Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)
Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Методические указания к выполнению лабораторной работы №2 по курсу «Базы данных» на тему:

Создание и заполнение БД в среде Oracle

Москва, 2018

Теоретическая часть

В основе комплекса программ лежит Oracle Database Express Edition (Oracle Database XE) - это бесплатная, урезанная по объему хранимой информации версия СУБД Oracle Database. Для администрирования и создания базы данных в Oracle Database XE предлагается использовать SQL Developer. Для разработки схемы базы данных предлагается использовать Oracle SQL Data Modeler

Oracle SQL Developer – бесплатный инструмент для написания SQL-запросов, разработки PL/SQL пакетов, процедур, функций, триггеров и т. п. Этот инструмент написан на языке Java и является кросс-платформенным, т. е. работает во всех операционных системах. Oracle SQL Developer интегрируется с Apex для разработки и администрирования приложений. Oracle SQL Developer позволяет выполнять экспорт и импорт данных и структур.

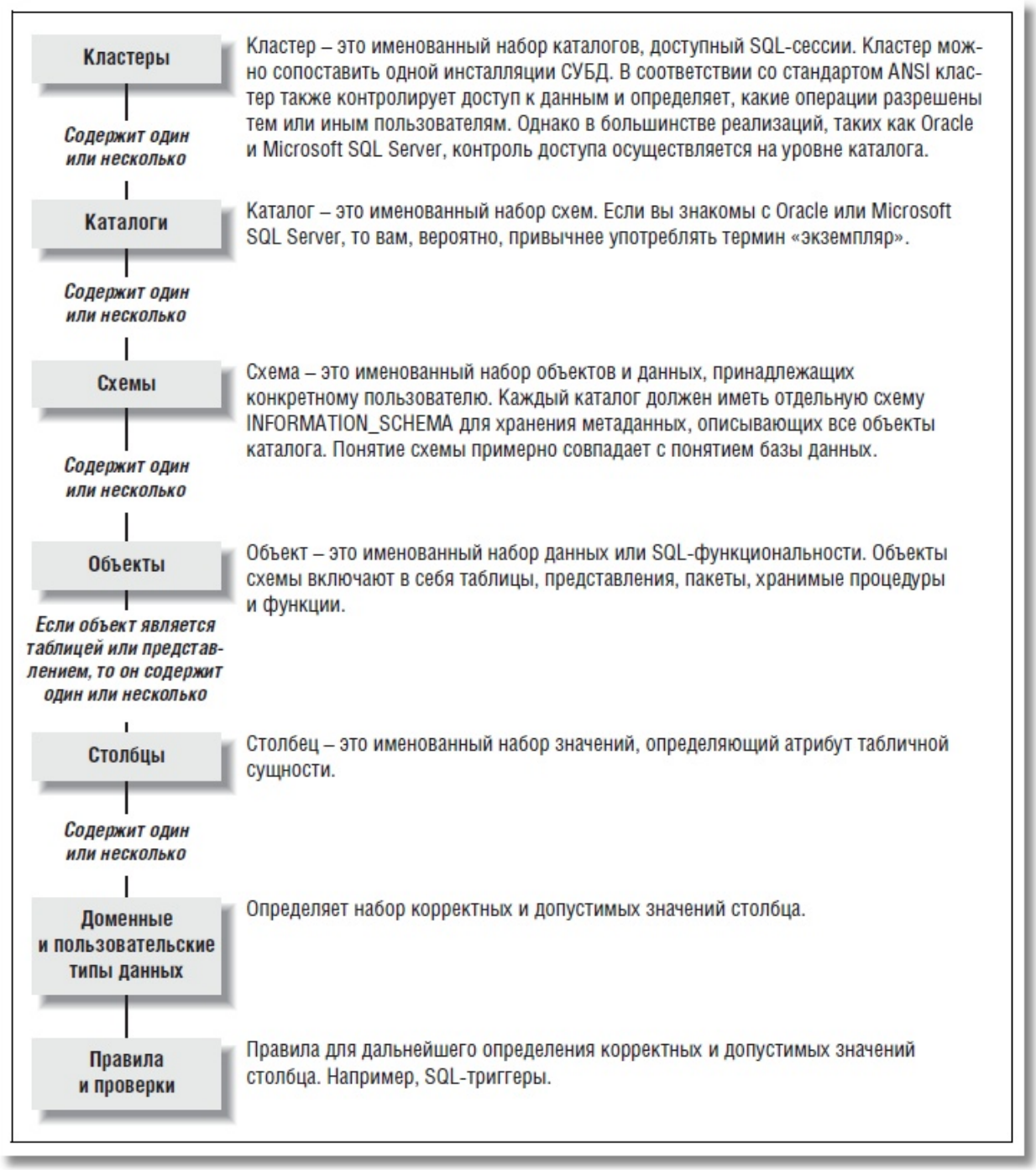
Oracle SQL Developer Data Modeler – это комплексное решение, позволяющее разработчикам проектировать реляционные модели взаимосвязей объектов для последующего преобразования их в полноценные базы данных. Продукт поддерживает логическое, реляционное, многомерное моделирование и моделирование типов данных, предлагая возможности многоуровневого проектирования и построения концептуальных диаграмм сущностей и связей. Пользователи могут создавать, расширять и модифицировать модели, а также сравнивать их с уже существующими.

В поставку Oracle Database XE также включен **Oracle Application Express** (APEX) - графическая среда разработки для создания web-приложений, основанных на базе данных поддерживаемой Oracle Database XE.

Oracle Database Express Edition- реляционная СУБД, поддерживающая все современные технологии компании ORACLE, предназначенная для обучения и поэтому она изначально содержит несколько схем данных представляющих собой примеры баз данных. Одна из этих схем – схема HR.

1.1. Схема

Схема – это именованный набор объектов и данных, принадлежащих конкретному пользователю. Понятие схемы примерно совпадает с понятием базы данных. На рисунке ниже представлена иерархия данных в стандарте SQL3.



Понятие схемы плотно связано с понятиями «пользователь» и «права доступа».

При создании пользователя автоматически создается его пустая схема, а дальше пользователь наделяется правами для работы с объектами этой схемы, например:

Пример простого создания пользователя (схемы) в БД Oracle:

--Создаем нового пользователя test с паролем 240580. Данный пример не является обязательным к исполнению с предложенными параметрами.

```
create user test IDENTIFIED BY 240580
default tablespace users
temporary tablespace temp
```

--Добавляем квоту на дисковое пространство.

```
alter user test quota 100M on users;
```

--Разрещаем создавать сессии пользователю (подключаться)

```
grant create session to test
```

--Разрешаем создавать таблицы пользователю

```
grant create table to test
```

--Разрешаем создавать процедуры

```
grant create procedure to test
```

--Разрешаем создавать триггеры

```
grant create trigger to test
```

--Разрешаем создавать представления

```
grant create view to test
```

--Разрешаем создавать счетчики

```
grant create sequence to test
```

--Разрешаем изменять таблицы, процедуры, триггеры и профиль

```
grant alter any table to test;
grant alter any procedure to test;
grant alter any trigger to test;
grant alter profile to test;
```

--Разрешаем удаления

```
grant delete any table to test;
grant drop any table to test;
grant drop any procedure to test;
grant drop any trigger to test;
grant drop any view to test;
grant drop profile to test;
```

В графическом интерфейсе SQL Developer это выглядит намного проще, но выливается в те же предложения SQL, которые Вы видите выше.

1.2. Таблицы

Таблица – это структура, которая хранит данные в реляционной базе данных. Таблица состоит из строк и столбцов. Таблица должна представлять, которую вы хотите отобразить в вашей системе. Такой сущностью может быть, например, список сотрудников вашей организации, или заказы, размещенные на продукты вашей компании. Таблицы могут быть связаны между собой посредством ключей.

Oracle позволяет задать условия, которым должны удовлетворять данные в БД. Условия задаются в декларативной форме, т. е. программист лишь задает условие, а сервер обеспечивает

его проверку при любом изменении данных. Такие условия называются правилами целостности (integrity constraint).

