поэтому, что создавать данного пользователя нужно только под ним.

После необходимо создать Database Links на каждую из баз данных на каждой из баз данных. Процесс показан на рисунке 5.2.

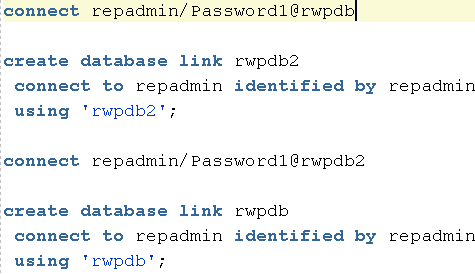


Рисунок 5.2 – Создание Database Links

Далее нам нужно автозадание, чтобы периодически подталкивать изменения на другие базы данных. Еще нам нужно автозадание по удалению изменений из очереди транзакций после того, как они были отправлены. Оба создаются методами пакета dbms\_ defer\_sys. Процесс показан на рисунке 5.3.

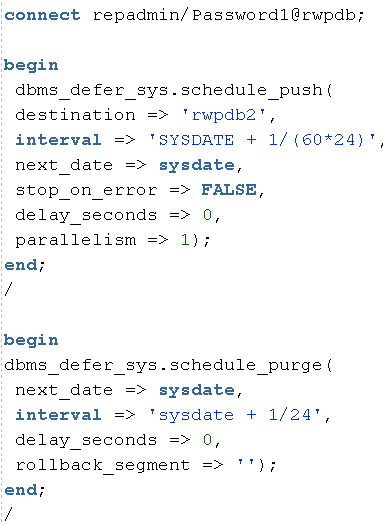


Рисунок 5.3 – Создание автозаданий

Далее идёт процесс создания группы репликации, добавление в неё объектов и создание поддержки репликации данных объектов. Процесс показан на рисунке 5.4.

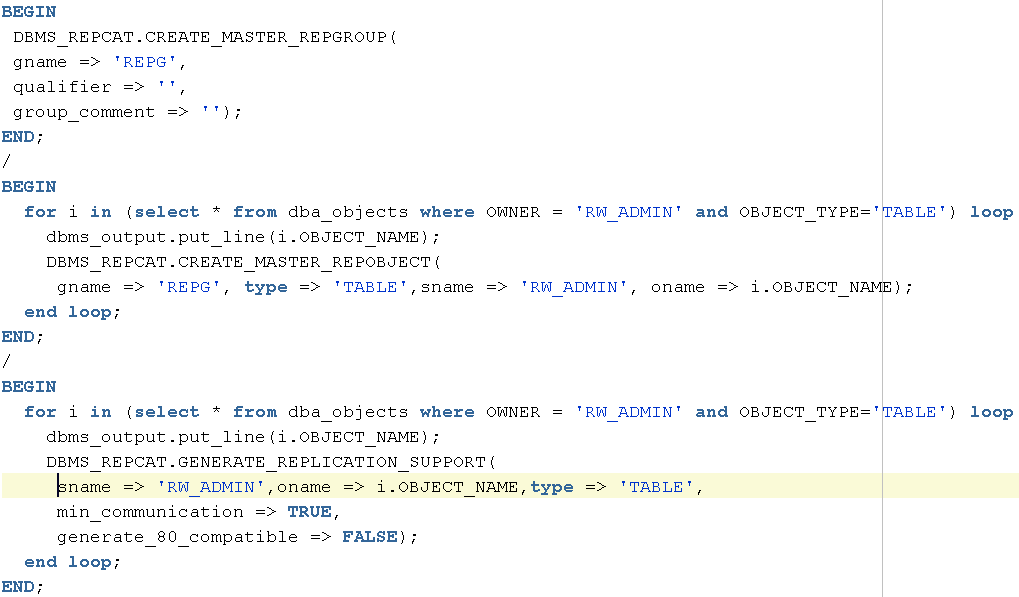


Рисунок 5.4 – Создание группы репликации

Группа репликации создаётся методом create\_master\_repgroup пакета dbms\_repcat, объекты добавляются методом create\_master\_repobject и их поддержка генерируется методом generate\_replication\_support. Данный метод создает объекты, триггеры, процедуры и т. д., необходимые Oracle для поддержки репликации.

Последний шаг — это добавление другой базы данных в группу репликации. Процесс показан на рисунке 5.5.

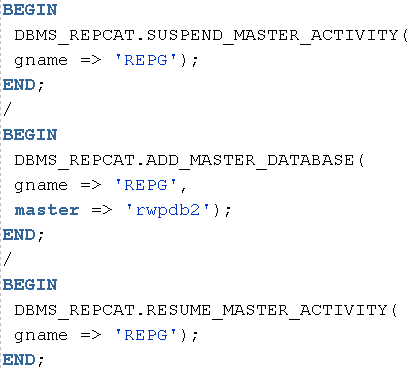


Рисунок 5.5 – Добавление базы данных в группу репликации

В показанном блоке происходит процесс остановки репликации методом suspend\_master\_activity. Эта команда остановит поддержку репликации не только на всех базах данных в схеме репликации. Он также помещает все объекты в группе репликации в режим только для чтения. Добавление базы данных в схему репликации осуществляется методом add\_master\_database. Возобновление процесса репликации осуществляется методом resume\_master\_activity.

