Московский Государственный Технический Университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Компьютерные системы и сети»

Фомин М.М.

**Методическое пособие по лабораторной работе №2**

**Создание базы данных в среде ORACLE**

Москва 2016

**Цель работы –** Научиться создавать БД в СУБД Oracle11g.

## Введение

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.

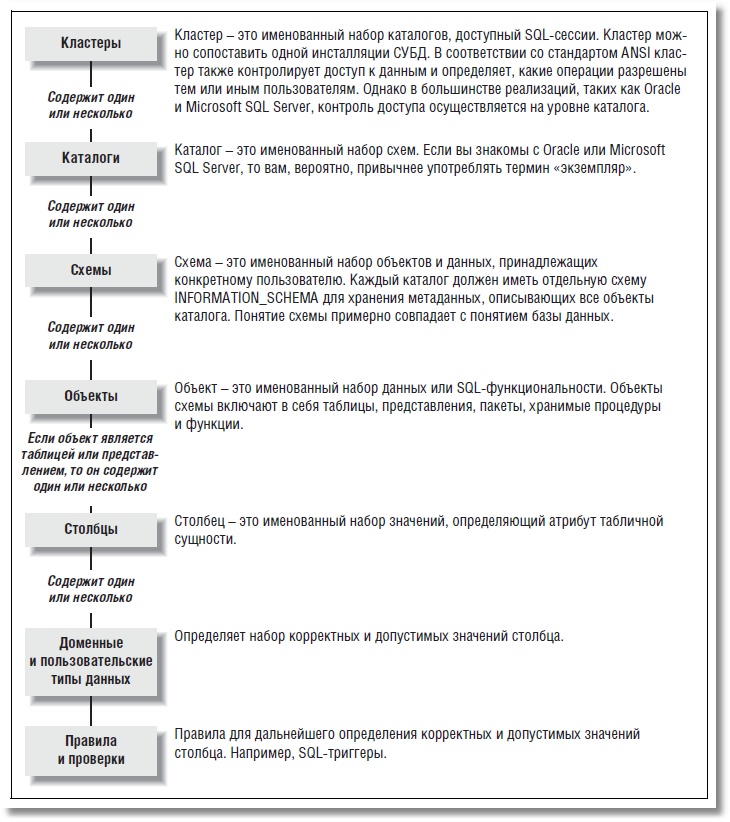
В поставку Oracle Database XE также включен **Oracle Application Express** (APEX) - графическая среда разработки для создания web-приложений, основанных на базе данных поддерживаемой Oracle Database XE.

**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.

# Теоретическая часть

# 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

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.

# 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.

# 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**>**

# 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**>**

# 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

# 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.

# 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**>**

# Соединение таблиц

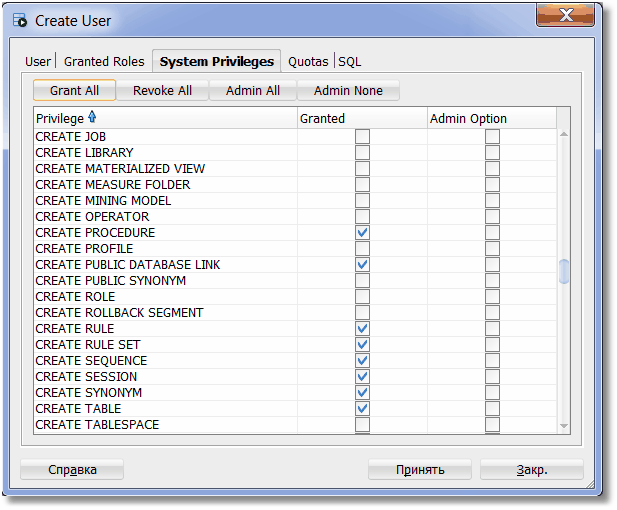
Для выборки данных из нескольких таблиц выполняется их соединение, т.е. действие, возвращающее одну таблицу, состоящую из комбинации данных двух исходных таблиц. Надо понимать, что существует большое количество способов комбинирования данных (записей и столбцов) нескольких таблиц. Самое простое соединение – «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 5х15=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

# Создание пользователя и его схемы.

Создайте пользователя с произвольным именем и наделите его правами как в п. 2.1. Обратите внимание, что в ORACLE понятие пользователя и его схемы неразрывно связаны – создавая пользователя вы одновременно создаете и его схему (сначала пустую).

Создайте пользователя через графический интерфейс SQL Developer и посмотрите, как это будет выглядеть на языке SQL.

