Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

**ОТЧЕТ по лабораторной работе №**11

по дисциплине «Базы данных»

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Сафонов

подпись

**Проверил:**

Кандидат тех. наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

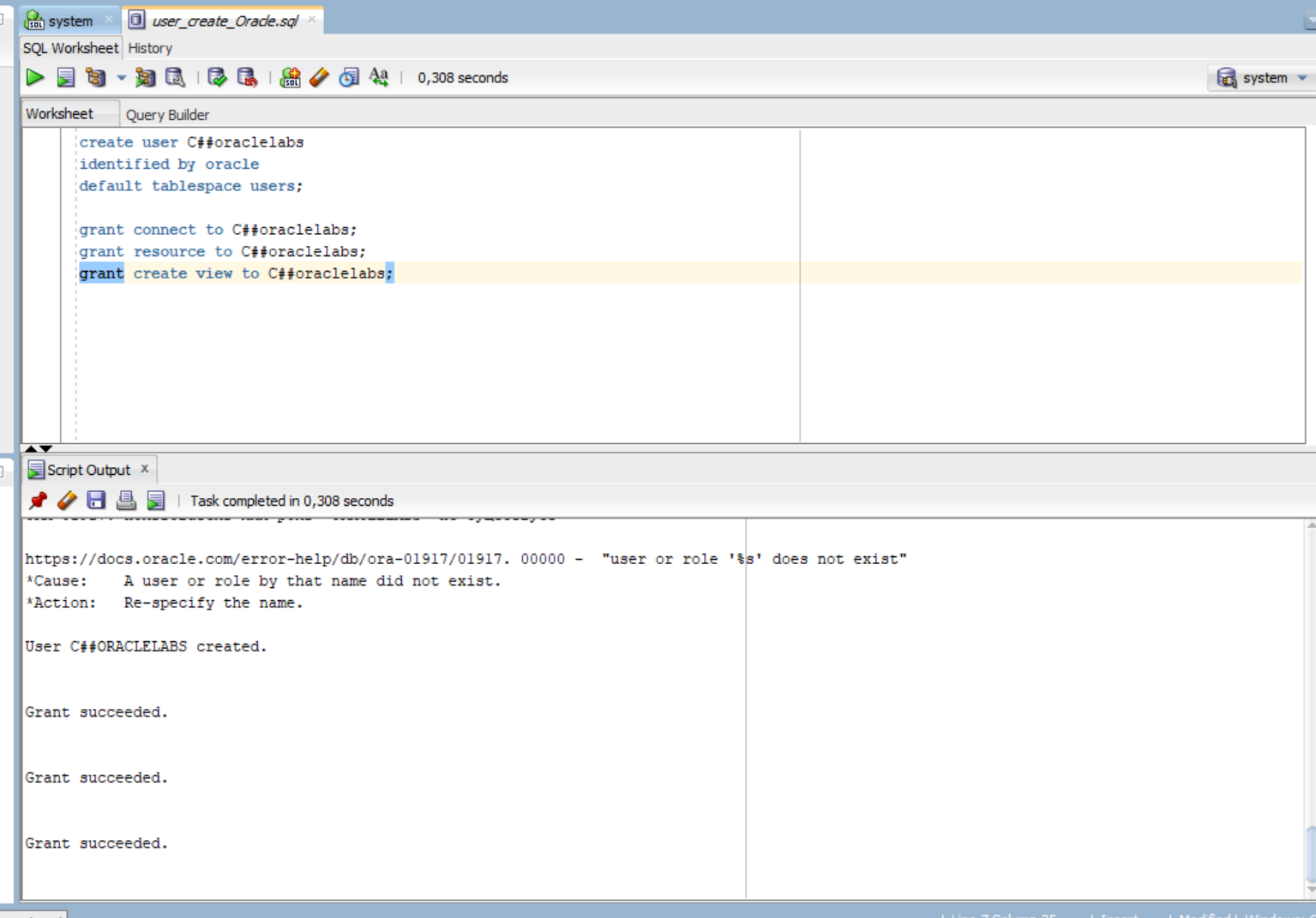
подпись

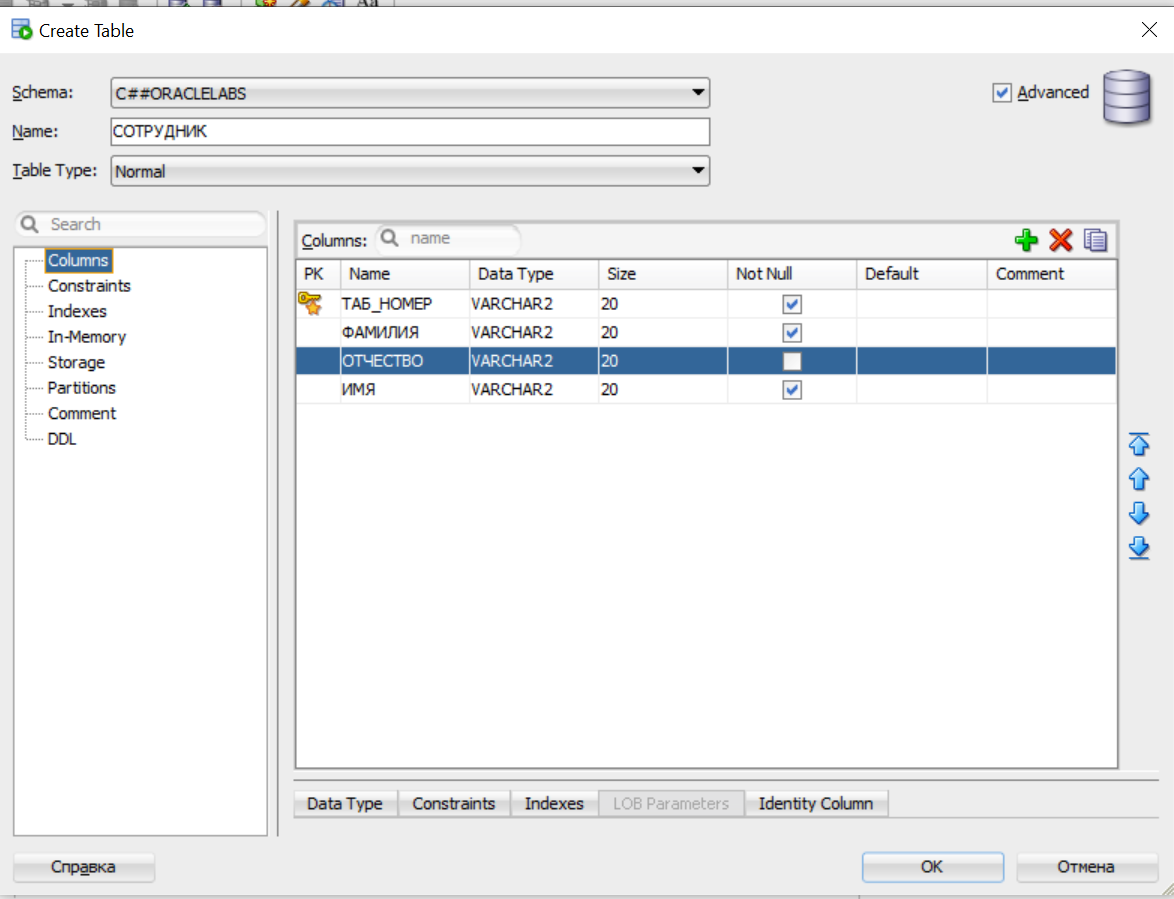
«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

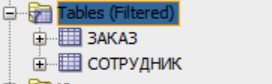
Санкт-Петербург, 2025

# ЛАБОРАТОРНАЯ РАБОТА 9.1 (ORACLE). ПОДКЛЮЧЕНИЕ К СУБД ORACLE. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЯ (СХЕМЫ). РАБОТА С ТАБЛИЦАМИ.

Запустите SQL Developer и посмотрите видео, выложенное на странице курса с описанием данной лабораторной работы – для подключения к базе и создания первых двух таблиц: Сотрудник и Заказ.

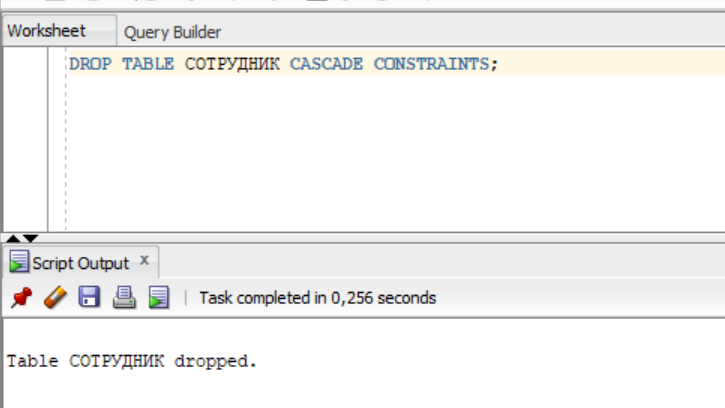


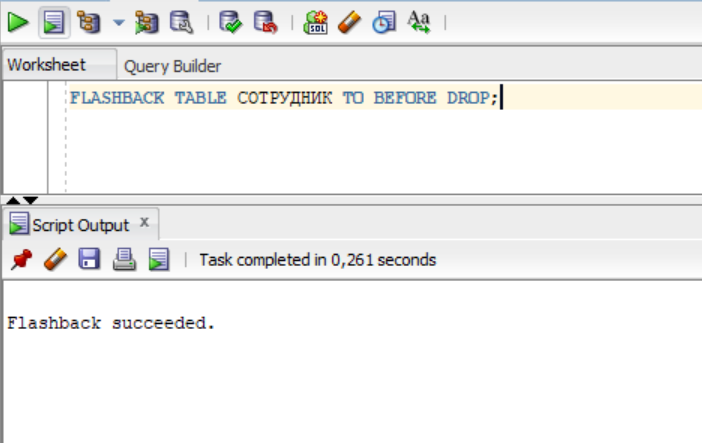


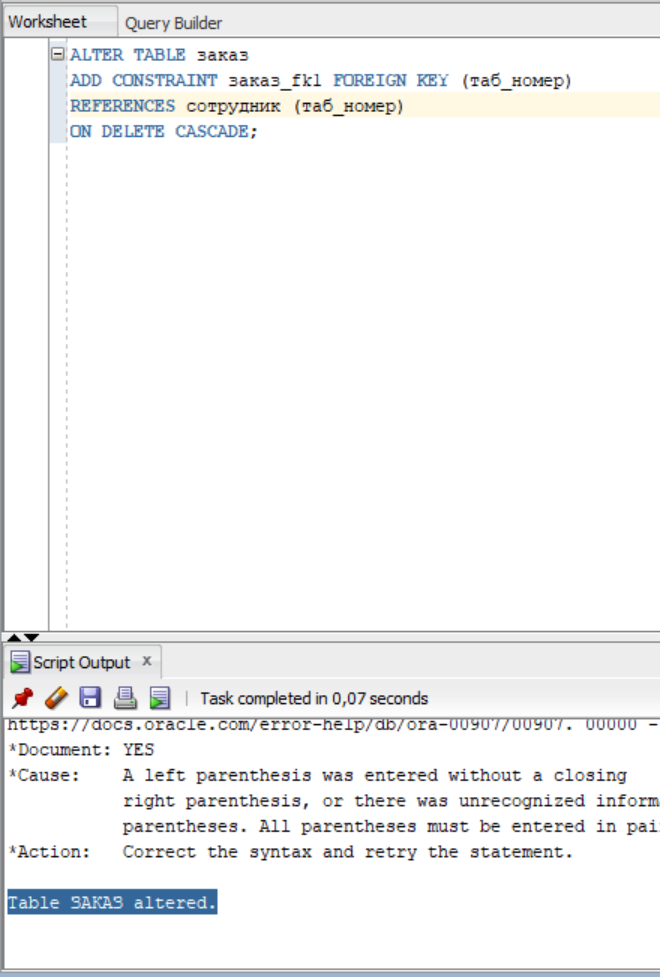


В конце данного видео мы удалили таблицу Сотрудник. По умолчанию команда DROP TABLE в Oracle не приводит к освобождению места в файле на диске, занимаемого данными таблицы, а перемещает её вместе со связанными индексами в так называемую корзину. Описание структуры таблицы при этом удаляется из словаря данных.