# **Руководство по использованию программного средства**

В данной главе буду освещены основные методы использования программного в соответствии с диаграммами вариантов использования, описанными в главе 3.1.

* 1. Область клиента

Неавторизованный клиент может просмотреть расписание и использованием функций get\_routes\_for\_path и get\_path\_info пакета rw\_net\_client.

Регистрация осуществляется методом register пакета rw\_auth.

Зарегистрированный клиент может получить ключ авторизации методом login пакета rw\_auth. Полученный ключ неявно используется методом is\_login пакета rw\_auth, который встроен в другие методы.

Используя полученный ключ клиент может просмотреть свои билеты методом get\_tickets пакета rw\_order, отмечать избранные поезда и маршруты методами пакета rw\_user, так же данный пакет отвечает за редактирование профиля пользователя и добавления пассажиров.

Заказ билета осуществляются методами пакета rw\_order. В первую очередь создаётся заказ методом create\_order, затем методом add\_ticket добавляются билеты к данному заказу. Отмена и оплата заказа осуществляется методами cancel\_order и paid\_order.

* 1. Область менеджера

Менеджеру перед началом работы необходимо авторизоваться. За регистрацию менеджера отвечает администратор.

Создание поездов и маршрутов для них осуществляются методами пакета rw\_net. Для создания поезда методом create\_train необходимо создать маршрут методом reate\_route. Затем для добавления пунктов остановки в маршрут используется метод add\_connection\_to\_route.

Контроль уведомлений осуществляются методами пакета rw\_notification.

Так же для менеджера предоставлена возможность просмотреть информацию о конкретном билете методом пакета rw\_order\_manager.

# **Заключение**

Данный курсовой проект был направлен на развитие навыков администрирования базы данных, а также на улучшение понимания взаимодействия программного средства с БД Oracle 12C на удалённом сервере.

В ходе выполнения работы были достигнуты следующие результаты:

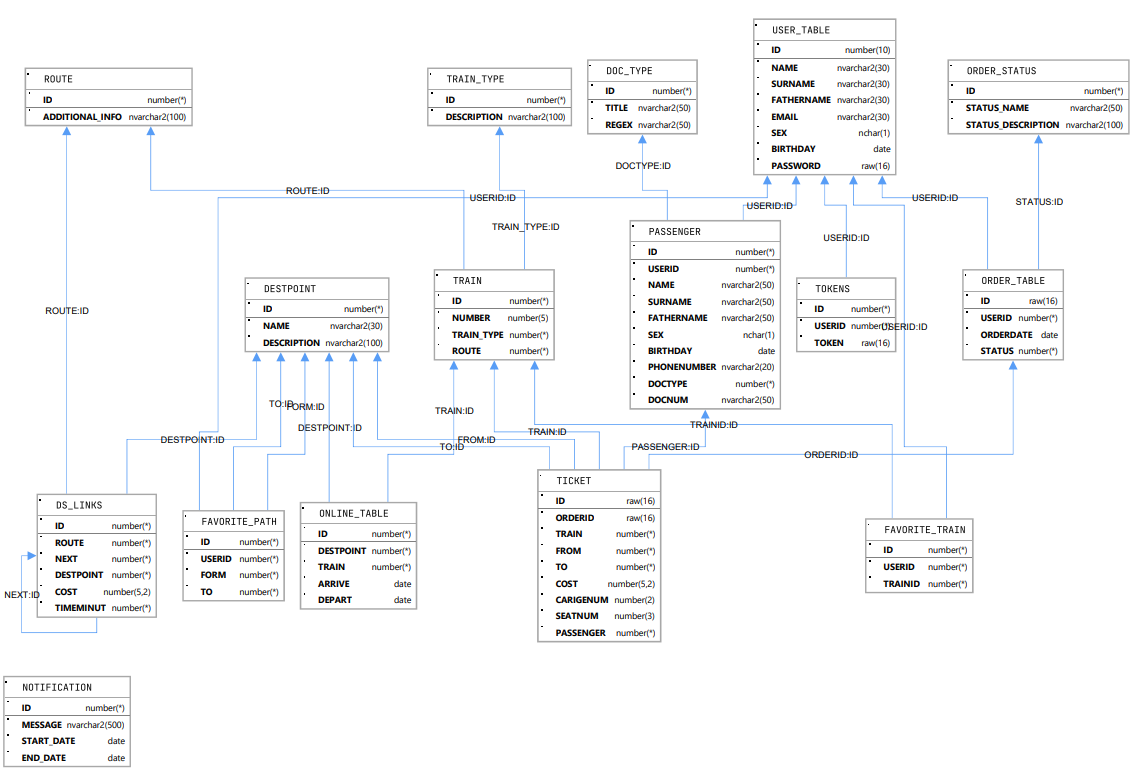
* исследованы обучающие приложения-аналоги;
* сформулирована модель данных и приведено её графическое представление;
* подготовлена база данных Oracle с использованием технологии резервного копирования и восстановления;
* подготовлены процедуры для осуществления импорта и экспорта данных с форматом xml;
* были созданы пользователи с различными привилегиями для работы с базой данных;
* были созданы скрипты для быстрого развертывания начальных таблиц компонентов и начальных таблиц пользователей базы данных с дальнейшим их заполнением.