В Oracle поддерживаются четыре правила целостности, налагающих ограничения на данные в пределах одной таблицы:

- запрет неопределенных значений (not null),
- проверка условия (check),
- уникальность значений (unique)
- первичный ключ (primary key).
- внешний ключ (foreign key).

Правило not null запрещает использование неопределенных значений. Если столбец объявлен как not null, данные в этом столбце не могут содержать значение null.

Более сложные правила целостности можно определить при помощи предложения check. Эти правила представляют собой логические выражения, которые должны выполняться для каждой строки таблицы. Они могут включать в себя выражения, содержащие значения столбцов, операции над ними и вызовы встроенных функций. Выражения в правилах check не могут содержать вызовов пользовательских хранимых функций и операций select.

Пусть, например, в таблице employee определены поля salary, hiredate и firedate. Тогда можно было бы ввести следующие правила check:

```
salary>100 -- минимальная зарплата

hiredate>='1-Jan-2001' -- защита от неправильного ввода

firedate>hiredate -- увольнение позже приема
```

Обратите внимание, что Oracle проверяет не положительный результат проверки, а отсутствие отрицательного результата. Это значит, что если для поля задано правило salary>l00, то это поле может принимать значение null, т. к. результатом любого сравнения с null (кроме is [not] null) будет null, а не true или false.

Согласно правилу unique, в таблице не может существовать двух строк с одинаковыми значениями столбца или набора столбцов. При этом неопределенные значения столбцов не считаются одинаковыми, т. е. две и более строк могут содержать неопределенные (null) значения во всех столбцах, объявленных как unique. Пусть в таблице test существует правило целостности unique, наложенное на столбец first, и таблица пуста:

```
insert into test (first) values (1) -- OK
insert into test (first) values (1) -- ошибка
insert into test (first) values (null) -- OK
```

```
insert into test        (first)   values     (null)    -- OK
```

Пусть теперь правило unique задано для пары столбцов first и second:

```
insert into test        (first,second)      values  (1,2) -- OK
insert into test        (first,second)      values  (1,2) -- ошибка
insert into test        (first,second)      values  (l,null) -- OK
insert into test        (first,second)      values  (l,null) -- ошибка
insert into test        (first,second)      values  (null,null) -- OK
insert into test        (first,second)      values  (null,null) -- OK
```

Для того, чтобы поддержать правило целостности unique, Oracle использует индекс. Если подходящего индекса нет, то Oracle автоматически создает уникальный индекс по тому набору полей, на который наложено правило unique. Если правило будет уничтожено или запрещено, то автоматически созданный индекс будет уничтожен. Если для поддержания правила unique используется готовый индекс, то этот индекс не обязан быть уникальным. Кроме того, в этом случае при уничтожении или запрещении правила индекс не уничтожается, соответственно, нет необходимости пересоздавать его при включении правила.

Правило primary key задает первичный ключ. Каждая строка таблицы однозначно определяется значениями столбца или набора столбцов, входящих в первичный ключ. Для обеспечения правила целостности primary key Oracle неявно создает правила not null для каждого столбца, входящего в ключ, и правило unique для набора столбцов. Правило primary key также может использовать готовый индекс, не обязательно уникальный. Таблица может иметь не более одного первичного ключа.

Правило целостности может иметь имя. При создании безымянного правила целостности Oracle автоматически генерирует для него уникальное имя. Если индекс, с помощью которого поддерживается правило unique или primary key, создан автоматически, то его имя совпадает с именем правила целостности. Поскольку индекс не может строиться более чем по 16 полям, одно правило unique или primary key не может включать в себя более 16 столбцов. Кроме того, общая длина полей, входящих в правило primary key или unique, не может превышать 3128 символов.

В Oracle для связей между таблицами существуют внешние ключи (foreign key).

Эти поля описывается следующим образом:

```
foreign key (<внешний ключ>) references <родительская таблица> (<ключ>) [on delete (cascade | set null)]
```

Внешний ключ — это поле или набор полей, перечисленных через запятую, количество и типы которых совпадают с количеством и типом полей первичного ключа в родительской

таблице. Поле, объявленное как foreign key, может иметь неопределенное значение (null), если только для него явно не задано правило not null. По умолчанию правило foreign key запрещает удаление строк, на которые есть ссылки, из родительской таблицы, т. е. реализует вариант “update, delete restrict”. Если требуется создать внешний ключ с правилами “delete cascade” и “set null”, это надо указать явно во фразе on delete.

Для поля (или набора полей), объявленных как foreign key, Oracle не создает индекс автоматически. Если это необходимо создайте индекс вручную.

Определение внешнего ключа позволяет реализовать отношение (dependence) много-к-одному (many-to-one) или один-к-одному (one-to-one).

Правила ссылочной целостности (referential integrity) позволяют добавлять или обновлять строки в зависимости от связей таблиц (значений первичных и внешних ключей). Примером могут служить таблицы, описывающие сотрудников и отделы: значение отдела, в котором работает сотрудник, должно представлять собой номер существующего отдела. В этом случае таблица отделов называется родительской, а таблица сотрудников — дочерней.

Oracle поддерживает три варианта действий, которые производятся над дочерней таблицей при изменении данных родительской таблицы — “update, delete restrict”, “delete cascade” и “set null”.

В случае “update, delete restrict” запрещается удалять или изменять ключевое значение в родительской таблице, если в дочерней таблице есть строки, ссылающиеся на изменяемую строку. В случае “delete cascade” при удалении строки из родительской таблицы автоматически будут удалены все строки дочерней таблицы, ссылавшиеся на удаленную строку, т. е., в нашем примере, при удалении отдела автоматически будут удалены все сотрудники этого отдела. В случае “set null” при удалении строки из родительской таблицы во всех строках дочерней таблицы, ссылавшихся на нее, значение поля-ключа будет установлено в null.

1.3. Представления

Представление – это виртуальная таблица, определяемая сохраненным запросом к БД. Результатом выполнения запроса, как известно, является таблица, что позволяет использовать представления в командах insert, update, delete и select так же, как и реальные таблицы БД. Данные в виртуальной таблице обычно изменить нельзя. При использовании представления в команде выборки данных СУБД Oracle подставляет его текст в запрос и затем разбирает и выполняет запрос целиком. В представлении нельзя определять правила целостности.

Представления используются для того, чтобы скрыть физическую структуру данных, обеспечив большую гибкость приложения, чтобы упростить запросы или чтобы ограничить доступ к данным. Представление не требует места в БД, кроме места на хранение своего определения в словаре данных.

Создается представление командой create view:

```
create [or replace] [force] view <имя> [(<имя столбца> {<имя столбца>})] as <запрос>
```

Если запрос некорректен (содержит ошибку или пользователь, создающий представление, не имеет привилегий на обращение к указанным в запросе объектам), представление не будет создано. Однако если указать параметр force, то представление будет создано в любом случае, а попытка обратиться к нему приведет к ошибке:

ORA-04063: view "<имя>" has errors

Представление может становиться ошибочным и в результате изменения структуры объектов, к которым оно обращается.

Столбцы представления получают те имена, которые возвращает SQL-запрос.

Уничтожается представление командой drop:

```
drop view <имя>
```

1.4. Индексы

Индексы — дополнительные объекты, связанные с таблицами. Индексы предназначены для ускорения поиска в таблицах.

Индекс строится по одному или нескольким полям; в индекс может входить до 16 полей. Порядок перечисления полей в составном индексе имеет значение. Если значение всех полей, по которым построен индекс, в некоторой строке не определено (равно null), то такая строка в индекс не входит.

Oracle допускает создание уникальных индексов, т. е. индексов, в котором каждый набор значений индексированных полей уникален. Однако создавать такие индексы вручную не рекомендуется, т. к. уникальность значения — понятие логическое, а не физическое. Oracle автоматически создает уникальные индексы при задании правил целостности unique и primary key.

Создается индекс командой create index, синтаксис которой приведен ниже:

```
create [ unique | bitmap ] index <имя> on <имя таблицы> (<имя столбца> {,<имя столбца>})
```

Ключевое слово unique позволяет создать уникальный индекс.