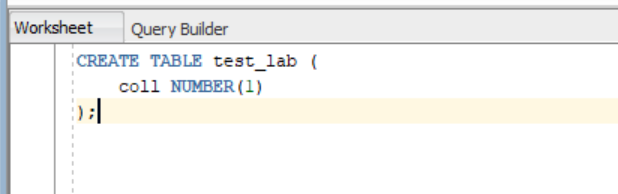
Восстановите таблицу Сотрудник из корзины.

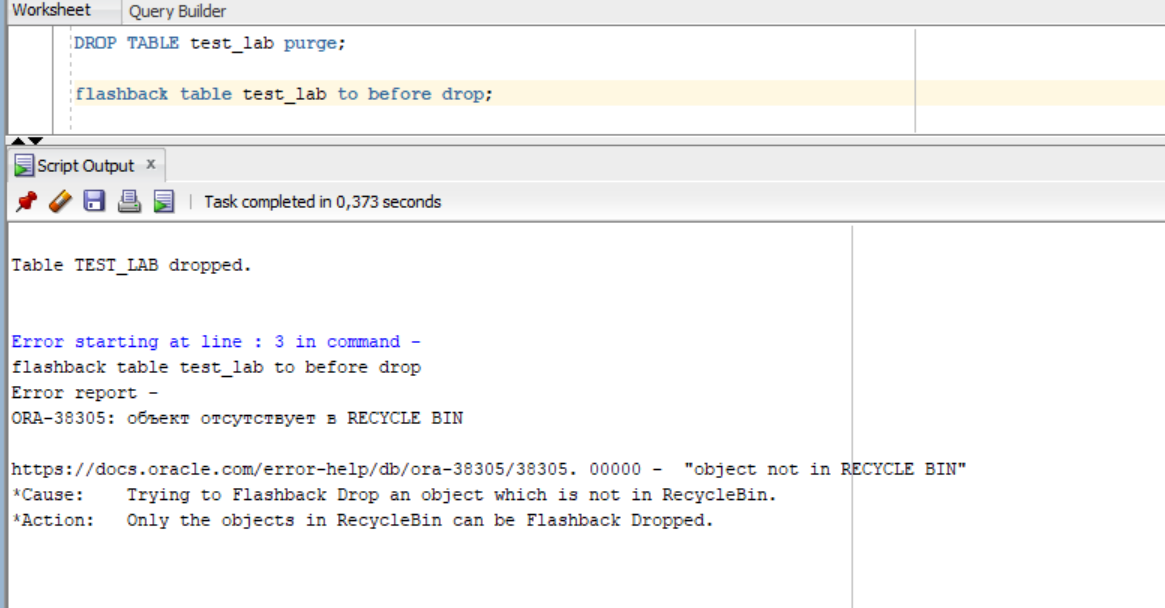




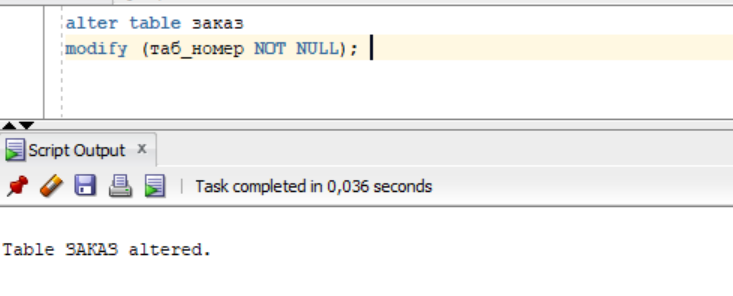


Для полного удаления таблицы, без помещения в корзину, необходимо добавить ключевое слово PURGE к команде DROP TABLE.

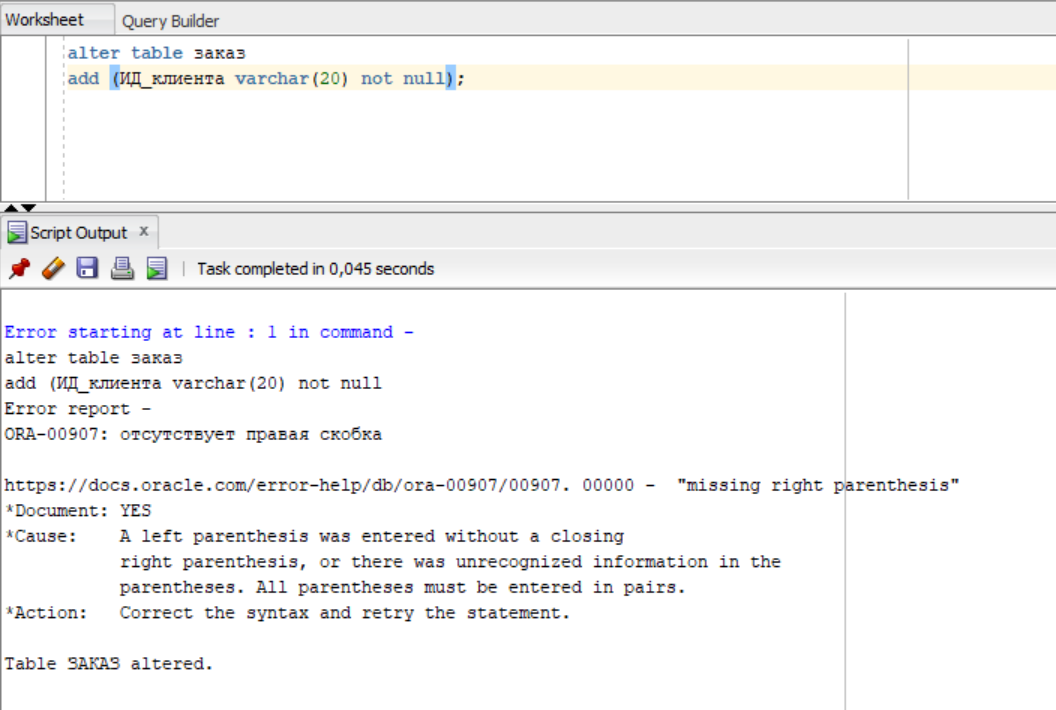




Добавьте ограничение NOT NULL для столбца таб\_номер таблицы Заказ. Определение столбца в данном случае должно включать имя столбца и изменяемые или добавляемые характеристики. Т.е. если не требуется изменять, например, тип данных, то не нужно его заново прописывать в команде.

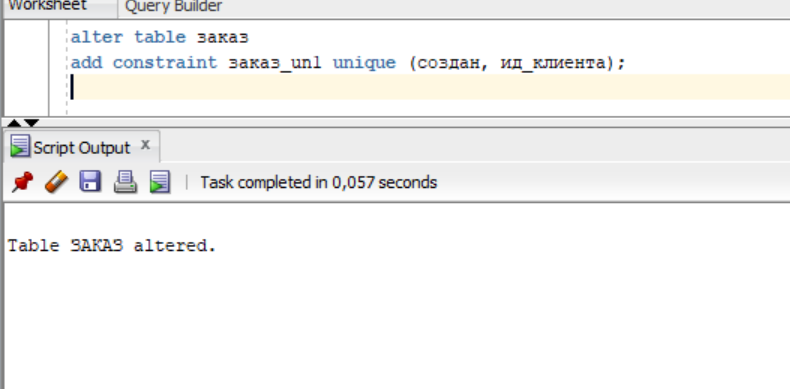


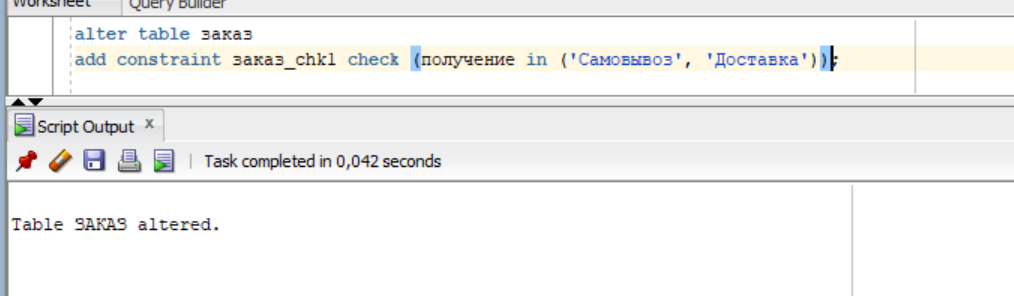
В соответствии с описанием таблиц из задания третьей недели добавьте недостающую колонку в таблицу Заказ и пересоздайте (удалите и создайте заново) имеющееся ограничение уникальности с её учётом. Создайте остальные таблицы и связи между ними.

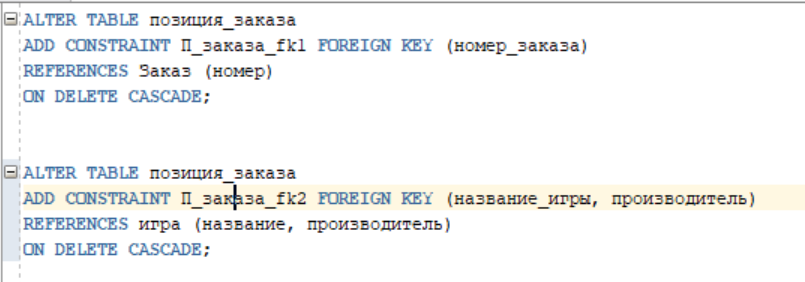


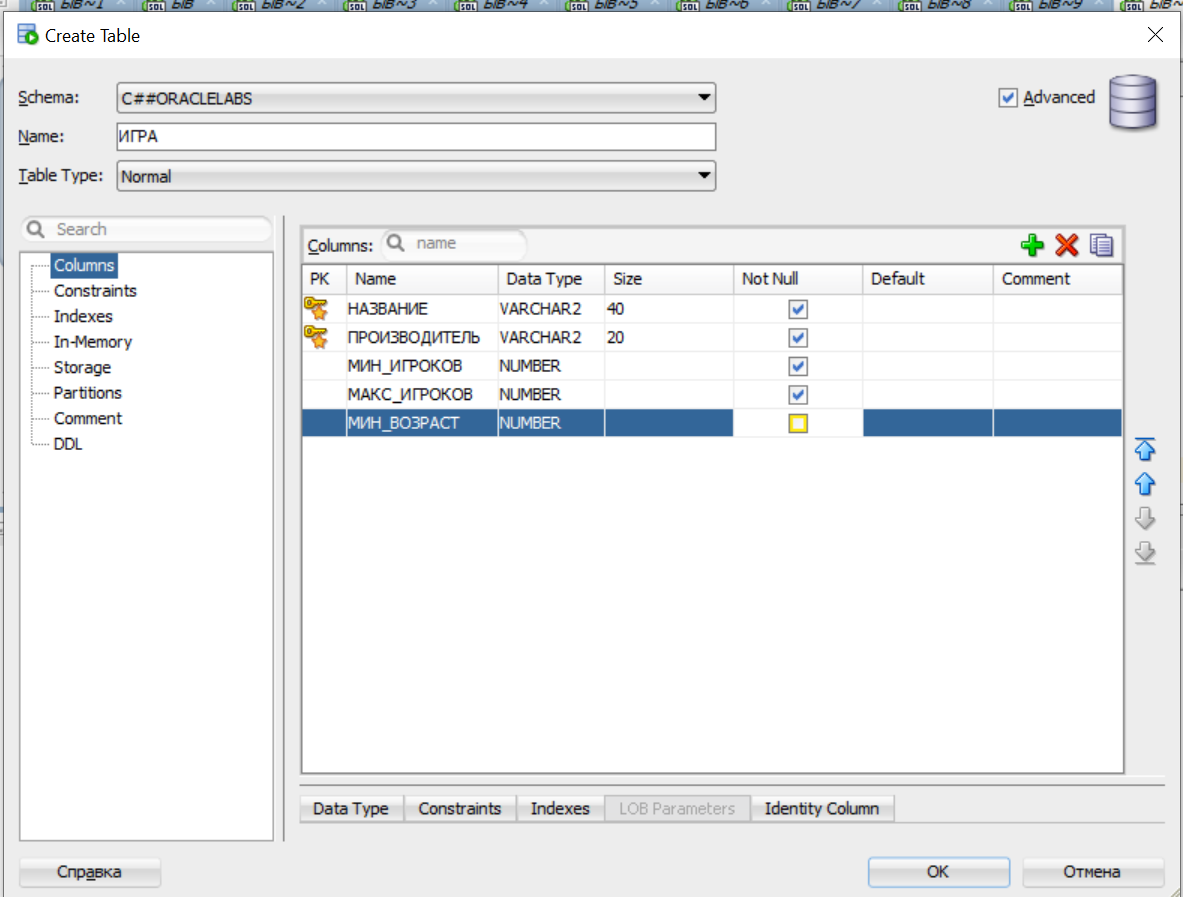
Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание