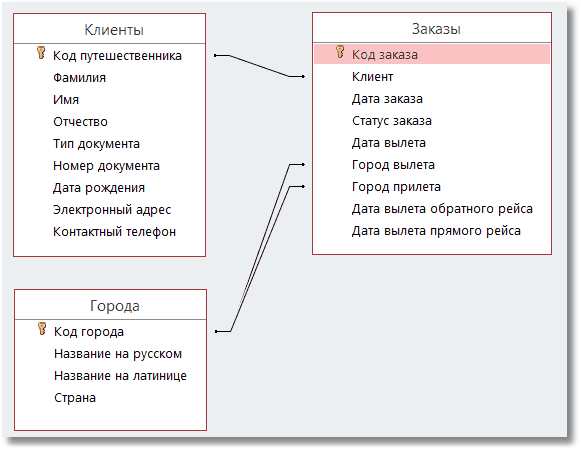


# Проектирование схемы базы данных

В данной лабораторной работе Вам предстоит разработать фрагмент базы данных системы бронирования авиационных билетов, то есть для автоматизации первого бизнес процесса бронирования – выбор маршрута путешествия. Итак, клиент задает город вылета и город назначения, определяет дату перелета и указывает дату обратного перелета, если таковой необходим. Система подбирает подходящие рейсы по маршруту перелета и показывает их клиенту для выбора.

Для осуществления этого процесса надо учитывать, что рейс определяется датой, аэропортом вылета, временем вылета, аэропортом прилета, авиаперевозчиком. В каждом городе может быть несколько аэропортов. На данном этапе мы не заморачиваемся, каким образом данные о клиенте, аэропортах, рейсах и т.д. попадают в систему. Нам нужно выделить сущности, атрибуты и определить связи между ними.

Сначала надо определиться с сущностями «клиент» и «заказ». Что же вводит клиент? Каковы атрибуты заказа? Ответив на эти вопросы, мы получим следующую схему:

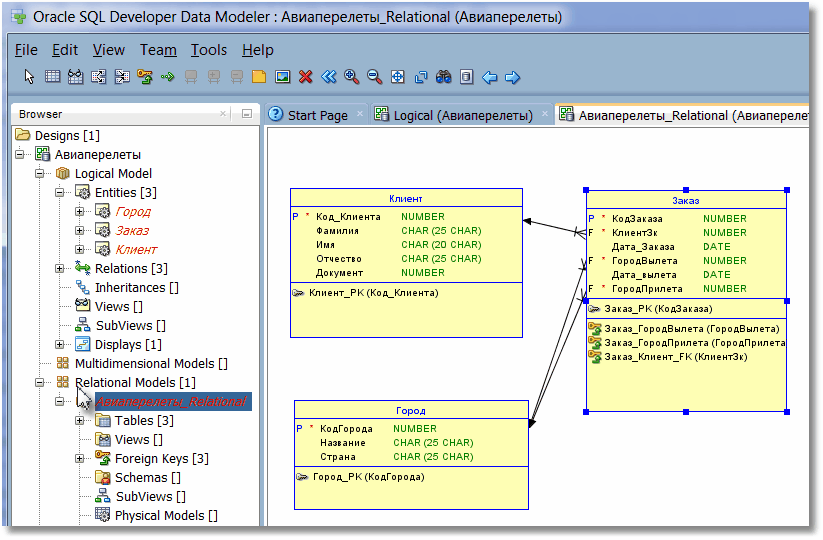


Откройте SDL DataModeler и проделайте следующие действия:

1. Создайте проект
2. Создайте таблицы
3. Создайте поля в таблицах
4. Создайте связи между таблицами согласно рисунку выше.

Обратите внимание на то, что предлагается создавать таблицы в разделе (окне) «Relation Model» т.к. создание сущностей в разделе Logical Model весьма специфична и не представляет для нас интереса.

Ниже приведена полученная схема, соответствующая верхней картинке, за исключением некоторых полей:



Создайте такую-же схему с полным набором полей.

Посмотрите какой код на языке SQL генерирует SQL Data Modeler. (третья справа иконка в меню)

Добейтесь, чтобы все первичные ключи автоматически генерировались при помощи последовательностей и триггеров, как на примере внизу:

CREATE TABLE Заказ (

КодЗаказа NUMBER NOT NULL ,

КлиентЗк NUMBER NOT NULL ,

Дата\_Заказа DATE ,

ГородВылета NUMBER NOT NULL ,

Дата\_вылета DATE ,

ГородПрилета NUMBER NOT NULL ) ;

ALTER TABLE Заказ ADD CONSTRAINT Заказ\_PK PRIMARY KEY ( КодЗаказа ) ;

CREATE SEQUENCE Зк\_КодЗаказа\_SEQ START WITH 1 NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER Зк\_КодЗаказа\_TRG BEFORE