Ключевое слово bitmap создает индекс, построенный на битовых картах. Такие индексы некоторых случаях могут быть значительно эффективнее и компактнее традиционных индексов, основанных на B-деревьях, но без точного понимания их работы применять их нельзя.

Необходимо помнить, что индекс ускоряет операцию выборки данных из таблицы – все остальные действия с этой таблицей замедляются!

Индекс в любой момент может быть уничтожен — это не повлияет на работоспособность приложений, хотя может существенно их замедлить. Уничтожается индекс командой drop index:

```
drop index <имя>
```

1.5. Последовательности

Последовательности — объекты БД, генерирующие неповторяющиеся номера. Эти объекты используются для генерации первичных ключей.

Создаются последовательности командой create sequence:

```
create sequence <имя> [start with <число>] [increment by <число>] [maxvalue <число>] [minvalue <число>] [cycle | nocycle] [cache <число> | nocache] [order | noorder]
```

Значения start with и increment by задают начальное значение последовательности и шаг, с которым будет увеличиваться значение. Задание начального значения и шага, отличных от 1, может оказаться полезным в распределенной базе данных — если, например, на двух серверах заданы начальные значения 1 и 2 и шаг 10, то ключи, сгенерированные разными серверами, будут уникальны в пределах обоих серверов.

Oracle не следит, чтобы все значения, выбранные из последовательности, были как-то использованы. Если, например, транзакция, выбравшая очередное значение, будет отменена, то выбранное значение будет утеряно, т.е. последовательность не обязательно содержит все значения подряд.

Параметры minvalue и maxvalue определяют минимальное и максимальное значения, которые могут быть выбраны из последовательности. Если минимальное (максимальное) значение будет достигнуто, то очередное обращение к последовательности приведет к ошибке:

```
ORA-08004: sequence <имя>.NEXTVAL exceeds MAXVALUE and cannot be instantiated
```

Если указан параметр cycle, то по достижении одной из границ последовательность будет снова генерировать числа, начиная с другой границы. Параметр cache управляет кэшированием значений. Если этот параметр задан, то обращения к последовательности будут кэшироваться, т. е. за один раз будет выбираться сразу несколько значений. Это ускоряет работу, но приводит к потере большого количества значений, если за сессию происходит мало обращений к последовательности.

В случае, если в команде create sequence указано ключевое слово order, Oracle гарантирует, что выбранное из последовательности значение будет больше (при отрицательном шаге — меньше), чем все предыдущие. В противном случае порядок может быть нарушен, но, как правило, это неважно.

Последовательность имеет два атрибута: nextval — следующее уникальное значение и currval — последнее выбранное значение. Обратиться к атрибуту currval можно только в том

случае, если в текущей сессии уже было обращение к nextval. Запомните: значение nextval для всех сессий в конкретный момент времени - одно, а вот значение currval для каждой сессии - свое.

Для обращения к последовательности (а также к любым объектам, не являющимся данными таблицы, например, функциям) применяется конструкция select ... from dual, впрочем вы можете указать любую доступную таблицу, но при этом предложение SQL будет работать несколько дольше.

```
select seq_emp.currval from dual
ORA-08002: sequence SEQ_EMP.CURRVAL is not yet defined in this session
select seq_emp.nextval from dual
```

1.6. Триггеры (Материал из Википедии)

Три́ггер (англ. *trigger*) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определенном столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события; например, до добавления записи) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись (конечно, при условии, что событие — не удаление записи). Некоторые СУБД накладывают ограничения на операторы, которые могут быть использованы в триггере (например, может быть запрещено вносить изменения в таблицу, на которой «висит» триггер, и т. п.).

Кроме того, триггеры могут быть привязаны не к таблице, а к представлению (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

В некоторых серверах триггеры могут вызываться не для каждой модифицируемой записи, а один раз на изменение таблицы. Такие триггеры называются табличными.

1.7. Синонимы

Синоним — дополнительное имя для объекта БД — таблицы, представления, последовательности или хранимой процедуры. Синоним не требует физического пространства, кроме как для своего определения в словаре данных.

На практике синонимы заводятся с разными целями:

- неявно запретить изменение структуры таблицы т.к. в предложениях SQL ALTER/DROP TABLE ссылаться можно только на истинное имя таблицы
- присвоить таблице имя, больше подходящее ее содержанию;
- упростить имя таблицы в запросе, например, OBJ вместо SYS.OBJ$;
- замаскировать обращение к таблице в другой БД или в другой схеме для придания гибкости кода.

Создается синоним командой

create synonym:

```
create [public] synonym <имя> for <имя объекта>
```

Уничтожается синоним командой drop:

```
drop synonym <имя>
```

Синоним может быть переименован командой rename:

```
rename <имя> to <новое имя>
```

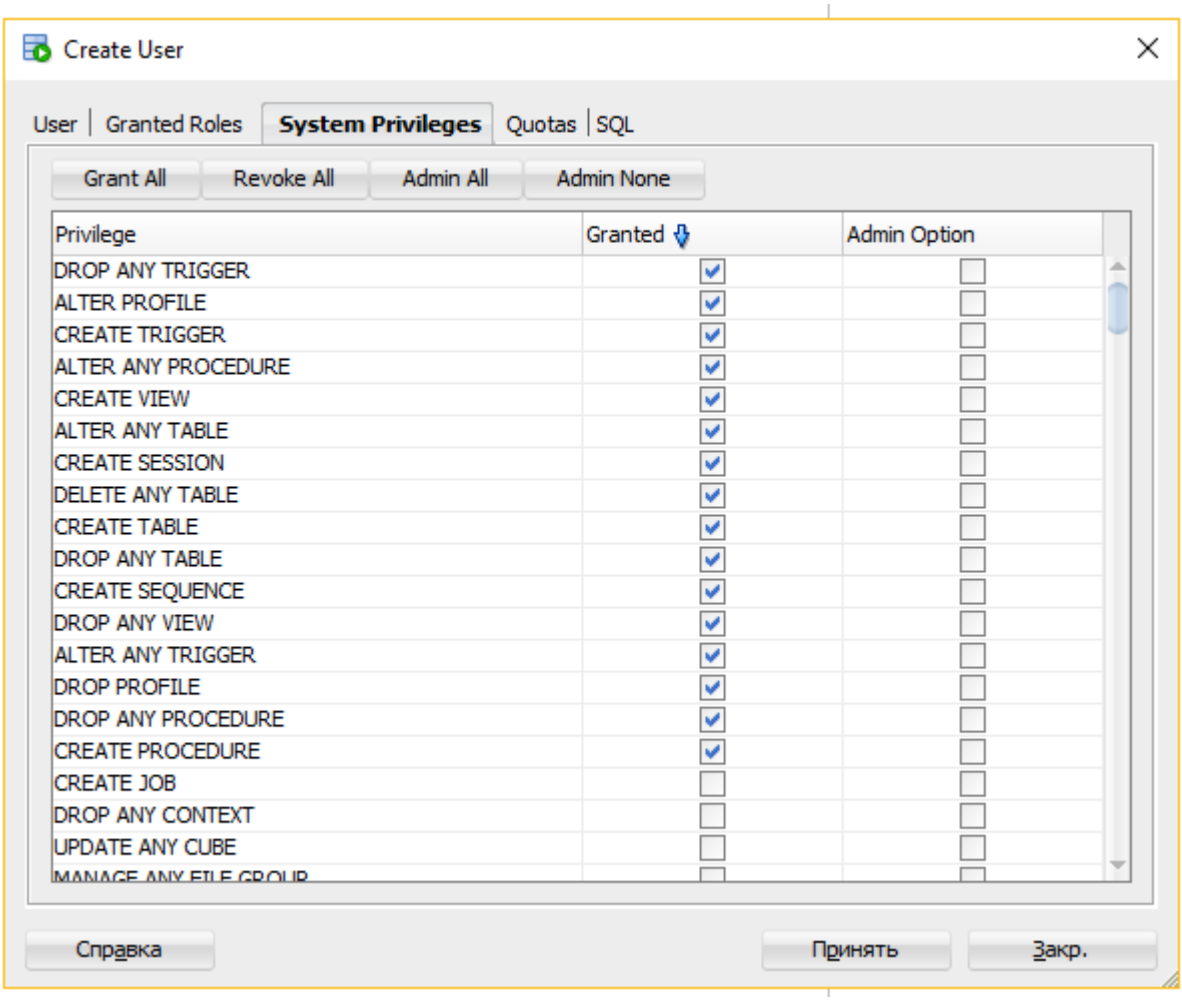
1.8. Соединение таблиц

Для выборки данных из нескольких таблиц выполняется их соединение, т.е. действие, возвращающее одну таблицу, состоящую из комбинации данных двух исходных таблиц. Надо понимать, что существует большое количество способов комбинирования данных (записей и столбцов) нескольких таблиц. Самое простое соединение – «декартово произведение», когда мы получаем все возможные комбинации записей соединяемых таблиц. Пусть в одной таблице будет 5 записей, а в другой 15, тогда количество их комбинаций будет 5x15=75, поэтому такое соединение и называется «декартовым ПРОИЗВЕДЕНИЕМ». Реализация такого соединения предельно проста: достаточно просто перечислить соединяемые таблицы в предложении FROM оператора SELECT.