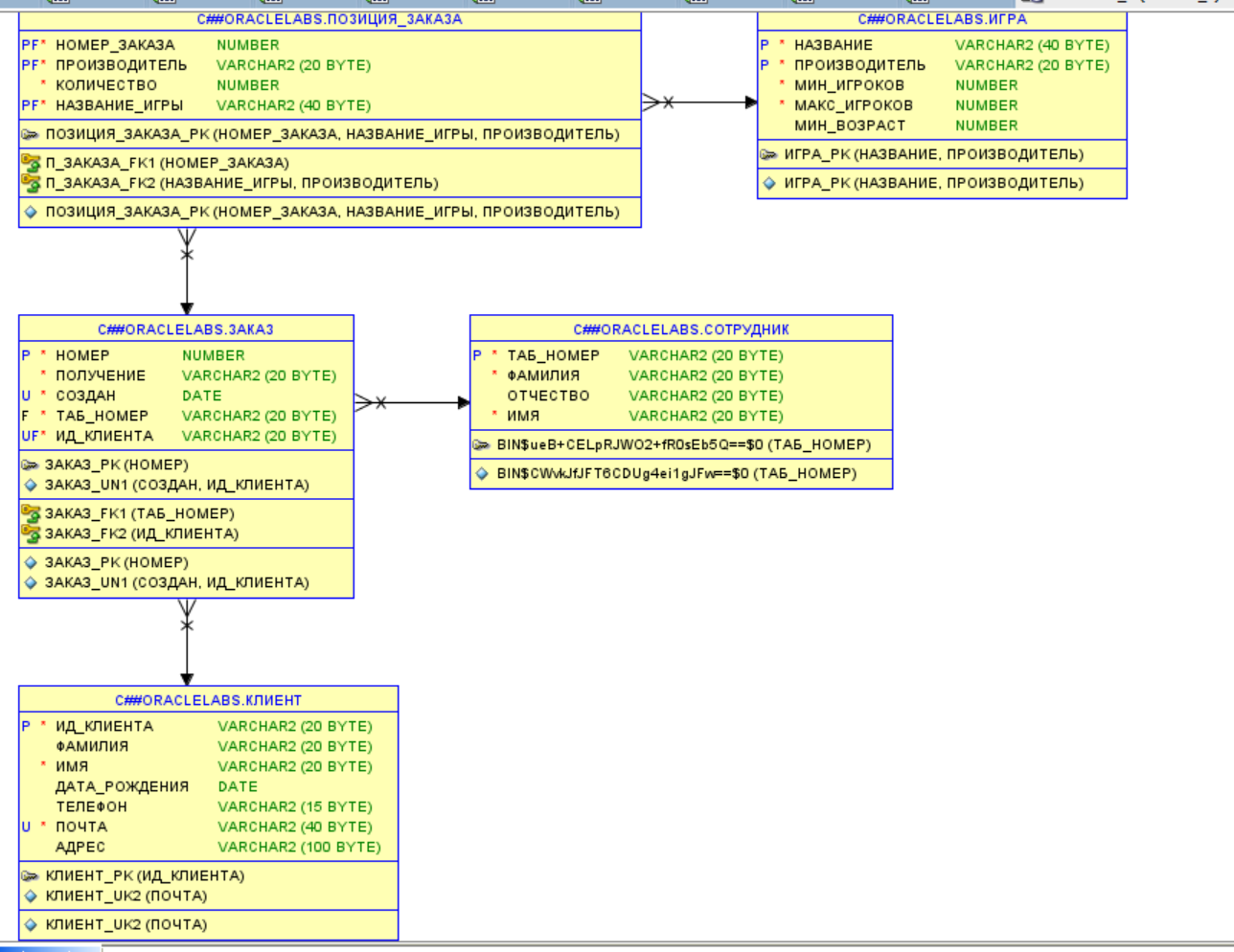




Для самопроверки отобразите реляционную модель, соответствующую созданным таблицам, с помощью встроенного средства проектирования баз данных Oracle Data Modeler

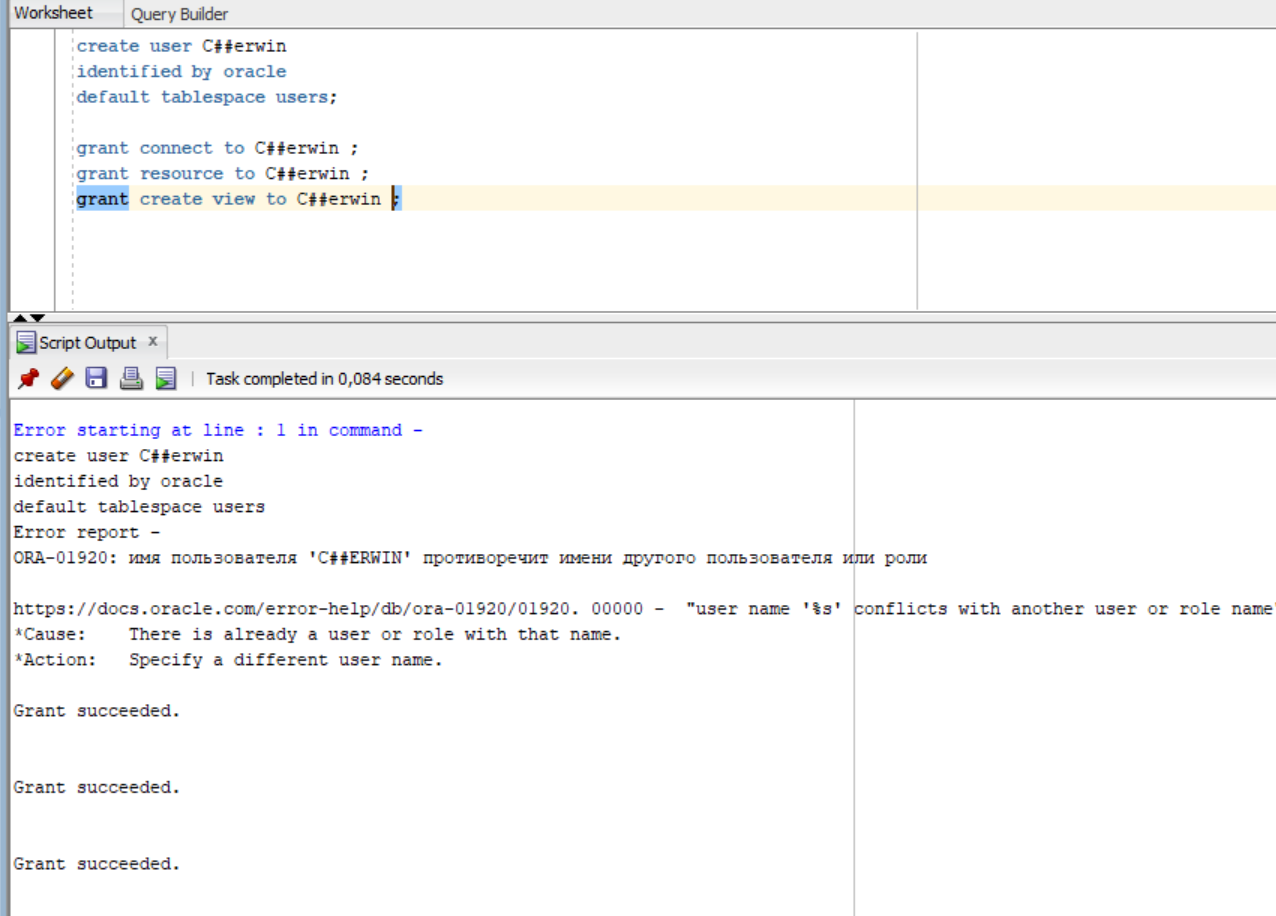
Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание



# ЛАБОРАТОРНАЯ РАБОТА №1 (ORACLE) ДЕВЯТОЙ НЕДЕЛИ КУРСА «УПРАВЛЕНИЕ ДАННЫМИ». СОЗДАНИЕ БАЗЫ ДАННЫХ НА ОСНОВЕ МОДЕЛИ ERWIN

Из-под учётной записи пользователя system создайте нового пользователя erwin с паролем erwin. Воспользуйтесь для этого соответствующим скриптом, выложенным в разделе с описанием лабораторной №1 (Oracle). Не забудьте заменить oraclelabs на erwin во всех командах скрипта.



-- -----------------------------------------------------

-- Table Процессор

-- -----------------------------------------------------

CREATE TABLE Процессор (

ProcID INT NOT NULL,

ProcFullName VARCHAR2(45) NULL,

ProcVendor VARCHAR2(45) NULL,

ProcArch VARCHAR2(45) NULL,

ProcFreq INT NULL,

PRIMARY KEY (ProcID)

);

-- -----------------------------------------------------

-- Table Роль\_компьютера

-- -----------------------------------------------------

CREATE TABLE Роль\_компьютера (

CompRoleID INT NOT NULL,

CompRoleFullName VARCHAR2(45) NULL,

PRIMARY KEY (CompRoleID)

);

-- -----------------------------------------------------

-- Table Тип\_RAM

-- -----------------------------------------------------

CREATE TABLE Тип\_RAM (

RamID INT NOT NULL,

RamType VARCHAR2(45) NULL,

RamName VARCHAR2(45) NULL,

PRIMARY KEY (RamID)

);

-- -----------------------------------------------------

-- Table Компьютер

-- -----------------------------------------------------

CREATE TABLE Компьютер (

CompID INT NOT NULL,

CompName VARCHAR2(45) NULL,

ProcNumber VARCHAR2(45) NULL,

RamSizeGB INT NULL,

CompRoleID INT NULL,

ProcID INT NOT NULL,

RamID INT NOT NULL,

PRIMARY KEY (CompID),

CONSTRAINT fk\_Компьютер\_Роль\_компьютера1

FOREIGN KEY (CompRoleID)

REFERENCES Роль\_компьютера (CompRoleID)

ON DELETE SET NULL,

CONSTRAINT fk\_Компьютер\_Процессор1

FOREIGN KEY (ProcID)

REFERENCES Процессор (ProcID),

CONSTRAINT fk\_Компьютер\_Тип\_RAM1

FOREIGN KEY (RamID)

REFERENCES Тип\_RAM (RamID)

);

CREATE INDEX fk\_Компьютер\_Роль\_компьютера1\_idx ON Компьютер (CompRoleID);

CREATE INDEX fk\_Компьютер\_Процессор1\_idx ON Компьютер (ProcID);

CREATE INDEX fk\_Компьютер\_Тип\_RAM1\_idx ON Компьютер (RamID);

-- -----------------------------------------------------

-- Table Жесткий\_диск

-- -----------------------------------------------------

CREATE TABLE Жесткий\_диск (

HDDModelID INT NOT NULL,

HDDModelFullName VARCHAR2(45) NULL,

HDDVendo VARCHAR2(45) NULL,

HDDSizeGB INT NULL,

HDDInterface VARCHAR2(45) NULL,

PRIMARY KEY (HDDModelID)

);