INSERT ON Заказ FOR EACH ROW

BEGIN

:NEW.КодЗаказа := Зк\_КодЗаказа\_SEQ.NEXTVAL;

END;

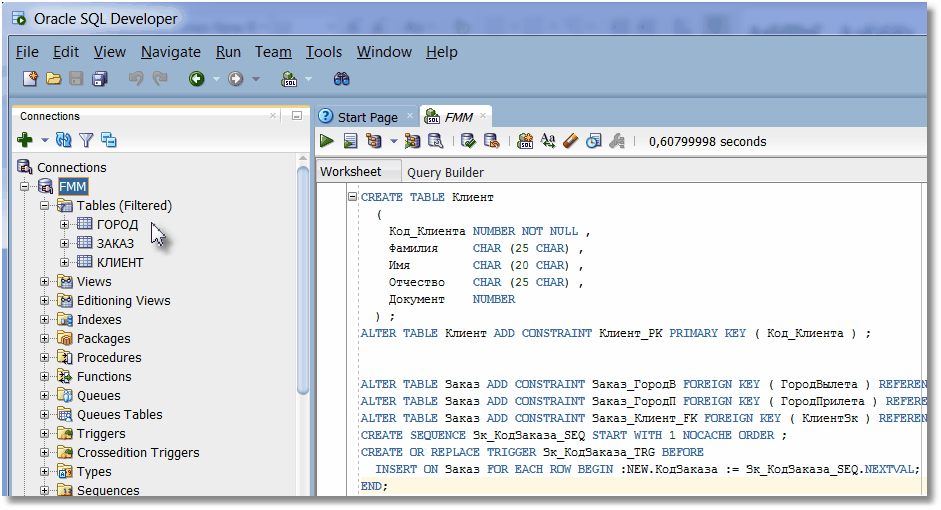
Посмотрите, как определяются вторичные ключи.

Подумайте, какие еще таблицы нужны для реализации проекта выбора маршрута перелета.

Спроектируйте полную схему и заведите ее в SQL Data Modeler.

# Перенос схемы БД из SQL Data Modeler в SQL Developer

Получите схему базы данных в виде SQL кода и выполните его в схеме пользователя, созданного в п 3.1. Исправьте ошибки SQL если таковые обнаружатся.



# Добавление данных в таблицы

Добавьте данные в полученные таблицы – «Город»- 5шт., «Аэропорт» 15 шт., «Клиент» – 1 шт. «Заказ» – 1шт. (Это минимум, большее не возбраняется :о). Попробуйте добавить записи из интерфейса, попробуйте добавить из таблицы EXCEL, попробуйте из ACCESS.

Добавление строк в таблицу «Рейсы» будем делать при помощи SQL.

Необходимо написать и отладить оператор INSERT, который создаст записи в таблице рейсы. Из каждого аэропорта в каждый аэропорт должны отправляться ежедневные рейсы всех авиаперевозчиков. Для простоты предполагаем, что вылет происходит с темпом 1 минута, в полете все рейсы проводят 4 часа и приземляются в Тпос = Твз + 4часа. Создайте последовательность для генерации времени вылета. Между аэропортами одного города самолеты не летают, Международный код рейса состоит из кода перевозчика и, собственно, номера рейса (4 цифры), например - SU1134, международный код рейса уникален. Напишите сначала SELECT, который будет выдавать все возможные рейсы, чтобы это получилось надо соединить таблицы Перевозчики, Аэропорты вылета и Аэропорты прилета как указано в п. 2.8., при этом предложение FROM получится примерно таким:

FROM ПЕРЕВОЗЧИКИ ПР, АЭРОПОРТ АЭВЫЛ, АЭРОПОРТ АЭПРИЛ

**Вариант №1**

Создайте последовательность для генерации уникального кода рейса и при помощи него вставьте записи в таблицу «Рейсы».

**Вариант№2**

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.

# Представления

Создайте представление: «Все рейсы из города вылета». Город вылета берется, естественно, из таблицы «Заказ». Попробуйте, как будет вести себя представление в различных запросах (без ограничения заказа или с указанием конкретного заказа)

**Рекомендуемая литература**

1. Дейт К. Введение в системы баз данных, 8-е издание: Пер. с англ. – М.:Издательский дом ”Вильямс”. 2005.-1328 с.
2. Грабер М. Введение в SQL: Пер с англ. – М.:Изд-во’ЛОРИ’,1996. – 380с.
3. Кайт Томас. Oracle для профессионалов: архитектура, методика программирования и основные особенности.: Пер с англ. – М.:Издательский дом ”Вильямс”. 2008. – 848 с.