Практическая часть

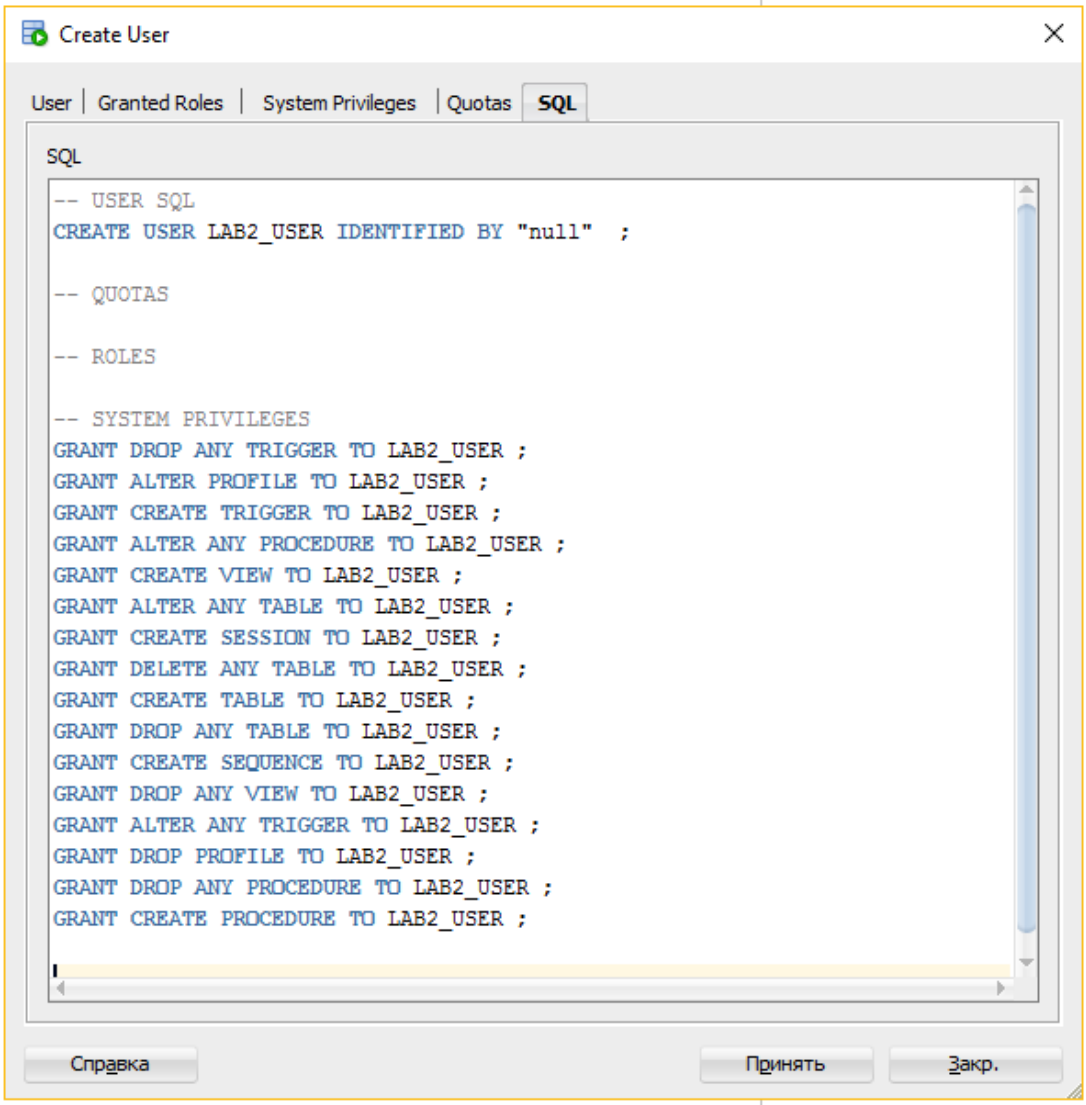
1. Создание пользователя и его схемы

Создадим пользователя LAB2_USER (можно использовать другое имя) через графический интерфейс SQL Developer. Наделим его следующими правами (рис. 1):



(рис. 1 – Права нового пользователя)

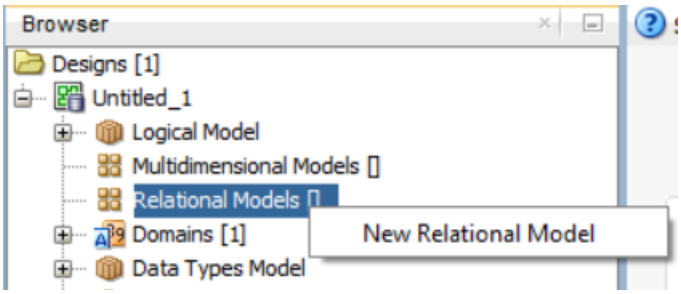
Код создания пользователя и наделения его правами на языке SQL выглядит следующим образом (рис. 2):



(рис. 2 SQL-код создания нового пользователя и добавления ему прав)

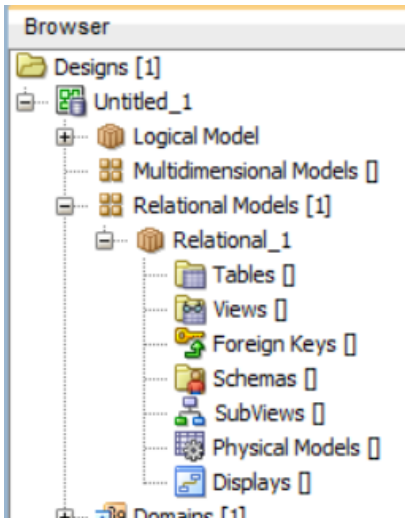
2. Проектирование схемы базы данных

Проектирование базы данных производится в программе SQL DataModeler в разделе «Relation Models» (рис. 3). Нужно создать таблицы *«Клиенты»*, *«Заказы»*, *«Города»*, *«Аэропорты»*, *«Рейсы»*, *«Перевозчики»*. **Названия лучше использовать латинские.**



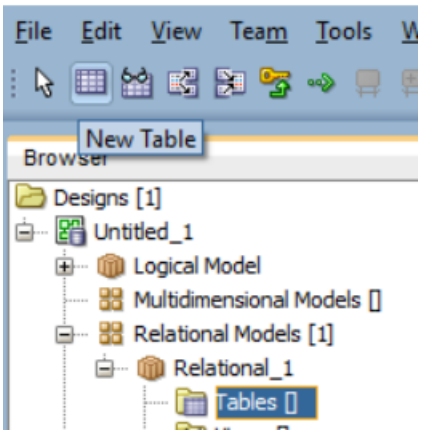
(рис. 3 Создание новой реляционной модели)

После создания нового проекта, появится следующее меню: (рис. 4)