-- -----------------------------------------------------

-- Table ОС

-- -----------------------------------------------------

CREATE TABLE ОС (

OSID INT NOT NULL,

OSName VARCHAR2(45) NULL,

OSVendor VARCHAR2(45) NULL,

OSInfo VARCHAR2(45) NULL,

PRIMARY KEY (OSID)

);

-- -----------------------------------------------------

-- Table Пользователь

-- -----------------------------------------------------

CREATE TABLE Пользователь (

UserID INT NOT NULL,

UserFIrstName VARCHAR2(45) NULL,

UserSecondName VARCHAR2(45) NULL,

UserLastName VARCHAR2(45) NULL,

UserLogin VARCHAR2(45) NOT NULL,

PRIMARY KEY (UserID)

);

-- -----------------------------------------------------

-- Table СетевоеУ

-- -----------------------------------------------------

CREATE TABLE СетевоеУ (

NetDeviceID INT NOT NULL,

NetDeviceName VARCHAR2(45) NULL,

NetDeviceType VARCHAR2(45) NULL,

NetDeviceModel VARCHAR2(45) NULL,

NetDeviceVendor VARCHAR2(45) NULL,

NetDevicePortNumber INT NULL,

PRIMARY KEY (NetDeviceID)

);

-- -----------------------------------------------------

-- Table Сетевой\_интерфейс

-- -----------------------------------------------------

CREATE TABLE Сетевой\_интерфейс (

NetInterfaceID INT NOT NULL,

NetInterfaceMAC VARCHAR2(45) NULL,

NetInterfaceIP VARCHAR2(45) NULL,

NetInterfaceInfo VARCHAR2(45) NULL,

CompID INT NOT NULL,

NetDeviceID INT NOT NULL,

PRIMARY KEY (NetInterfaceID, CompID),

CONSTRAINT fk\_Сетевой\_интерфейс\_Компьютер1

FOREIGN KEY (CompID)

REFERENCES Компьютер (CompID)

ON DELETE CASCADE,

CONSTRAINT fk\_Сетевой\_интерфейс\_СетевоеУ1

FOREIGN KEY (NetDeviceID)

REFERENCES СетевоеУ (NetDeviceID)

);

CREATE INDEX fk\_Сетевой\_интерфейс\_Компьютер1\_idx ON Сетевой\_интерфейс (CompID);

CREATE INDEX fk\_Сетевой\_интерфейс\_СетевоеУ1\_idx ON Сетевой\_интерфейс (NetDeviceID);

-- -----------------------------------------------------

-- Table Пользователь\_has\_Компьютер

-- -----------------------------------------------------

CREATE TABLE Пользователь\_has\_Компьютер (

UserID INT NOT NULL,

CompID INT NOT NULL,

PRIMARY KEY (UserID, CompID),

CONSTRAINT fk\_Пользователь\_has\_Компьютер\_Польз1

FOREIGN KEY (UserID)

REFERENCES Пользователь (UserID),

CONSTRAINT fk\_Пользователь\_has\_Компьютер\_Компь1

FOREIGN KEY (CompID)

REFERENCES Компьютер (CompID)

);

CREATE INDEX fk\_Пользователь\_has\_Компьютер\_Комп\_idx ON Пользователь\_has\_Компьютер (CompID);

CREATE INDEX fk\_Пользователь\_has\_Компьютер\_Поль\_idx ON Пользователь\_has\_Компьютер (UserID);

-- -----------------------------------------------------

-- Table ОС\_has\_Компьютер

-- -----------------------------------------------------

CREATE TABLE ОС\_has\_Компьютер (

OSID INT NOT NULL,

CompID INT NOT NULL,

PRIMARY KEY (OSID, CompID),

CONSTRAINT fk\_ОС\_has\_Компьютер\_ОС1

FOREIGN KEY (OSID)

REFERENCES ОС (OSID),

CONSTRAINT fk\_ОС\_has\_Компьютер\_Компьютер1

FOREIGN KEY (CompID)

REFERENCES Компьютер (CompID)

);

CREATE INDEX fk\_ОС\_has\_Компьютер\_Компьютер1\_idx ON ОС\_has\_Компьютер (CompID);

CREATE INDEX fk\_ОС\_has\_Компьютер\_ОС1\_idx ON ОС\_has\_Компьютер (OSID);

-- -----------------------------------------------------

-- Table Компьютер\_has\_Жесткий\_диск

-- -----------------------------------------------------

CREATE TABLE Компьютер\_has\_Жесткий\_диск (

CompID INT NOT NULL,

HDDModelID INT NOT NULL,

PRIMARY KEY (CompID, HDDModelID),

CONSTRAINT fk\_Компьютер\_has\_Компьютер\_Компьюте1

FOREIGN KEY (CompID)

REFERENCES Компьютер (CompID),

CONSTRAINT fk\_Компьютер\_has\_Компьютер\_Жесткий1

FOREIGN KEY (HDDModelID)

REFERENCES Жесткий\_диск (HDDModelID)

);

CREATE INDEX fk\_Компьютер\_has\_Компьютер\_Жесткий\_idx ON Компьютер\_has\_Жесткий\_диск (HDDModelID);

-- -----------------------------------------------------

-- View view1

-- -----------------------------------------------------

CREATE OR REPLACE VIEW view1 AS

SELECT

c.CompName AS "Название компьютера",

u.UserLogin AS "Имя учетной записи"

FROM

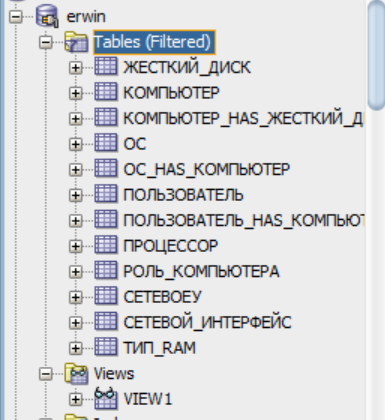
Компьютер c

JOIN

Пользователь\_has\_Компьютер uc ON c.CompID = uc.CompID

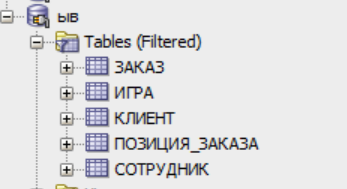
JOIN

Пользователь u ON uc.UserID = u.UserID;

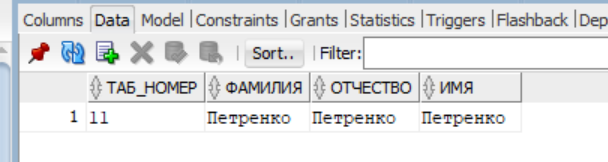


# ЛАБОРАТОРНАЯ РАБОТА (ORACLE) ДЕСЯТОЙ НЕДЕЛИ КУРСА «УПРАВЛЕНИЕ ДАННЫМИ». ДОБАВЛЕНИЕ, ОБНОВЛЕНИЕ, УДАЛЕНИЕ ДАННЫХ

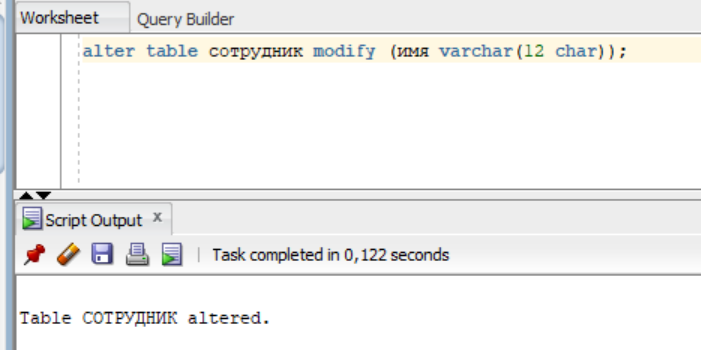
В SQL Developer откройте новое соединение для пользователя oraclelabs, созданного в ходе выполнения лабораторных работ прошлой недели.



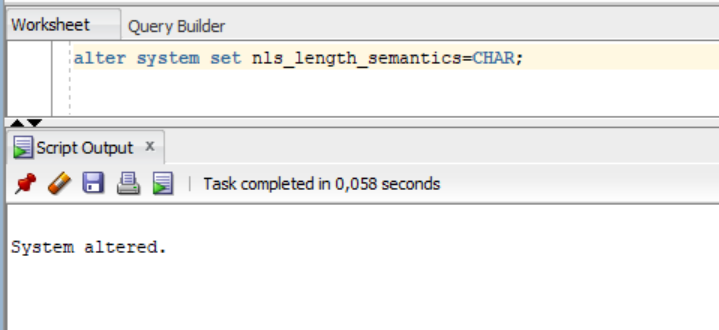
Добавьте информацию о сотруднике Петренко в таблицу СОТРУДНИК, используя графический интерфейс SQL Developer

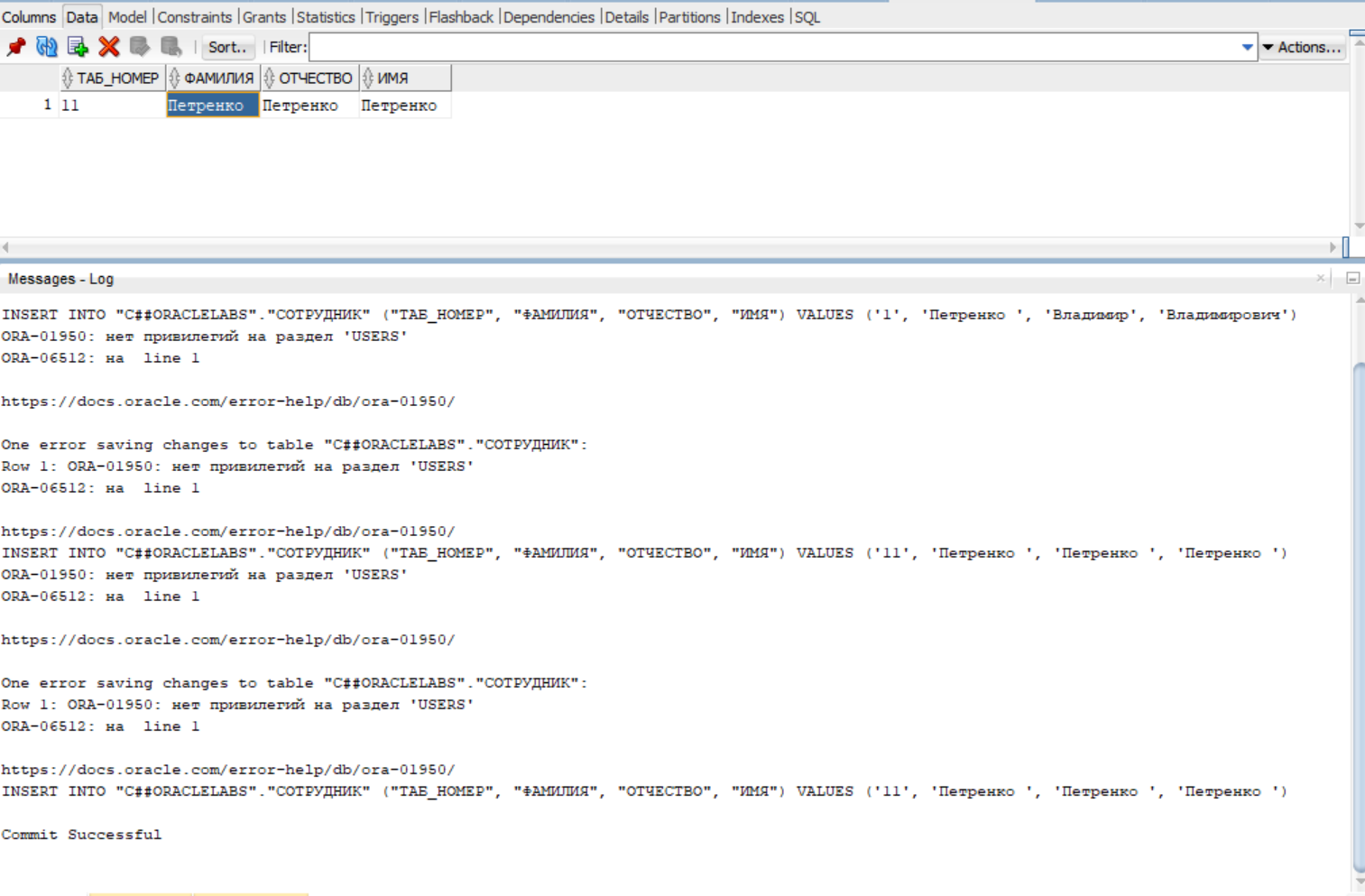


отредактировать таблицу СОТРУДНИК, увеличив длину столбца Имя до 24 или оставив 12, но явно прописав, что мы хотим иметь возможность добавлять данные длиной до 12 символов (CHAR), а не байт