# **Cписок источников**

1. Документация Oracle [Электронный ресурс] / Foundation, Inc. https://docs.oracle.com/cd/B28359\_01/server.111/b31222/toc.htm
2. Официальный сайт Oracle [Электронный ресурс] / Foundation, Inc. https://www.oracle.com/database/database-vault/index.html
3. Продукты Oracle [Электронный ресурс] [Электронный ресурс] / Foundation, Inc. http://www.interface.ru/home.asp?artId=24678
4. Документы Oracle [Электронный ресурс] / Foundation, Inc. http://www.oracle.com/technetwork/database/security/database-vault-ds-12c-1898877.pdf

## **Приложение А**

**Логическая схема базы данных**

****

## **Приложение Б**

**Создание таблиц**

create table USER\_TABLE  
(  
 ID int generated always as identity,  
 NAME nvarchar2(30) not null,  
 SURNAME nvarchar2(30) not null,  
 FATHERNAME nvarchar2(30),  
 EMAIL nvarchar2(30) not null,  
 SEX nchar  
 constraint SEX\_CHECK  
 check ( SEX in ('m', 'f')),  
 BIRTHDAY date,  
 PASSWORD raw(16) not null,  
 constraint USER\_PK primary key(ID)  
);  
/  
create table TOKENS  
(  
 ID int generated by default as identity  
 constraint TOKENS\_PK  
 primary key,  
 USERID int not null  
 constraint TOKENS\_USER\_TABLE\_FK  
 references USER\_TABLE (ID)  
 on delete cascade,  
 TOKEN RAW(16) not null  
)  
/  
create table DOC\_TYPE  
(  
 ID int generated as identity  
 constraint DOC\_TYPE\_PK  
 primary key,  
 TITLE nvarchar2(50) not null,  
 REGEX nvarchar2(50) not null  
)  
/  
create table PASSENGER  
(  
 ID int generated always as identity  
 constraint PASSENGER\_PK  
 primary key ,  
 USERID int not null  
 constraint PASSENGER\_USER\_TABLE\_FK  
 references USER\_TABLE(ID)  
 on delete cascade,  
 NAME nvarchar2(50) not null,  
 SURNAME nvarchar2(50) not null,  
 FATHERNAME nvarchar2(50),  
 SEX nchar  
 constraint PASSENGER\_SEX\_CHECK  
 check ( SEX in ('m', 'f')),  
 BIRTHDAY date not null ,  
 PHONENUMBER nvarchar2(20),  
 DOCTYPE int not null  
 constraint PASSENGER\_DOC\_TYPE\_FK  
 references DOC\_TYPE(ID),  
 DOCNUM nvarchar2(50) not null  
)  
/  
create table ORDER\_STATUS(  
 ID int generated always as identity  
 constraint ORDER\_STATUS\_PK  
 primary key,  
 STATUS\_NAME nvarchar2(50) not null,  
 STATUS\_DESCRIPTION nvarchar2(100) not null  
)  
/  
create table ORDER\_TABLE(  
 ID raw(16)  
 constraint ORDER\_TABLE\_PK  
 primary key,  
 USERID int not null  
 constraint ORDER\_TABLE\_USER\_TABLE\_FK  
 references USER\_TABLE(ID),  
 ORDERDATE date not null,  
 STATUS int not null  
 constraint ORDER\_TABLE\_ORDER\_STATUS\_FK  
 references ORDER\_STATUS(ID)  
)  
/  
  
create table DESTPOINT(  
 ID int generated as identity  
 constraint DESTPOINT\_PK  
 primary key,  
 NAME nvarchar2(30) not null,  
 DESCRIPTION nvarchar2(100) not null  
)  
/  
  