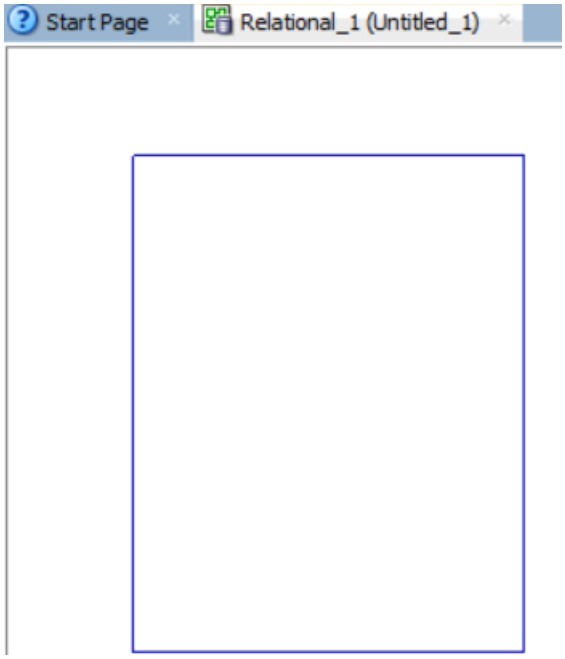


(рис. 4 – Меню модели 1)

Далее требуется перейти о вкладку Tables и через специальную кнопку создать таблицы. (рис. 5, 6)

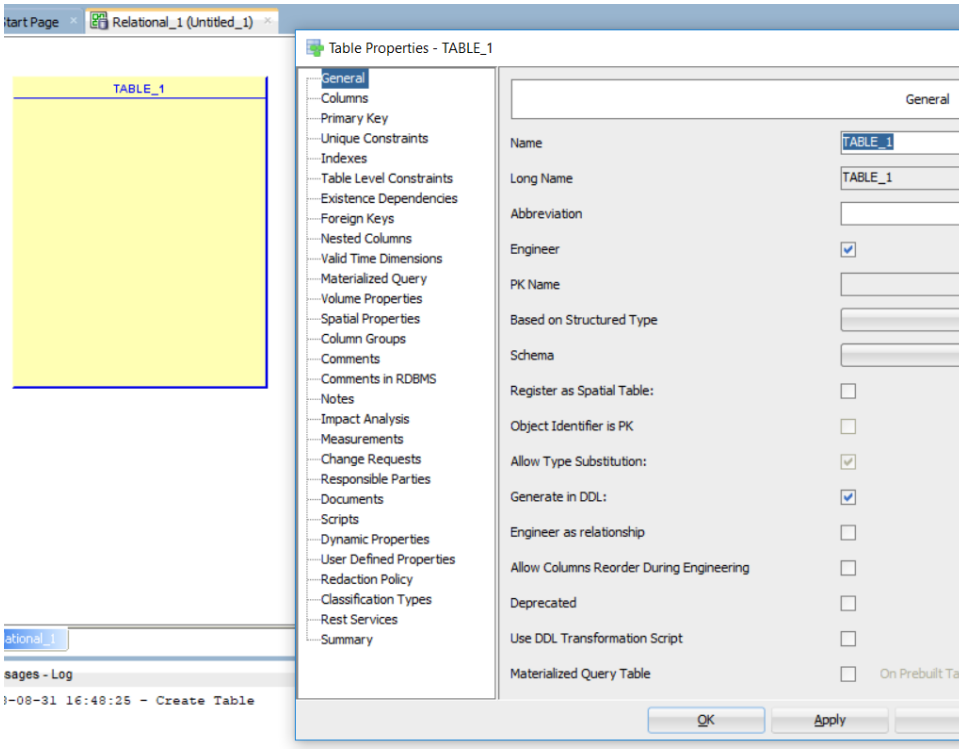


(рис. 5 – Создание новой таблицы)



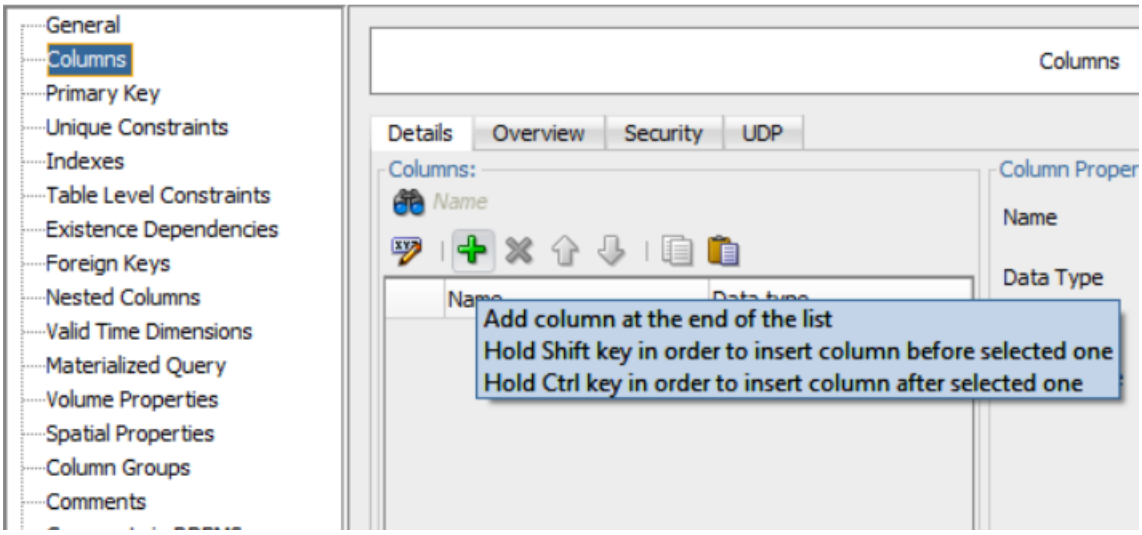
(рис. 6 Отрисовка таблицы в графической области)

Для создания таблицы ее стоит нарисовать (растянуть рамку). По завершении рисования откроется следующее меню редактирования. (рис. 7)

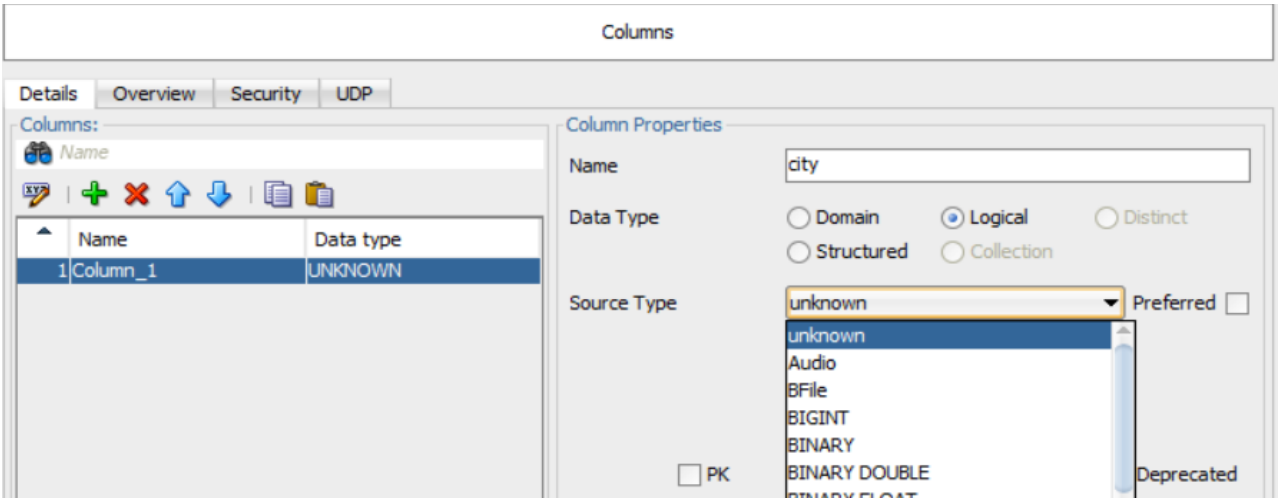


(рис. 7 – Свойства таблицы)

Далее для таблицы меняется имя (выделено на рис. 7), открываем вкладку columns и добавляем нужные столбцы в таблицу. (рис. 8)