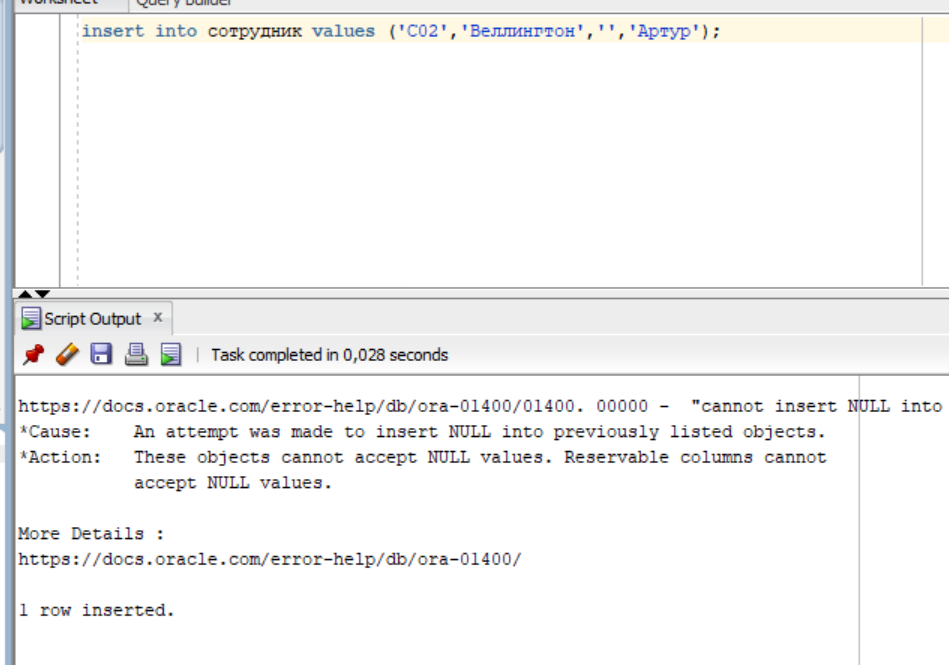


от имени пользователя system выполнить команду для изменения семантики хранения символьных значений на уровне базы данных

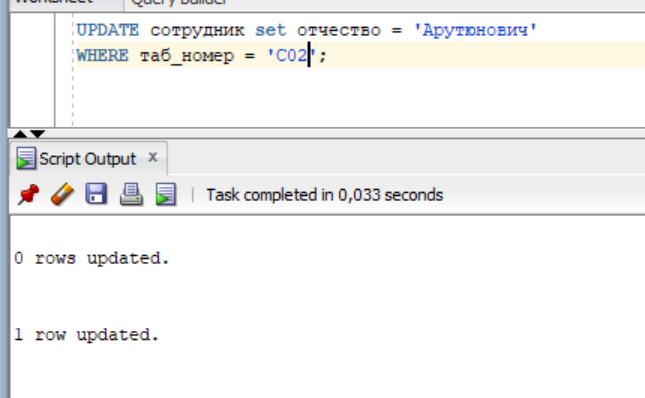


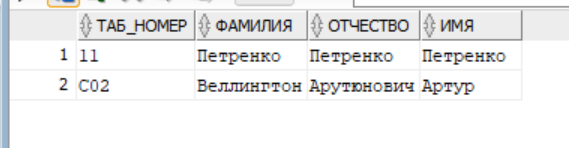


Добавим информацию о втором сотруднике с помощью SQL-команды INSERT без явного указания столбцов таблицы.

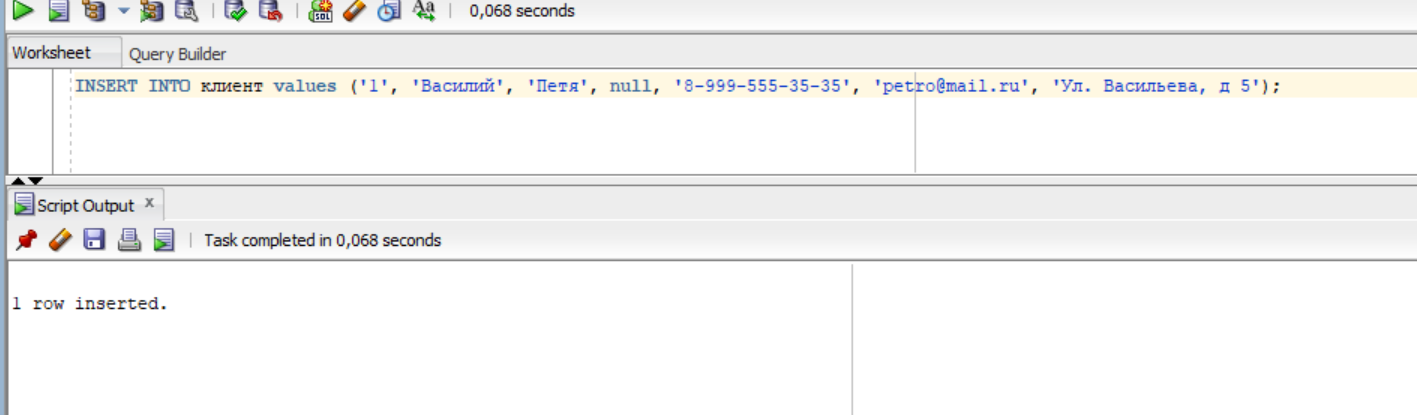


С помощью команды UPDATE измените неизвестное значение отчества для второго сотрудника на 'Арутюнович'. Воспользуйтесь разделом WHERE и значением первичного ключа, чтобы явно указать строку, в которой необходимо произвести изменения

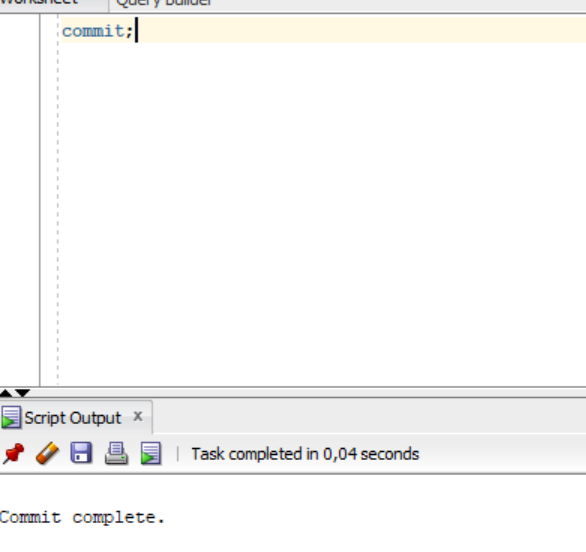




Добавьте первого клиента в таблицу Клиент.



Сохраните все внесённые до этого момента изменения с помощью команды COMMIT



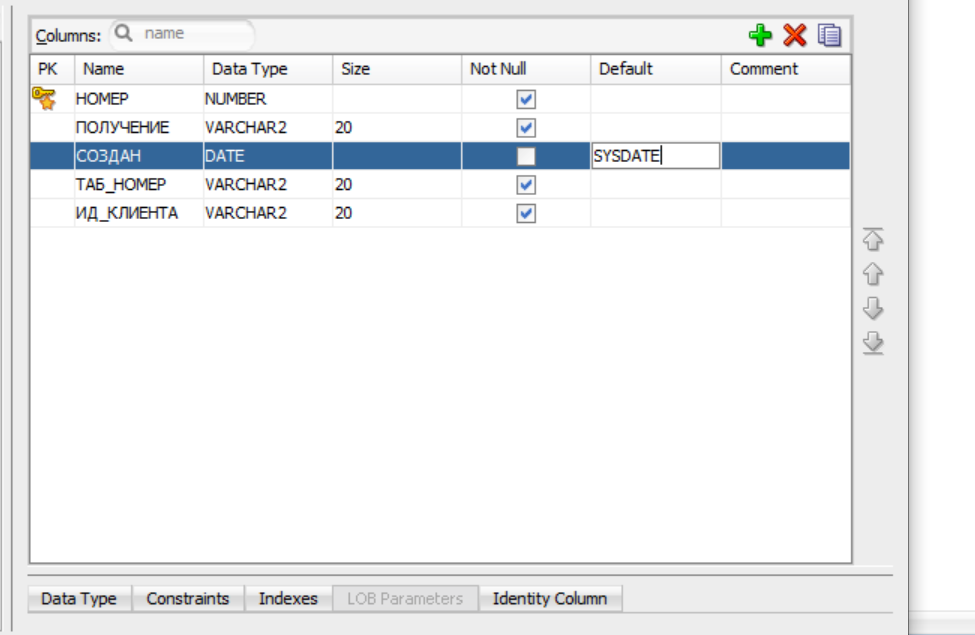
Добавим теперь информацию о первом заказе с помощью SQL-команды INSERT с явным перечислением столбцов таблицы

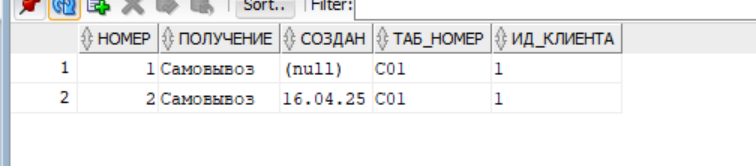
Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

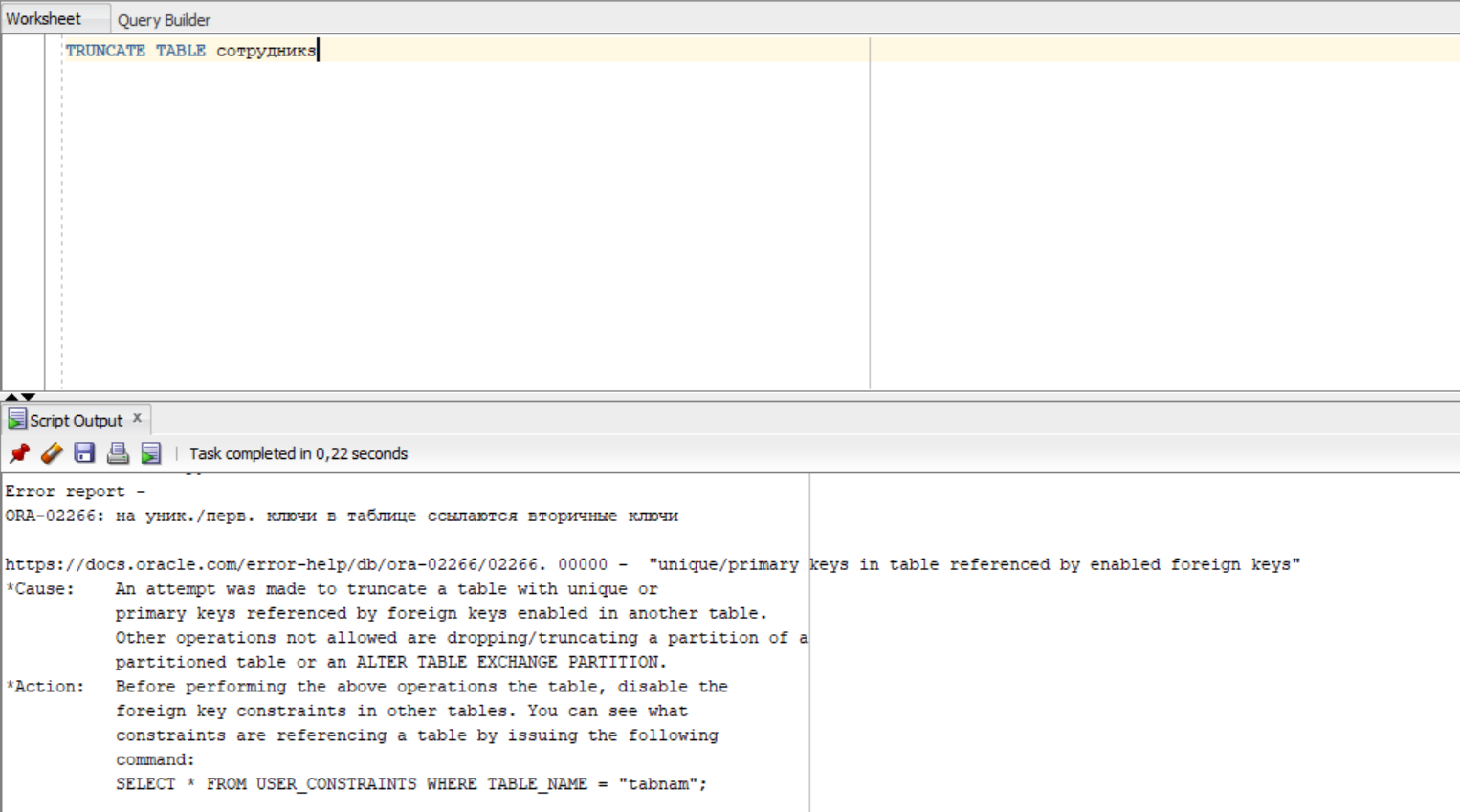
Ошибка ORA-12899 возникает из-за попытки вставить значение, которое превышает максимально допустимый размер столбца в таблице.