create table TRAIN\_TYPE(  
 ID int generated always as identity  
 constraint TRAIN\_TYPE\_PK  
 primary key,  
 DESCRIPTION nvarchar2(100) not null  
)  
/  
create table ROUTE(  
 ID int generated by default as identity  
 constraint ROUTE\_PK  
 primary key,  
 ADDITIONAL\_INFO nvarchar2(100)  
)  
/  
create table TRAIN(  
 ID int generated as identity  
 constraint TRAIN\_PK  
 primary key,  
 "NUMBER" decimal(5, 0),  
 TRAIN\_TYPE int not null  
 constraint TRAIN\_TRAIN\_TYPE  
 references TRAIN\_TYPE(ID),  
 ROUTE int not null  
 constraint TRAIN\_ROUTE\_FK  
 references ROUTE(ID)  
)  
/  
create table DS\_LINKS(  
 ID int generated as identity  
 constraint DS\_LINKS\_PK  
 primary key,  
 ROUTE int not null  
 constraint DS\_LINKS\_ROUTE  
 references ROUTE(ID)  
 on delete set null,  
 NEXT int  
 constraint DS\_LINKS\_INNER\_FK  
 references DS\_LINKS(ID)  
 on delete set null,  
 DESTPOINT int  
 constraint DS\_LINKS\_DESTPOINT\_FK  
 references DESTPOINT(ID),  
 COST decimal(5, 2) not null  
)  
/  
alter table DS\_LINKS add TIMEMINUT int;  
/  
create table TICKET(  
 ID raw(16) not null  
 constraint TICKET\_PK  
 primary key,  
 ORDERID raw(16) not null  
 constraint TICKET\_ORDER\_TABLE\_FK  
 references ORDER\_TABLE(ID)  
 on delete set null,  
 TRAIN int not null  
 constraint TICKET\_TRAIN\_FK  
 references TRAIN(ID)  
 on delete set null,  
 "FROM" int not null  
 constraint TICKET\_DESTPOINT\_FROM\_FK  
 references DESTPOINT(ID),  
 "TO" int not null  
 constraint TICKET\_DESTPOINT\_TO\_FK  
 references DESTPOINT(ID),  
 COST decimal(5, 2) not null,  
 CARIGENUM decimal(2, 0),  
 SEATNUM decimal(3, 0)  
)  
/  
alter table TICKET add PASSENGER int  
 constraint TICKET\_PASSENGER\_FK  
 references PASSENGER(ID);  
/  
create table FAVORITE\_TRAIN(  
 ID int generated as identity  
 constraint FAVORITE\_TRAIN\_PK  
 primary key,  
 USERID int  
 constraint FAVORITE\_TRAIN\_USER\_TABLE\_FK  
 references USER\_TABLE(ID)  
 on delete cascade,  
 TRAINID int  
 constraint FAVORITE\_TRAIN\_TRAIN\_FK  
 references TRAIN(ID)  
)  
/  
create table FAVORITE\_PATH(  
 ID int generated as identity  
 constraint FAVORITE\_PATH\_PK  
 primary key,  
 USERID int  
 constraint FAVORITE\_PATH\_USER\_TABLE\_FK  
 references USER\_TABLE(ID)  
 on delete cascade,  
 "FORM" int not null  
 constraint FAVORITE\_PATH\_DESTPOINT\_TO  
 references DESTPOINT(ID),  
 "TO" int not null  
 constraint FAVORITE\_PATH\_DESTPOINT\_FROM  
 references DESTPOINT(ID)  
)  
/  
create table NOTIFICATION(  
 ID int generated as identity  
 constraint NOTIFICATIONS\_PK  
 primary key,  
 MESSAGE nvarchar2(500) not null,  
 START\_DATE DATE not null,  
 END\_DATE DATE not null  
)

create table ONLINE\_TABLE(  
 ID int generated as identity  
 constraint ONLINE\_TABLE\_PK  
 primary key,  
 DESTPOINT int  
 constraint DESTPOINT\_ONLINE\_TABLE\_FK  
 references DESTPOINT(ID),  
 TRAIN int  
 constraint ONLINE\_TABLE\_TRAIN\_FK  
 references TRAIN(ID),  
 ARRIVE date,  
 DEPART date  
)  
/

/  
commit;  
/

create index TOKENS\_TOKEN  
 on TOKENS(TOKEN, USERID);  
/  
create index PASSENGER\_USERID  
 on PASSENGER(USERID);  
/  
create index TICKET\_USERID  
 on TICKET(ORDERID);

Приложение В

**Тело пакета RW\_AUTH**

create or replace package body rw\_auth is  
  
 procedure *register*(  
 user\_name nvarchar2,  
 user\_surname nvarchar2,  
 user\_fathername nvarchar2,  
 user\_email nvarchar2,  
 user\_sex nchar,  
 user\_birthday date,  
 user\_password raw  
 ) is begin  
 insert into USER\_TABLE(NAME, SURNAME, FATHERNAME, EMAIL, SEX, BIRTHDAY, PASSWORD)  
 values (user\_name,  
 user\_surname,  
 user\_fathername,  
 user\_email,  
 user\_sex,  
 user\_birthday,  
 user\_password);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *register*;  
  
 function *login*(  
 emailL nvarchar2,  
 passwordL raw  
 ) return raw is  
 key raw(16);  
 uid int := 0;  
 begin  
 select ID into uid from USER\_TABLE u  
 where u.EMAIL = emailL and u.PASSWORD = passwordL;  
 if (uid < 0) then  
 return null;  
 end if;  
 key := SYS\_GUID();  
 insert into TOKENS(USERID, TOKEN)  
 values (uid, key);  
 commit;  
 return key;  
 exception  
 when NO\_DATA\_FOUND then  
 return null;  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *login*;  
  
  
 function *is\_login*(  
 key raw  
 ) return number is  
 tikenid NUMBER := null;  
 begin  
 select USERID into tikenid from TOKENS where TOKEN = key;  
 return tikenid;  
 exception  
 when NO\_DATA\_FOUND then  
 return null;  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *is\_login*;  
end rw\_auth;

Приложение Г

**Тело пакета RW\_USER**

create or replace package body rw\_user as  
 procedure *update\_info*(  
 key raw,  
 username nvarchar2,  
 surname nvarchar2,  
 father\_name nvarchar2,  
 email nvarchar2,  
 sex nchar,  
 birthday date  
 )as  
 userid number;  
 user\_row USER\_TABLE%rowtype;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 select \* into user\_row from USER\_TABLE where id = userid;  
 if(username is not null) then  
 user\_row.name := username;  
 end if;  
 if(surname is not null) then  
 user\_row.surname := surname;  
 end if;  
 if(father\_name is not null) then  
 user\_row.fathername:= father\_name;  
 end if;  
 if(email is not null) then  
 user\_row.email := email;  
 end if;  
 if(sex is not null) then  
 user\_row.sex := sex;  
 end if;  
 if(birthday is not null) then  
 user\_row.birthday := birthday;  
 end if;  
 update USER\_TABLE set ROW = user\_row where ID = userid;  
 commit;  
 end *update\_info*;  
  