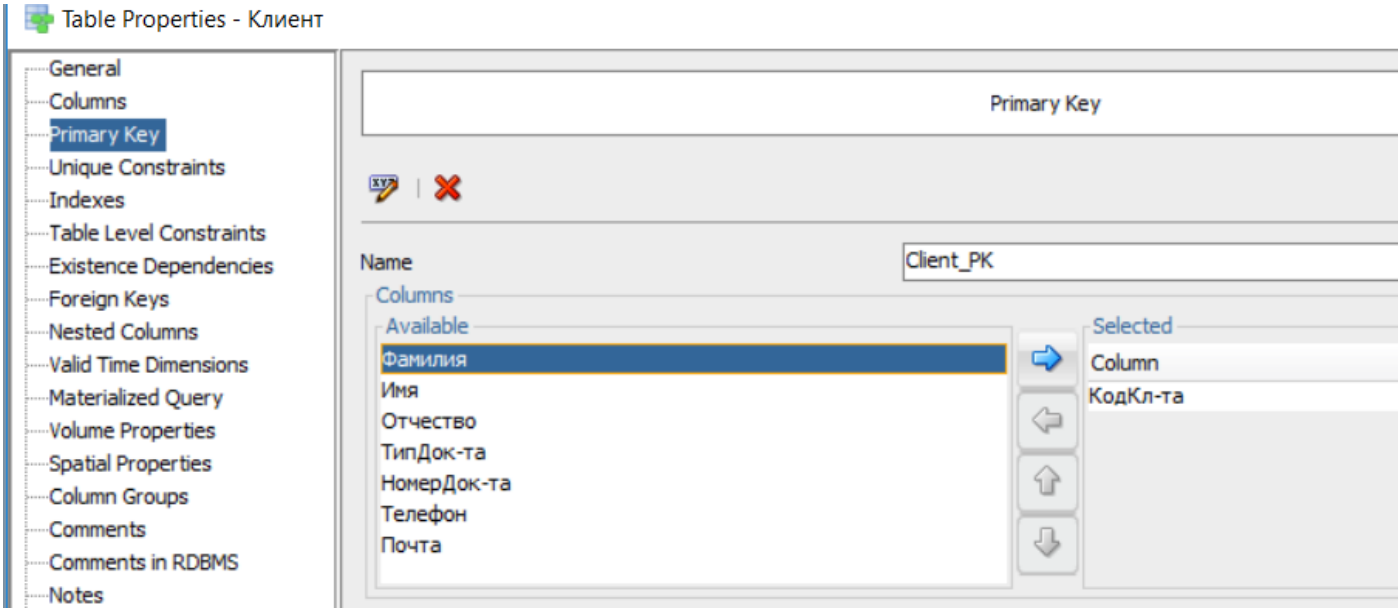
(рис. 8 Добавление столбцов в таблицу)



(рис. 9 – Настройка данных в столбце)

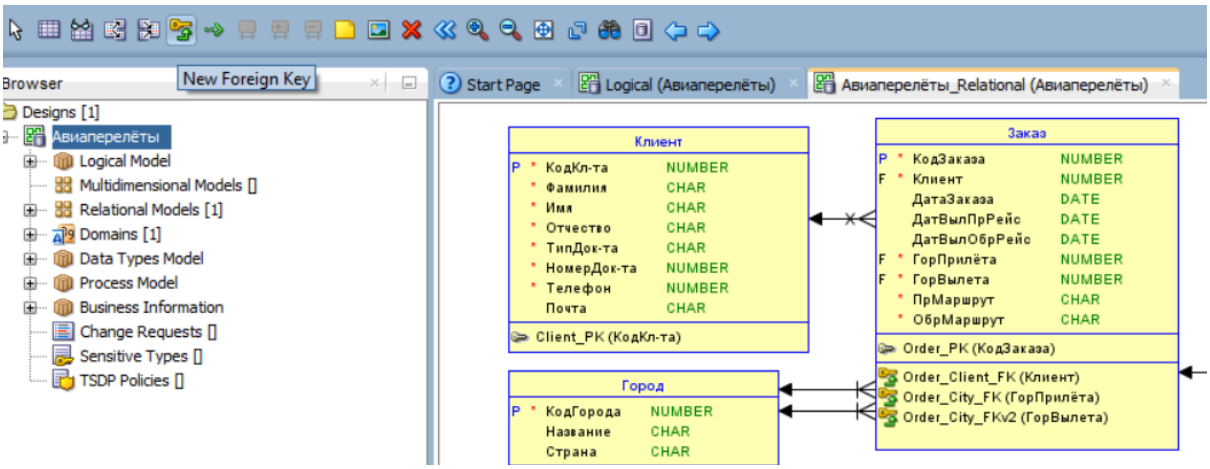
После изменения названия столбца можно изменить тип данных в этом столбце (для этого стоит отметить Data Type как Logical, после этого в Source Type можно будет выбрать нужный нам тип. (рис. 9)

Для каждой таблицы следует выбрать Primary Key, в соответствующем меню. (рис. 10)



(рис. 10 – Выбор Primary Key для таблицы)

Для каждого нового внешних ключа его нужно создать через кнопку в контекстном меню. (рис. 11) Для этого следует нажать поочередно на столбец, который будет основным, затем на столбец, который будет зависимым. **Схему нужно нарисовать на латинице.**



(рис. 11 Связи между таблицами)

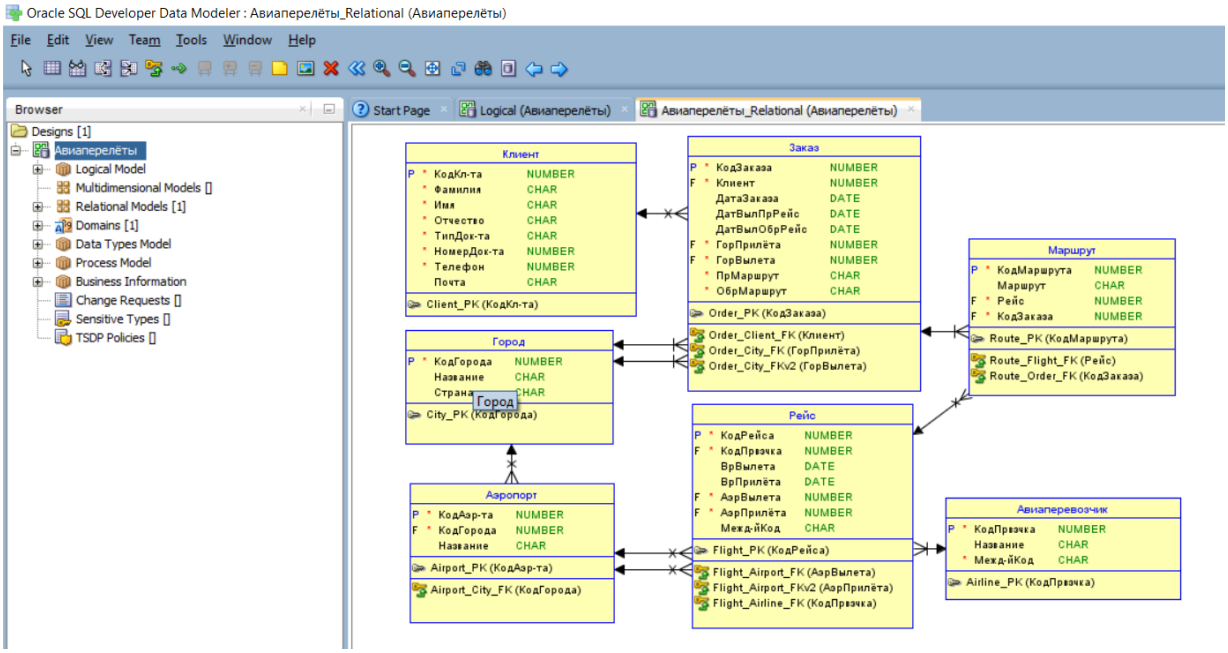
Задание:

1. Необходимо придумать какие поля и таблицы нужно добавить в текущую схему, так чтобы получаемые данные из таблиц позволили в дальнейших лабораторных работах создать маршрут-квитанцию к заказу.