Допустим, мы хотим, чтобы при добавлении информации о новом заказе в базу в столбец Создан для него автоматически вставлялось текущее значение системной даты. Измените таблицу ЗАКАЗ, указав в качестве стандартного значения для столбца Создан функцию SYSDATE.

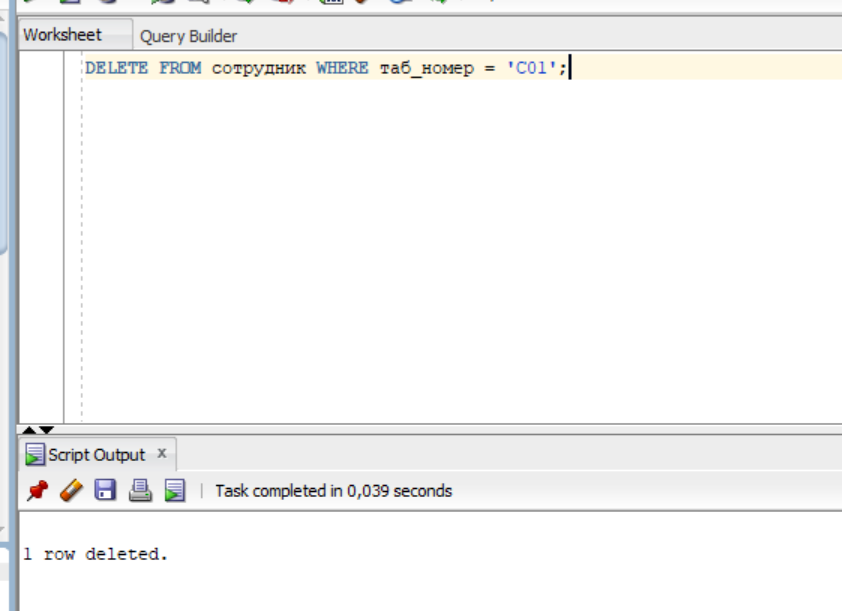




Попробуйте полностью очистить таблицу СОТРУДНИК с помощью команды TRUNCATE.



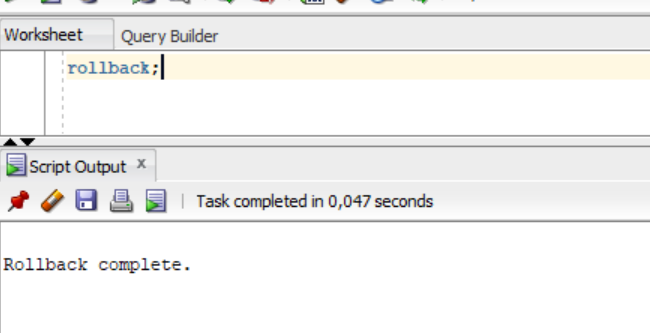
Убедитесь, что правило поддержки ссылочной целостности, определённое для внешнего ключа дочерней таблицы ЗАКАЗ работает, удалив сотрудника с табельным номером С01 с помощью команды DELETE. Воспользуйтесь разделом WHERE и значением первичного ключа, чтобы явно указать строку, в которой необходимо произвести изменения. Посмотрите содержимое таблицы ЗАКАЗ и убедитесь, что из неё одновременно была удалена информация о первом заказе.

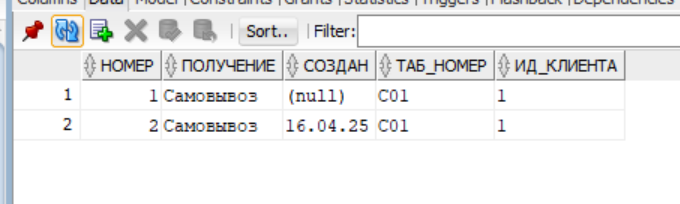


Изображение выглядит как текст, программное обеспечение, Значок на компьютере, снимок экрана

Автоматически созданное описание

Откатите выполненные изменения, выполнив команду ROLLBACK. Убедитесь, что данные, удалённые из таблиц СОТРУДНИК и ЗАКАЗ в п. 10, снова на месте





# Из пособия Романа Симакова

Работа с таблицами в реляционных базах данных: полное руководство

Введение в таблицы реляционных баз данных

Таблицы представляют собой фундаментальную структуру организации данных в реляционных базах данных. Они состоят из столбцов (полей), определяющих структуру данных, и строк (записей), содержащих непосредственно информацию. Каждая таблица обладает уникальным именем в пределах базы данных и должна содержать как минимум один столбец. Количество строк в таблице теоретически не ограничено, при этом все строки имеют идентичную структуру.

Создание и определение таблиц

Основой для создания таблиц служит оператор CREATE TABLE, который включает уникальное имя таблицы, список определений столбцов и необязательные ограничения таблицы. При создании таблицы employees, например, мы определяем столбцы для идентификатора сотрудника, его имени, даты приема на работу и зарплаты.

При определении столбцов необходимо указать уникальное имя столбца в пределах таблицы, тип данных или имя домена, а также дополнительные параметры: значение по умолчанию (DEFAULT), запрет на NULL-значения (NOT NULL) и различные ограничения (PRIMARY KEY, UNIQUE и другие).

Типы столбцов и их особенности

В реляционных базах данных существуют три основных типа столбцов. Обычные столбцы хранят непосредственные данные, такие как идентификатор продукта и его цена. Вычисляемые столбцы не хранят данные, а рассчитывают их при выполнении запроса, например, общую стоимость заказа как произведение количества на цену. Столбцы идентификации поддерживают автоинкремент и часто используются для первичных ключей.

Система ограничений таблиц

Реляционные базы данных предоставляют мощную систему ограничений для обеспечения целостности данных. Первичный ключ (PRIMARY KEY) гарантирует уникальность идентификатора в таблице. Ограничение уникальности (UNIQUE) обеспечивает неповторяемость значений в столбце. Внешний ключ (FOREIGN KEY) устанавливает связи между таблицами, а проверочное ограничение (CHECK) задает условия для значений в столбце, например, что возраст сотрудника должен быть не менее 18 лет.

Модификация структуры таблиц

Для изменения существующих таблиц используется оператор ALTER TABLE. С его помощью можно добавлять новые столбцы, удалять существующие, изменять типы данных столбцов и добавлять ограничения к уже созданной таблице. Эти операции позволяют гибко адаптировать структуру базы данных к изменяющимся требованиям без необходимости ее полного пересоздания.

Удаление таблиц и связанные операции

Удаление таблиц выполняется командой DROP TABLE. Базовый вариант команды просто удаляет таблицу, а с использованием ключевого слова CASCADE система автоматически удаляет все объекты, зависящие от этой таблицы. Это мощный инструмент, который следует использовать с осторожностью, чтобы случайно не удалить важные данные.

Практические рекомендации по проектированию

При работе с таблицами рекомендуется использовать осмысленные и последовательные имена для таблиц и столбцов. Для первичных ключей лучше выбирать целочисленные типы данных. Важно применять соответствующие ограничения, такие как NOT NULL, для обеспечения целостности данных. Для текстовых данных с кириллицей следует указывать VARCHAR(N CHAR), чтобы избежать проблем с кодировкой. Не менее важно документировать структуру базы данных для последующего сопровождения.

Комплексный пример проектирования

Рассмотрим пример создания таблицы library\_books для учета книг в библиотеке. Таблица включает уникальный идентификатор книги, ее название, автора, ISBN-код, год публикации, цену, признак доступности и дату последнего взятия. Для каждого столбца заданы соответствующие типы данных и ограничения: первичный ключ для идентификатора, обязательные поля для названия и автора, уникальность для ISBN-кода, проверка года публикации и цены. Такой подход обеспечивает надежное хранение и целостность данных.

Это руководство предоставляет полный обзор работы с таблицами в реляционных базах данных. Для эффективного использования материала рекомендуется структурировать документ с помощью заголовков, добавить оглавление, включать практические примеры и отмечать распространенные ошибки. Такой подход поможет как начинающим, так и опытным разработчикам в работе с базами данных.

Транзакции в базах данных: полное руководство

Понятие транзакции и ее свойства

Транзакции представляют собой фундаментальную концепцию баз данных, обеспечивающую целостность данных. Транзакция - это последовательность операций с базой данных, которая должна быть выполнена как единое целое по принципу "все или ничего".

Стандартно выделяют четыре базовых свойства транзакций, известных как ACID-свойства:

1. Атомарность (Atomicity) - гарантирует, что транзакция будет выполнена полностью или не будет выполнена вообще

2. Согласованность (Consistency) - обеспечивает переход базы данных из одного согласованного состояния в другое

3. Изолированность (Isolation) - предотвращает взаимное влияние параллельных транзакций

4. Долговечность (Durability) - гарантирует сохранение результатов завершенной транзакции даже после сбоев

Практический пример банковской транзакции

Рассмотрим пример банковской системы, где основной операцией является перевод денег между счетами. Эта операция включает два действия: уменьшение баланса одного счета и увеличение баланса другого счета на ту же сумму.

В SQL это реализуется двумя запросами UPDATE. При этом важно обеспечить, чтобы либо оба запроса выполнились успешно, либо ни один из них не был выполнен. В случае сбоя (например, отключения электричества) система должна автоматически откатить все изменения, сделанные в рамках незавершенной транзакции.

Аномалии параллельного доступа

При параллельном выполнении транзакций могут возникать следующие проблемы:

1. Потерянное обновление - возникает при одновременном изменении одних и тех же данных разными транзакциями, когда теряются все изменения, кроме последних.

2. Грязное чтение - происходит, когда транзакция читает незафиксированные изменения другой транзакции, которые впоследствии могут быть отменены.

3. Неповторяющееся чтение - ситуация, когда повторное чтение одних и тех же данных в рамках одной транзакции дает разные результаты.

4. Фантомное чтение - возникает, когда при повторном чтении диапазона данных появляются новые строки, добавленные другими транзакциями.

Уровни изоляции транзакций

Стандарт SQL определяет четыре основных уровня изоляции:

1. READ UNCOMMITTED (Чтение незафиксированных данных) - самый низкий уровень, допускающий все типы аномалий.

2. READ COMMITTED (Чтение зафиксированных данных) - уровень по умолчанию в большинстве СУБД, предотвращает грязное чтение и потерянные обновления.