 function *get\_passengers*(  
 key raw  
 ) return t\_passenger\_table pipelined is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 for i in (select \* from PASSENGER  
 where USERID = user\_id) loop  
 pipe row ( i );  
 end loop;  
 end *get\_passengers*;  
  
 procedure *add\_passenger*(  
 key raw,  
 passenger\_name nvarchar2,  
 passenger\_surname nvarchar2,  
 father\_name nvarchar2,  
 passenger\_sex nchar,  
 passenger\_birthday date,  
 phone\_number nvarchar2,  
 docid number,  
 doc nvarchar2  
 ) as  
 userid number;  
 doc\_regex nvarchar2(50);  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 select regex into doc\_regex from DOC\_TYPE where ID = docid;  
 if not regexp\_like(doc, doc\_regex) then  
 return;  
 end if;  
 insert into PASSENGER(NAME, SURNAME, FATHERNAME, SEX, BIRTHDAY, PHONENUMBER, DOCTYPE, DOCNUM)  
 values (passenger\_name,  
 passenger\_surname,  
 father\_name,  
 passenger\_sex,  
 passenger\_birthday,  
 phone\_number,  
 docid,  
 doc);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_passenger*;  
  
 procedure *remove\_passenger*(  
 key raw,  
 passenger\_id number  
 )as  
 userid number;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 delete PASSENGER where ID = passenger\_id;  
 end *remove\_passenger*;  
  
 procedure *add\_favorite\_path*(key raw, fromD number, toD number) as  
 userid number;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 insert into FAVORITE\_PATH(USERID, "FORM", "TO") values (userid, fromD, toD);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_favorite\_path*;  
 procedure *remove\_favorite\_path*(key raw, idFP number) as  
 userid number;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 delete FAVORITE\_PATH where ID = idFP;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_favorite\_path*;  
 function *get\_favorite\_paths*(key raw) return t\_favorite\_path\_table pipelined is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 for i in (select \* from FAVORITE\_PATH  
 where USERID = user\_id) loop  
 pipe row ( i );  
 end loop;  
 end *get\_favorite\_paths*;  
  
 procedure *add\_favorite\_train*(key raw, idT number) as  
 userid number;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 insert into FAVORITE\_TRAIN(USERID, TRAINID) values (userid, idT);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_favorite\_train*;  
 procedure *remove\_favorite\_train*(key raw, idFT number)as  
 userid number;  
 begin  
 userid := RW\_AUTH.*IS\_LOGIN*(key);  
 if userid is null then  
 return;  
 end if;  
 delete FAVORITE\_TRAIN where ID = idFT;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_favorite\_train*;  
 function *get\_favorite\_trains*(key raw) return t\_favorite\_train\_table pipelined is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 for i in (select \* from FAVORITE\_TRAIN  
 where USERID = user\_id) loop  
 pipe row ( i );  
 end loop;  
 end *get\_favorite\_trains*;  
end rw\_user;

Приложение Д

**Тело пакета RW\_NET**

create or replace package body rw\_net is  
 function *get\_destpoints* return t\_destpoint\_table pipelined  
 is begin  
 for i in (select \* from DESTPOINT) loop  
 PIPE ROW ( i );  
 end loop;  
 end *get\_destpoints*;  
  
 procedure *create\_destpoint*(  
 namedp nvarchar2,  
 descriptiondp nvarchar2,  
 dpid out number  
 ) is  
 begin  
 insert into DESTPOINT(NAME, DESCRIPTION)  
 values (namedp, descriptiondp)  
 returning ID into dpid;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *create\_destpoint*;  
  
 procedure *update\_destpoint*(  
 dpid number,  
 namedp nvarchar2,  
 descriptiondp nvarchar2  
 ) is  
 begin  
 update DESTPOINT set NAME = namedp, DESCRIPTION = descriptiondp  
 where ID = dpid;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_destpoint*;  
  
 procedure *remove\_destpoint*(  
 dpid number  
 ) is  
 begin  
 for i in (select d1.ID id1,  
 d2.ID id2,  
 d2.NEXT next2,  
 d2.COST cost2,  
 d2.COST time2  
 from DS\_LINKS d1  
 inner join DS\_LINKS d2 on d1.NEXT = d2.ID  
 where d2.DESTPOINT = dpid  
 for update) loop  
 update DS\_LINKS set  
 NEXT = i.next2,  
 COST = COST + i.cost2,  
 TIMEMINUT = TIMEMINUT + i.time2  
 where ID = i.id1;  
 delete DS\_LINKS where ID = i.id2;  
 end loop;  
 delete DESTPOINT where ID = dpid;  
 commit;  
 exception  
 when others then  
 rollback;  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_destpoint*;  
  
 function *get\_routes* return t\_route\_table pipelined is  
 begin  
 for i in (select \* from ROUTE) loop  
 PIPE ROW ( i );  
 end loop;  
 end *get\_routes*;  
  
 procedure *create\_route*(idr number, info nvarchar2) is  
 begin  
 insert into ROUTE values(idr, info);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *create\_route*;  
  