2. Необходимо продумать каким образом будут сформированы рейсы с пересадками.

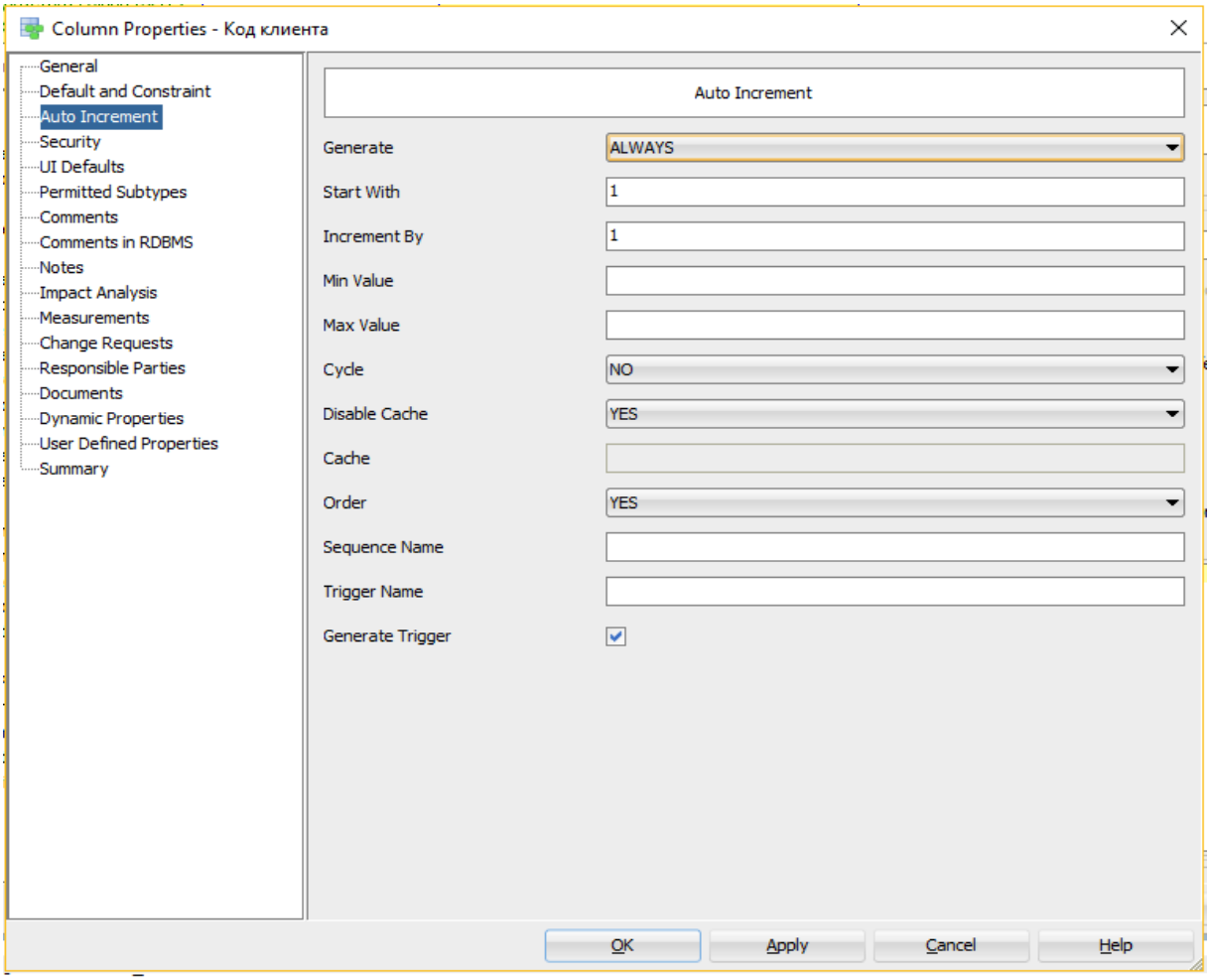
Если у вас не было семинара с проработкой схемы данных, и нет мыслей по тому что можно придумать или добавить, советуем Вам обратиться к вашим однопоточникам. Одним из

примеров конечной схемы данных представлен на рис. 12. **Настоятельно рекомендуем придумать свой вариант.**



(рис. 12 Итоговая схема (пример))

Для каждой таблицы должны быть указаны первичные ключи, внешние ключи, а также созданы триггеры и последовательности, позволяющие автоматически генерировать первичные ключи таблиц. (рис. 13)



(рис. 13 Проверка свойств таблицы)

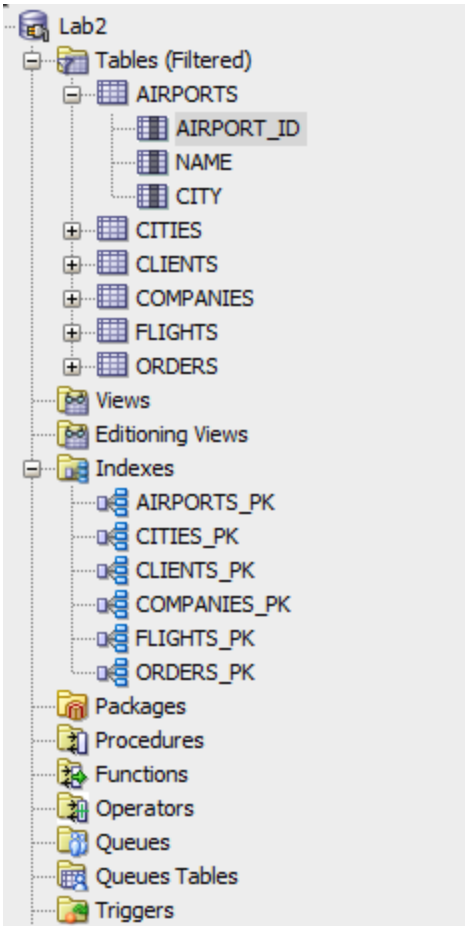
По завершении работы надо сохранить все, а дальше экспортировать созданную таблицу в SQL Developer.

Для экспорта схемы базы данных в SQL Developer воспользуемся кодом SQL, сгенерированным программой.

(View -> DDL File Editor -> Generate)

3. Перенос схемы БД из SQL Data Modeler в SQL Developer

После выполнения кода SQL в схеме созданного ранее пользователя LAB2_USER, получим спроектированную базу данных. (Создаем новое подключение на нашего пользователя, только после этого SQL код выполняется корректно) (рис. 14)



(рис. 14 – Вид перенесенной БД в SQL Developer)

4. Заполнение таблиц в БД

Добавим данные в таблицы:

«*Аэропорты*» – 30 наименований,

«*Города*» – не менее 20 наименований.,

«*Клиенты*» – не менее 20 клиентов,

«*Авиакомпании*» – не менее 10 наименований,

«Заказы» – не менее 15 заказов.

Дополнительные таблицы, созданные в рамках лабораторной работы заполняются по вашему усмотрению, но не менее 1 записи в таблице.

Добавлять записи в таблицы можно вручную или импортируя из других файлов.