3. REPEATABLE READ (Повторяемое чтение) - дополнительно предотвращает неповторяющееся чтение.

4. SERIALIZABLE (Упорядочиваемый) - самый строгий уровень, предотвращающий все аномалии.

Управление транзакциями

Для работы с транзакциями используются следующие операторы:

1. SET TRANSACTION - запускает новую транзакцию с указанными параметрами:

- READ WRITE/READ ONLY - определяет режим доступа

- ISOLATION LEVEL - устанавливает уровень изоляции

2. COMMIT - подтверждает все изменения, сделанные в рамках текущей транзакции.

3. ROLLBACK - отменяет все изменения текущей транзакции.

Транзакции играют критически важную роль в обеспечении целостности данных, особенно в системах с высокой нагрузкой и большим количеством параллельных операций. Правильное использование транзакций и уровней изоляции позволяет создавать надежные и отказоустойчивые приложения для работы с базами данных.

Последовательности и генераторы в базах данных

Последовательности (sequences) и генераторы (generators) представляют собой важные объекты баз данных, предназначенные для генерации уникальных целочисленных значений. Эти механизмы особенно востребованы при работе с искусственными первичными ключами, обеспечивая надежный способ получения уникальных идентификаторов для записей таблиц.

Основная функция последовательностей заключается в генерации 8-байтных целых чисел в диапазоне от -2⁶³ до 2⁶³-1. Важной особенностью является их работа вне контекста транзакций, что исключает возможность блокировок при параллельном доступе и гарантирует уникальность каждого сгенерированного значения независимо от количества одновременно работающих пользователей.

Исторически в таких СУБД как Interbase, Firebird и Red Database изначально использовались генераторы. С развитием стандарта SQL был введен термин "последовательность", хотя внутренняя реализация механизма осталась прежней. В современных версиях оба термина считаются синонимами, и разработчики могут использовать любой из них.

Для создания новой последовательности применяется команда CREATE SEQUENCE, где можно указать начальное значение (параметр START WITH) и шаг приращения (INCREMENT). Если параметры не заданы, по умолчанию используются значения 0 и 1 соответственно. Альтернативный вариант - команда CREATE GENERATOR - сохраняется для обратной совместимости.

Управление существующими последовательностями осуществляется с помощью команды ALTER SEQUENCE, которая позволяет изменить начальное значение, шаг приращения или выполнить перезапуск последовательности. Для простой установки текущего значения можно использовать устаревшую, но поддерживаемую команду SET GENERATOR. Универсальная команда CREATE OR ALTER SEQUENCE позволяет одновременно создавать новую или изменять существующую последовательность.

Для получения следующего значения последовательности рекомендуется использовать современный синтаксис NEXT VALUE FOR. Альтернативный вариант - функция GEN\_ID() - сохраняется для поддержки legacy-кода, но считается устаревшим. Оба метода гарантируют получение уникального числа, но отличаются в деталях реализации изменения текущего значения последовательности.

Особое внимание следует уделять работе с последовательностями при достижении граничных значений. При превышении максимального положительного числа происходит автоматический переход к отрицательным значениям, что важно учитывать при проектировании систем. Также следует с осторожностью изменять параметры последовательностей, используемых в качестве первичных ключей, чтобы избежать нарушения целостности данных.

Удаление последовательностей выполняется командой DROP SEQUENCE (или DROP GENERATOR), но только после проверки отсутствия зависимых объектов - триггеров, хранимых процедур и функций, которые могут ссылаться на удаляемую последовательность.

На практике последовательности чаще всего используются в INSERT-запросах для генерации уникальных идентификаторов новых записей. Однако область их применения не ограничивается первичными ключами - они могут быть полезны в любых сценариях, требующих гарантированно уникальных числовых значений, таких как нумерация документов, генерация кодов и других подобных задач.

При проектировании новых систем рекомендуется использовать современный синтаксис SEQUENCE, а не устаревший GENERATOR. Также предпочтительнее применять конструкцию NEXT VALUE FOR вместо функции GEN\_ID(), так как это соответствует текущим стандартам SQL и обеспечивает лучшую читаемость кода.

оператор MERGE

Упрощенный синтаксис MERGE из диалекта Transact-SQL:

A white text on a white background

AI-generated content may be incorrect.

Здесь [INTO] <target\_table> определяет «целевую» таблицу или представление, данные из которой сравниваются с «исходной» таблицей, представлением или выражением, обозначенным как <table\_source>. По результатам сравнения изменения происходят в целевой таблице. Условия, выполнение которых проверяется в отношении записей исходной и целевой таблиц, определяются выражением ON <merge\_search\_condition>.

Действия, выполняемые при соответствии значений записей условиям, определяет необязательный раздел [WHEN MATCHED [AND <search\_condition>]. Одно выражение MERGE может иметь не больше двух разделов WHEN MATCHED. Если их два, в первом обязательно должны быть дополнительные условия: AND <search condition>. Тогда второй WHEN MATCHED применяется только в тех случаях, если не применяется первый. Если имеются два WHEN MATCHED, один должен указывать действие UPDATE (изменить записи), а другой - действие DELETE (удалить).

Наличие раздела [WHEN NOT MATCHED [BY TARGET] [AND <search\_condition>] говорит, что если в целевой таблице нет строки из «исходной» таблицы (выражения) и такая строка соответствует дополнительным условиям <search\_condition>, то она должна быть вставлена (INSERT) в таблицу target\_table так, как определяется в THEN <merge\_not\_matched>. Инструкция MERGE может иметь только один раздел WHEN NOT MATCHED BY TARGET.

[WHEN NOT MATCHED BY SOURCE [AND <search\_condition>] THEN <merge\_matched>] определяет действия (UPDATE или DELETE) над строками, которые присутствуют в целевой таблице, но отсутствуют в исходной. Как и в случае с WHEN MATCHED, в од- ном выражении MERGE может иметь не больше двух разделов WHEN NOT MATCHED BY SOURCE, один из которых определяет изменение данных, другой - удаление. Рассмотрим пример использования данного оператора. Пусть имеется две таблицы - Subj и NewSubj (табл. 7.3 и 7.4). В первой таблице перечислены читаемые учебные курсы, во второй приведены изменения, вводимые в учебные планы. Поле SubjId является первичным ключом в обеих таблицах.

A screenshot of a computer

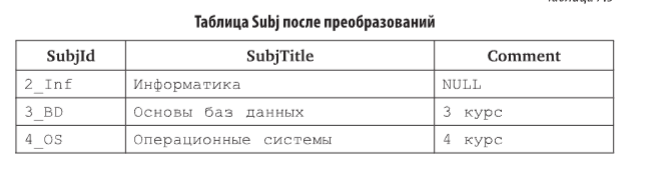
AI-generated content may be incorrect.

Пусть требуется согласовать содержимое таблиц таким образом, что если имеющемуся в таблице Subj курсу в таблице NewSubj дано новое название, то мы обновляем только название, а если В NewSubj появляется новый (по значению SubjId) курс, мы его добавляем в Subj. Это можно сделать следующим выражением:

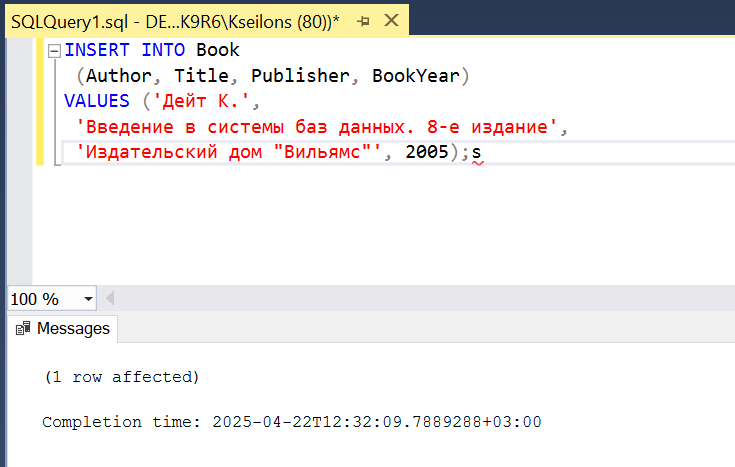
A white text with black text

AI-generated content may be incorrect.

После выполнения данного выражения в таблице Subj будут значения, представленные в табл. 7.5. Нужно отметить, что аналогичный результат можно получить, последовательно выполнив запросы на обновление (UPDATE) и добавление (INSERT) требуемых данных. Однако с точки зрения производительности это будет менее эффективным решением.

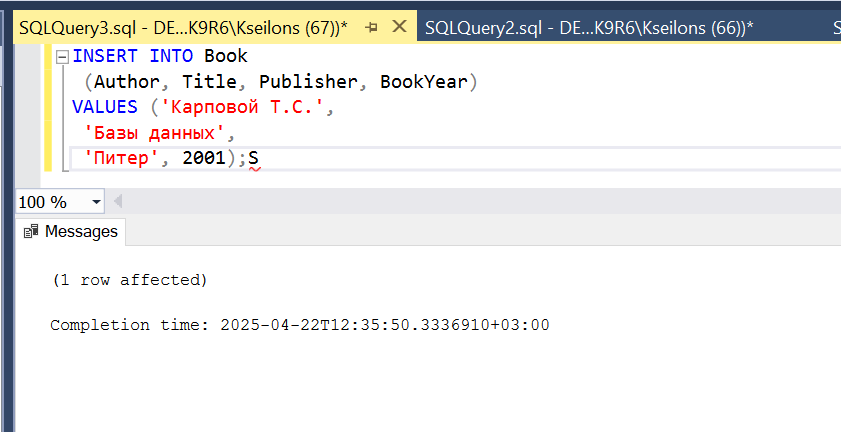


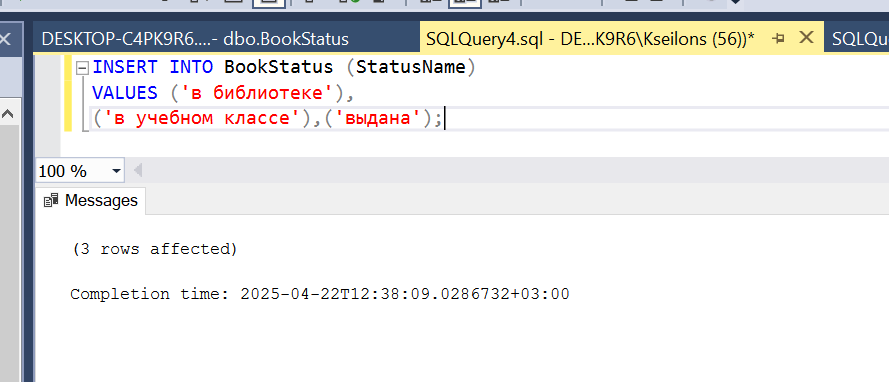
ЛАБОРАТОРНАЯ РАБОТА № 1 (SQL SERVER) ДЕСЯТОЙ НЕДЕЛИ КУРСА «УПРАВЛЕНИЕ ДАННЫМИ».