 procedure *remove\_route*(idr number) is  
 begin  
 delete ROUTE where ID = idr;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_route*;  
  
 procedure *add\_connection\_to\_route*(  
 routec number,  
 fromds\_lID number,  
 ds number,  
 costc number,  
 tm number,  
 dplid out number  
 ) is  
 begin  
 insert into DS\_LINKS(ROUTE, DESTPOINT, COST, TIMEMINUT, NEXT)  
 values (routec, ds, costc, tm, null) returning ID into dplid;  
 update DS\_LINKS set NEXT = dplid where ID = fromds\_lID;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_connection\_to\_route*;  
  
 function *routes\_for\_path*(src number, dest number)  
 return t\_number pipelined is  
 l\_return sys\_refcursor;  
 num number;  
 begin  
 open l\_return for  
 with d(destination) as (  
 select dest from dual  
 )  
 select ROUTE from d  
 left join (select ROUTE, DESTPOINT from DS\_LINKS  
 START WITH DESTPOINT = src  
 CONNECT BY PRIOR NEXT = ID) a  
 on d.destination = a.DESTPOINT;  
 LOOP  
 FETCH l\_return INTO num;  
 EXIT WHEN l\_return%NOTFOUND;  
 PIPE ROW ( num );  
 END LOOP;  
 end *routes\_for\_path*;  
  
 function *get\_path\_data*(routeid number, src number, dest number)  
 return t\_path\_info pipelined  
 is  
 l\_return sys\_refcursor;  
 pathr r\_path\_info;  
 destn number;  
 costn number;  
 pathtimen number;  
 begin  
 open l\_return for  
 select d2.DESTPOINT dest,  
 sum(d1.COST) over (ORDER BY d1.ROUTE, d1.DESTPOINT) cost,  
 sum(d1.TIMEMINUT) over (ORDER BY d1.ROUTE, d1.DESTPOINT, d2.DESTPOINT) pathtime  
  
 from DS\_LINKS d1  
 left join DS\_LINKS d2 on d1.NEXT = d2.ID  
 where d1.ROUTE = routeid  
 start with d1.DESTPOINT = src  
 CONNECT BY PRIOR d1.NEXT = d1.ID and d1.DESTPOINT != dest;  
 loop  
 FETCH l\_return INTO destn, costn, pathtimen;  
 EXIT WHEN l\_return%NOTFOUND;  
 pathr.dest := destn;  
 pathr.cost := costn;  
 pathr.pathtime := pathtimen;  
 PIPE ROW ( pathr );  
 end loop;  
 end *get\_path\_data*;  
  
 procedure *create\_train*(  
 tnum number,  
 ttype number,  
 troute number,  
 tid out number  
 ) is  
 begin  
 insert into TRAIN("NUMBER", TRAIN\_TYPE, ROUTE)  
 values (tnum, ttype, troute)  
 returning ID into tid;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *create\_train*;  
  
 procedure *remove\_train*(  
 tid number  
 )is  
 begin  
 delete TRAIN where ID = tid;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_train*;  
  
 procedure *update\_train*(  
 tid number,  
 tnum number,  
 ttype number,  
 troute number  
 )is  
 trow TRAIN%rowtype;  
 begin  
 select \* into trow from TRAIN where ID = tid;  
 if tnum is not null then  
 trow."NUMBER" := tnum;  
 end if;  
 if ttype is not null then  
 trow.TRAIN\_TYPE := ttype;  
 end if;  
 if troute is not null then  
 trow.ROUTE := troute;  
 end if;  
 update TRAIN set ROW = trow  
 where ID = tid;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_train*;  
  
 function *get\_train\_info*(tid number)  
 return TRAIN%rowtype  
 is  
 trow TRAIN%rowtype;  
 begin  
 select \* into trow from TRAIN where ID = tid;  
 return trow;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *get\_train\_info*;  
end rw\_net;

Приложение Е

**Тело пакета RW\_ORDER**

create or replace package body rw\_order as  
 function *get\_order*(key raw)  
 return ORDER\_TABLE%rowtype  
 is  
 user\_id number;  
 orow ORDER\_TABLE%rowtype;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 select \* into orow from ORDER\_TABLE  
 where USERID = user\_id and STATUS = 1;  
 return orow;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *get\_order*;  
  
 procedure *create\_order*(  
 key raw,  
 order\_id out raw  
 ) is  
 user\_id number;  
 guid raw(16);  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 guid := SYS\_GUID();  
 insert into ORDER\_TABLE(USERID, ORDERDATE, STATUS)  
 values (user\_id, SYSDATE, 1)  
 returning ID into order\_id;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *create\_order*;  
  
 procedure *cancel\_order*(  
 key raw, order\_id raw  
 ) is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 delete ORDER\_TABLE where ID = order\_id;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *cancel\_order*;  
  
 procedure *paid\_order*(  
 key raw, order\_id raw  
 )is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 update ORDER\_TABLE set STATUS = 2 where ID = order\_id;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *paid\_order*;  
  