Таблицы были заполнены данными. (рис. 15)

AIRPORT_ID	NAME	CITY
1	ATL port	4
2	PEK port	2
3	DXB port	3
4	ORD port	2
5	HND port	3
6	LHR port	5
7	LAX port	1
8	HKG port	1
9	CDG port	3
10	DFW port	2
11	IST port	5
12	FRA port	1
13	PVG port	3
14	AMS port	5
15	JFK port	4

COMPANY_ID	NAME
S7	S7 Airlines
RN	Ryanair
LH	Lufthansa
AA	Airastana
BA	British Airways

CITY_ID	NAME	COUNTRY
1	New-York	USA
2	Moscow	Russia
3	Paris	France
4	Tokio	Japan
5	London	GB

CLIENT_ID	NAME	SURNAME	PASSPORT
1	Ura	Ren	12344321

	ORDER_ID	CLIENT	ORDER_DATE	ORDER_S...	CITY_FROM	CITY_TO
1	1	1	11.12.17 22:14:46	ok	1	2

(рис. 15 – Пример созданных таблиц)

Добавление строк в таблицу «*Рейсы*» будем делать при помощи SQL. Для этого использован следующий код: (рис. 16)

```
create SEQUENCE dep_time_min
START WITH 10
INCREMENT BY 1
MAXVALUE 1140
MINVALUE 0
CYCLE
NOCACHE
ORDER;

create SEQUENCE flight_code
START WITH 1000
INCREMENT BY 1
MAXVALUE 9999
MINVALUE 1000
NOCYCLE
NOCACHE
ORDER;

INSERT INTO FLIGHTS (FLIGHT_ID, COMPANY, FROM_PORT, TO_PORT, DEP_DATE, ARR_DATE)
SELECT concat(COM.COMPANY_ID, flight_code.nextval), COM.COMPANY_ID FID, AIR_FROM.AIRPORT_ID AIRFR, AIR_TO.AIRPORT_ID AIRTO,
TO_DATE(concat('12.11.2017 ', concat(FLOOR(dep_time_min.nextval/60),
concat(':', concat(MOD(dep_time_min.currval, 60), ':00')))), 'DD.MM.YYYY HH24:MI:SS'),
TO_DATE(concat('12.11.2017 ', concat(4 + FLOOR(dep_time_min.nextval/60),
concat(':', concat(MOD(dep_time_min.currval, 60), ':00')))), 'DD.MM.YYYY HH24:MI:SS')
FROM COMPANIES COM, AIRPORTS AIR_FROM, AIRPORTS AIR_TO
WHERE AIR_FROM.CITY <> AIR_TO.CITY
```

(рис. 16 – функции заполнения данных и построения отчета)

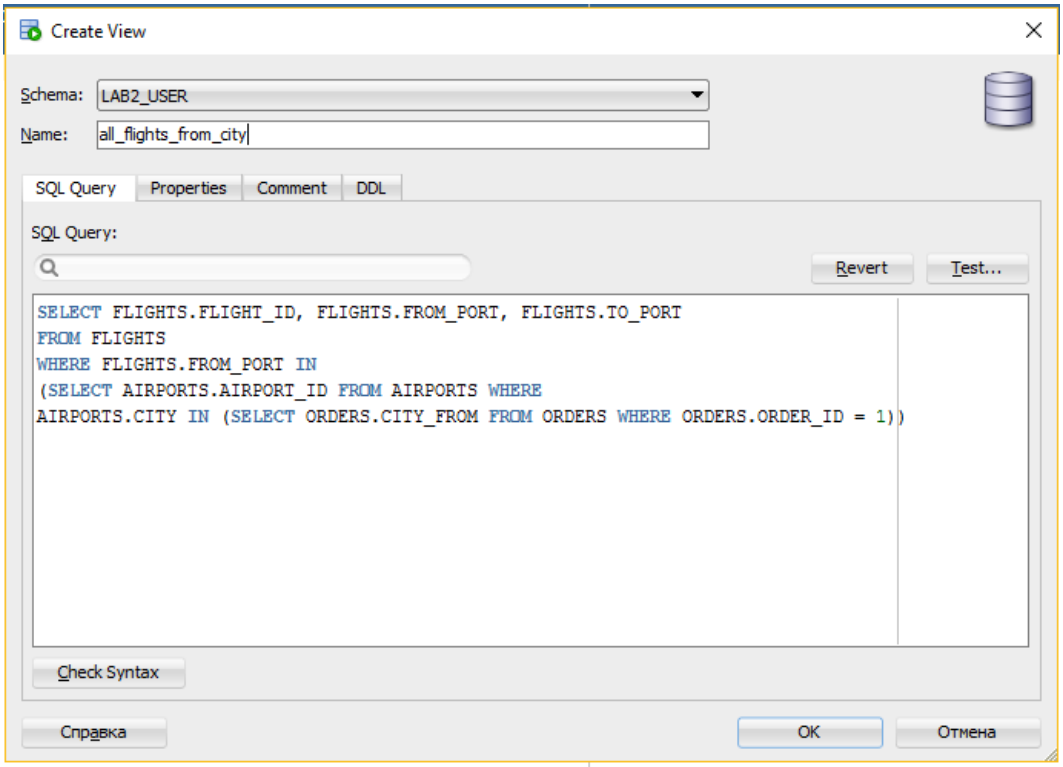
Результат заполнения таблицы *«Рейсы»* данными: (рис. 17)

	FLIGHT_ID	COMPANY	FROM_PORT	TO_PORT	DEP_DATE	ARR_DATE
1	AA1000	AA	1	2	12.11.17 00:10:00	12.11.17 04:10:00
2	BA1001	BA	1	2	12.11.17 00:11:00	12.11.17 04:11:00
3	LH1002	LH	1	2	12.11.17 00:12:00	12.11.17 04:12:00
4	RN1003	RN	1	2	12.11.17 00:13:00	12.11.17 04:13:00
5	S71004	S7	1	2	12.11.17 00:14:00	12.11.17 04:14:00
6	AA1005	AA	1	3	12.11.17 00:15:00	12.11.17 04:15:00
7	BA1006	BA	1	3	12.11.17 00:16:00	12.11.17 04:16:00
8	LH1007	LH	1	3	12.11.17 00:17:00	12.11.17 04:17:00
9	RN1008	RN	1	3	12.11.17 00:18:00	12.11.17 04:18:00
10	S71009	S7	1	3	12.11.17 00:19:00	12.11.17 04:19:00
11	AA1010	AA	1	4	12.11.17 00:20:00	12.11.17 04:20:00
12	BA1011	BA	1	4	12.11.17 00:21:00	12.11.17 04:21:00
13	LH1012	LH	1	4	12.11.17 00:22:00	12.11.17 04:22:00
14	RN1013	RN	1	4	12.11.17 00:23:00	12.11.17 04:23:00
15	S71014	S7	1	4	12.11.17 00:24:00	12.11.17 04:24:00
16	AA1015	AA	1	5	12.11.17 00:25:00	12.11.17 04:25:00
17	BA1016	BA	1	5	12.11.17 00:26:00	12.11.17 04:26:00
18	LH1017	LH	1	5	12.11.17 00:27:00	12.11.17 04:27:00
19	RN1018	RN	1	5	12.11.17 00:28:00	12.11.17 04:28:00
20	S71019	S7	1	5	12.11.17 00:29:00	12.11.17 04:29:00
21	AA1020	AA	1	6	12.11.17 00:30:00	12.11.17 04:30:00
22	BA1021	BA	1	6	12.11.17 00:31:00	12.11.17 04:31:00
23	LH1022	LH	1	6	12.11.17 00:32:00	12.11.17 04:32:00
24	RN1023	RN	1	6	12.11.17 00:33:00	12.11.17 04:33:00
25	S71024	S7	1	6	12.11.17 00:34:00	12.11.17 04:34:00
26	AA1025	AA	1	7	12.11.17 00:35:00	12.11.17 04:35:00
27	BA1026	BA	1	7	12.11.17 00:36:00	12.11.17 04:36:00
28	LH1027	LH	1	7	12.11.17 00:37:00	12.11.17 04:37:00

(рис. 17 Отчет по запросу на рис. 16)

5. Представления

Создадим представление «Все рейсы из города вылета». Город вылета берется из заказа, для каждого аэропорта из города вылета берутся рейсы из таблицы *«Рейсы»*. (рис. 18)



(рис. 18 – SQL-код запроса)

.

Задание на самостоятельное выполнение

1. Создание представления

А) Создайте последовательность для генерации уникального кода рейса и при помощи него заполните т таблицу «Рейсы».

Б) Генерируйте номер рейса из случайной последовательности при помощи функции dbms_random.value(0,9999). Это подход красивый и не требующий дополнительно последовательности, но он имеет свой недостаток – код рейса может получиться неуникальным. Напишите предложение SELECT, которое будет выводить одинаковые значения номеров рейса. Напишите предложение DELETE, которое будет удалять одно из одинаковых значений

2. Предусмотрите контроль вводимой информации для полей таблиц, где это требуется. К примеру: Тип документа, email, Телефон, Номер паспорта, дата и время и тому подобное.