 procedure *add\_ticket*(  
 key raw,  
 order\_id raw,  
 train\_id number,  
 src number,  
 dest number,  
 pass number,  
 carrige\_num number,  
 seat\_num number  
 )is  
 user\_id number;  
 trow TRAIN%rowtype;  
 pathcur sys\_refcursor;  
 pc number;  
 pt number;  
 pd number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 trow := RW\_NET.*GET\_TRAIN\_INFO*(train\_id);  
 pathcur := RW\_NET.*GET\_PATH\_DATA*(trow.ROUTE, src, dest);  
 --open pathcur;  
 loop  
 exit when pathcur%notfound;  
 fetch pathcur into pd, pc, pt;  
 end loop;  
 insert into TICKET (ORDERID, TRAIN, "FROM", "TO", PASSENGER, COST, CARIGENUM, SEATNUM)  
 values(order\_id, train\_id, src, dest, pass, pc, carrige\_num, seat\_num);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_ticket*;  
  
 procedure *remove\_ticket*(  
 key raw,  
 ticket\_id raw  
 ) is  
 user\_id number;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 delete TICKET where ID = ticket\_id;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_ticket*;  
  
 function *get\_tickets*(  
 key raw  
 ) return t\_ticket\_table pipelined is  
 user\_id number;  
 trow TICKET%rowtype;  
 begin  
 user\_id := RW\_AUTH.*IS\_LOGIN*(key);  
 if user\_id is null then  
 return;  
 end if;  
 for i in (  
 select t.\* from TICKET t  
 inner join ORDER\_TABLE o on t.ORDERID = o.ID  
 where o.USERID = user\_id)  
 loop  
 PIPE ROW ( i );  
 end loop;  
 end;  
end rw\_order;

Приложение Ж

**Тело пакета RW\_NOTIFICATION**

create or replace package body rw\_notification as  
 procedure *add\_notification*(text nvarchar2, start\_d date, end\_d date) as  
 begin  
 insert into NOTIFICATION(MESSAGE, START\_DATE, END\_DATE) values (text, start\_d, end\_d);  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_notification*;  
 procedure *update\_notification*(idN number, text nvarchar2, start\_d date, end\_d date)as  
 begin  
 update NOTIFICATION set MESSAGE = text, START\_DATE = start\_d, END\_DATE = end\_d where ID = idN;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_notification*;  
 procedure *remove\_notification*(idN number) as  
 begin  
 delete NOTIFICATION where ID = idN;  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_notification*;  
 function *get\_all\_notifications* return t\_notification\_table pipelined is  
 begin  
 for i in (select \* from NOTIFICATION) loop  
 pipe row ( i );  
 end loop;  
 end *get\_all\_notifications*;  
 function *get\_today\_notifications* return t\_notification\_table pipelined is  
 begin  
 for i in (select \* from NOTIFICATION where SYSDATE between START\_DATE and END\_DATE) loop  
 pipe row ( i );  
 end loop;  
 end *get\_today\_notifications*;  
end rw\_notification;  
/  
create or replace package rw\_notification\_client as  
 type t\_notification\_table is table of NOTIFICATION%ROWTYPE;  
 function *get\_today\_notifications* return t\_notification\_table pipelined;  
end rw\_notification\_client;  
/  
create or replace package body rw\_notification\_client as  
 function *get\_today\_notifications* return t\_notification\_table pipelined is  
 begin  
 for i in (select \* from NOTIFICATION where SYSDATE between START\_DATE and END\_DATE) loop  
 pipe row ( i );  
 end loop;  
 end *get\_today\_notifications*;  
end rw\_notification\_client;

Приложение З

**Тело пакета RW\_TYPES**

create or replace package body rw\_types as  
 procedure *add\_train\_type*(  
 train\_description nvarchar2,  
 train\_id out number)  
 is  
 begin  
 insert into TRAIN\_TYPE(DESCRIPTION)  
 values(train\_description)  
 returning ID into train\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_train\_type*;  
  
 procedure *remove\_train\_type*(  
 train\_id number  
 )is  
 begin  
 delete TRAIN\_TYPE where ID = train\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_train\_type*;  
  
 procedure *update\_train\_type*(  
 train\_id number,  
 train\_description nvarchar2  
 )is  
 begin  
 update TRAIN\_TYPE set DESCRIPTION = train\_description  
 where ID = train\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_train\_type*;  
  
 procedure *add\_order\_status*(  
 status nvarchar2,  
 description nvarchar2,  
 status\_id out number  
 )is  
 begin  
 insert into ORDER\_STATUS(STATUS\_DESCRIPTION, STATUS\_NAME)  
 values(status, description)  
 returning ID into status\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_order\_status*;  
  
 procedure *remove\_order\_status*(status\_id number)is  
 begin  
 delete ORDER\_STATUS where ID = status\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_order\_status*;  
  
 procedure *update\_order\_status*(status\_id number,status nvarchar2, description nvarchar2)is  
 begin  
 update ORDER\_STATUS set STATUS\_NAME = status, STATUS\_DESCRIPTION = description  
 where ID = status\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_order\_status*;  
  
 procedure *add\_doc\_type*(  
 doc\_title nvarchar2,  
 doc\_rgx nvarchar2,  
 doc\_id out number  
 )is  
 begin  
 insert into DOC\_TYPE(TITLE, REGEX)  
 values(doc\_title, doc\_rgx)  
 returning ID into doc\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *add\_doc\_type*;  
  
 procedure *remove\_doc\_type*(doc\_id number)is  
 begin  
 delete DOC\_TYPE where ID = doc\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *remove\_doc\_type*;  
 procedure *update\_doc\_type*(doc\_id number, doc\_title nvarchar2, doc\_rgx nvarchar2)is  
 begin  
 update DOC\_TYPE set TITLE = doc\_title, REGEX = doc\_rgx  
 where ID = doc\_id;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *update\_doc\_type*;  
end rw\_types;

Приложение И

**Тело пакета RW\_XML**

create or replace package body rw\_xml as  
  
 procedure *importTickets* as  
 F UTL\_FILE.file\_type;  
 XMLTEXT CLOB;  
 begin  
 XMLTEXT := DBMS\_XMLGEN.*GETXML*('select ID, ORDERID, TRAIN, "FROM", "TO", COST, CARIGENUM, SEATNUM, PASSENGER from TICKET');  
 F := UTL\_FILE.*FOPEN*('C:\XML', 'TICKETS.XML', 'W');  
 UTL\_FILE.*PUT*(F, XMLTEXT);  
 UTL\_FILE.*FCLOSE*(F);  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *importTickets*;  
  
 procedure *exportTickets* as  
 begin  
 insert into TICKET(ID, ORDERID, TRAIN, "FROM", "TO", COST, CARIGENUM, SEATNUM, PASSENGER)  
 select \* from  
 XMLTABLE('ROWSET/ROW'  
 PASSING XMLTYPE(BFILENAME('XMLDIR', 'TICKETS.XML'),  
 NLS\_CHARSET\_ID('CHAR\_CS'))  
 columns ID raw(16) PATH 'ID',  
 ORDERID raw(16) PATH 'ORDERID', TRAIN int PATH 'TRAIN',  
 "FROM" int PATH '"FROM"', "TO" int PATH '"TO"',  
 COST number(5, 2) PATH 'COST', CARIGENUM number(2) PATH 'CARIGENUM',  
 SEATNUM number(3) PATH 'SEATNUM', PASSENGER int PATH 'PASSENGER'  
 );  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *exportTickets*;  
end rw\_xml;

Приложение И

**Тело пакета RW\_XML**

create or replace package body rw\_xml as  
  
 procedure *importTickets* as  
 F UTL\_FILE.file\_type;  
 XMLTEXT CLOB;  
 begin  
 XMLTEXT := DBMS\_XMLGEN.*GETXML*('select ID, ORDERID, TRAIN, "FROM", "TO", COST, CARIGENUM, SEATNUM, PASSENGER from TICKET');  
 F := UTL\_FILE.*FOPEN*('C:\XML', 'TICKETS.XML', 'W');  
 UTL\_FILE.*PUT*(F, XMLTEXT);  
 UTL\_FILE.*FCLOSE*(F);  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *importTickets*;  
  
 procedure *exportTickets* as  
 begin  
 insert into TICKET(ID, ORDERID, TRAIN, "FROM", "TO", COST, CARIGENUM, SEATNUM, PASSENGER)  
 select \* from  
 XMLTABLE('ROWSET/ROW'  
 PASSING XMLTYPE(BFILENAME('XMLDIR', 'TICKETS.XML'),  
 NLS\_CHARSET\_ID('CHAR\_CS'))  
 columns ID raw(16) PATH 'ID',  
 ORDERID raw(16) PATH 'ORDERID', TRAIN int PATH 'TRAIN',  
 "FROM" int PATH '"FROM"', "TO" int PATH '"TO"',  
 COST number(5, 2) PATH 'COST', CARIGENUM number(2) PATH 'CARIGENUM',  
 SEATNUM number(3) PATH 'SEATNUM', PASSENGER int PATH 'PASSENGER'  
 );  
 commit;  
 exception  
 when others then  
 raise\_application\_error(-20001,'An error was occurred - '||SQLCODE||' -Message- '||SQLERRM);  
 end *exportTickets*;  
end rw\_xml;

Приложение К

**Тело пакета ONLINE\_TABLE**

create or replace package body rw\_online\_tab is  
  
 procedure *add\_train\_to\_tab*(train\_id int, dp\_id int, arrival\_time date, depart\_time date) is  
 count\_t int;  
 begin  
 select count(\*) into count\_t from ONLINE\_TABLE;  
 if(count\_t != 0) then  
 raise\_application\_error(-20001, 'the train has already been added to the online\_table');  
 end if;  
 insert into ONLINE\_TABLE(TRAIN, DESTPOINT, ARRIVE, DEPART) values(train\_id, dp\_id, arrival\_time, depart\_time);  
 end *add\_train\_to\_tab*;  
  
 procedure *edit\_train\_in\_tab*(train\_id int, dp\_id int, arrival\_time date, depart\_time date)is  
 begin  
 update ONLINE\_TABLE set DESTPOINT = dp\_id, ARRIVE = arrival\_time, DEPART = depart\_time where TRAIN = train\_id;  
 end *edit\_train\_in\_tab*;  
  
 procedure *remove\_train*(train\_id int)is  
 begin  
 delete ONLINE\_TABLE where TRAIN = train\_id;  
 end *remove\_train*;  
  
 function *get\_online\_tab* return t\_online\_tab\_table pipelined is  
 begin  
 for i in (  
 select \* from ONLINE\_TABLE)  
 loop  
 PIPE ROW ( i );  
 end loop;  
 end *get\_online\_tab*;  
  
end rw\_online\_tab;