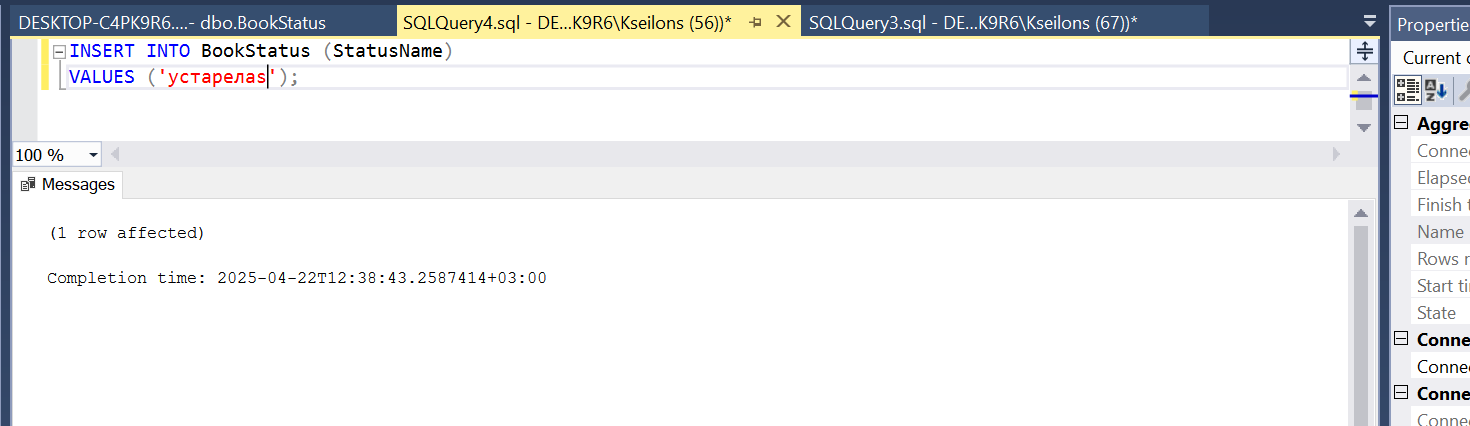
Что касается двойных кавычек, то их использование регулируется параметром SET QUOTED\_IDENTIFIER { ON | OFF } Когда установлено SET QUOTED\_IDENTIFIER ON (по умолчанию), то строки в двойных кавычках рассматриваются как идентификаторы объектов (имена таблиц, столбцов и т.д.). Если параметру SET QUOTED\_IDENTIFIER присвоено значение OFF, строковые константы можно заключать либо в одинарные, либо двойные кавычки

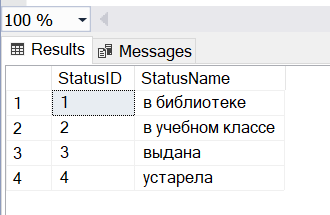
Задание. По аналогии добавьте в таблицу новую запись о книге Карповой Т.С. «Базы данных», вышедшей в издательстве «Питер» в 2001 году.



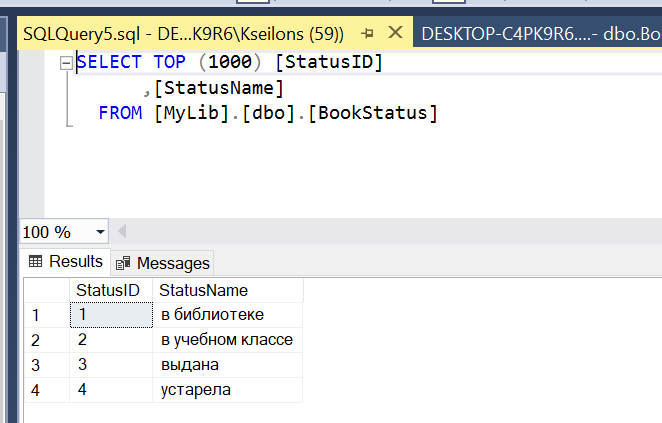


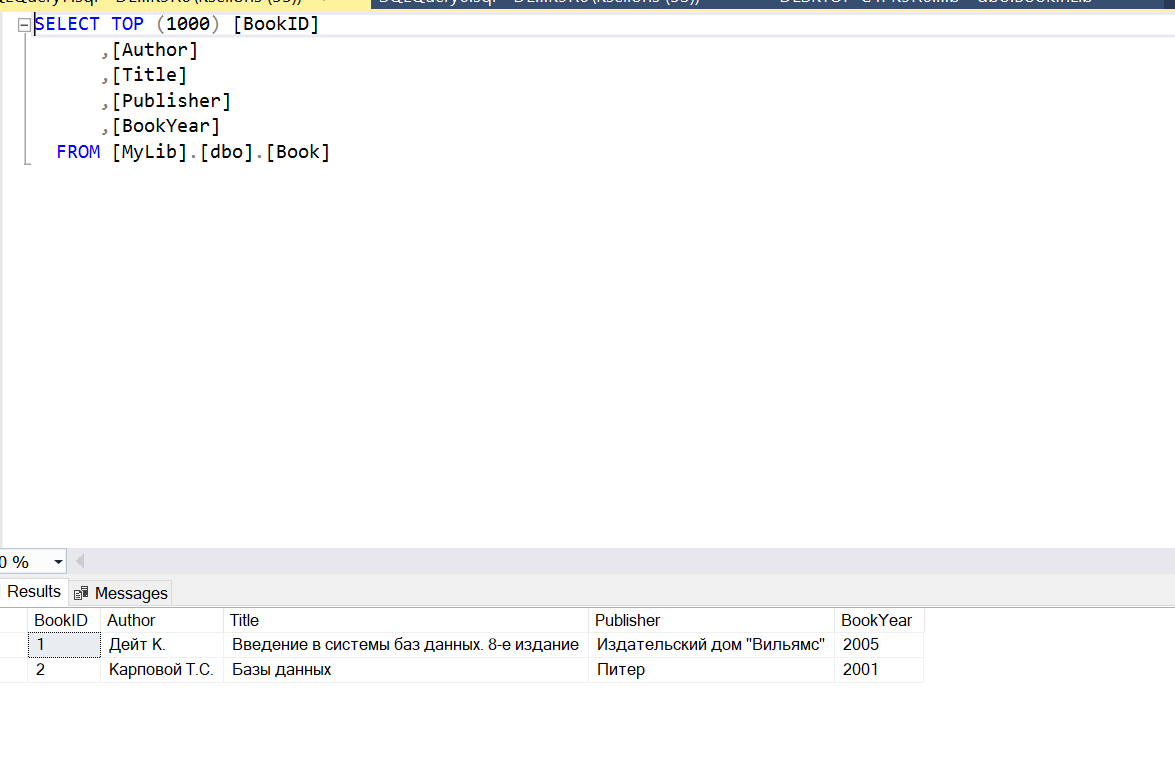
Задание. Используя базу из файлов к лабораторной работе, напишите запрос, добавляющий новый статус «устарела» в таблицу BookStatus

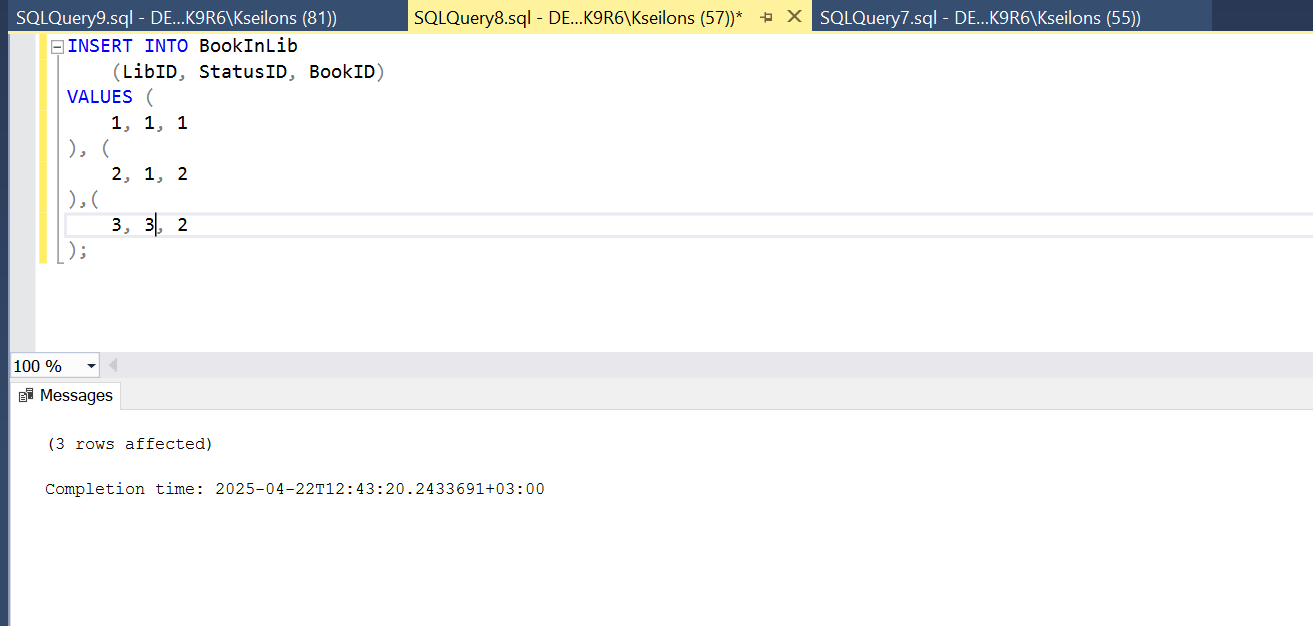


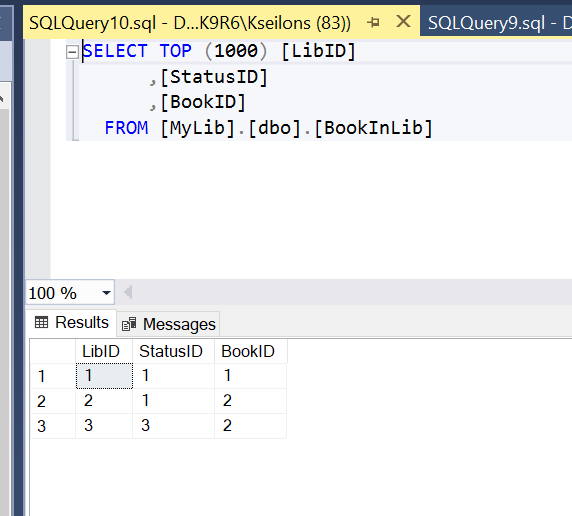


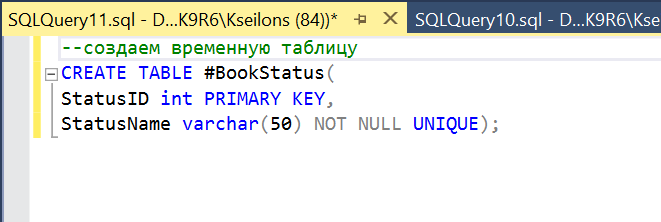
Задание. Посмотрите имеющиеся данные в таблицах Book и BookStatus (например, выполнив Select \* From Book; и аналогично – для второй таблицы). Напишите и выполните скрипт, вносящий 3-5 строк в таблицу BookInLib, обратите внимание, чтобы не нарушались ограничения первичного и внешних ключей. Кроме того, не забудьте, что ключевое поле LibID не определялось как счетчик и его значение надо явно указывать.

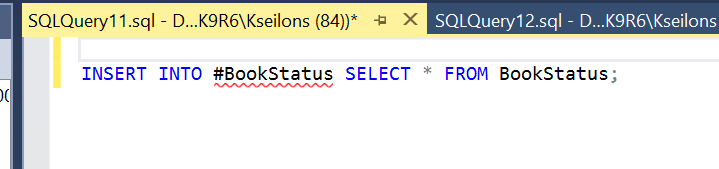




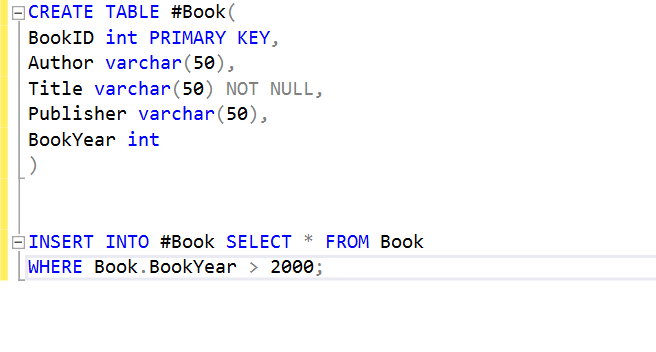


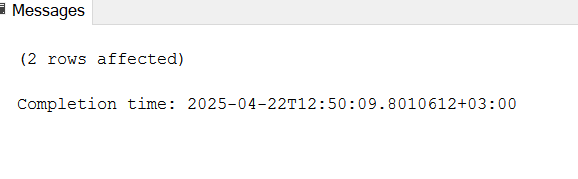


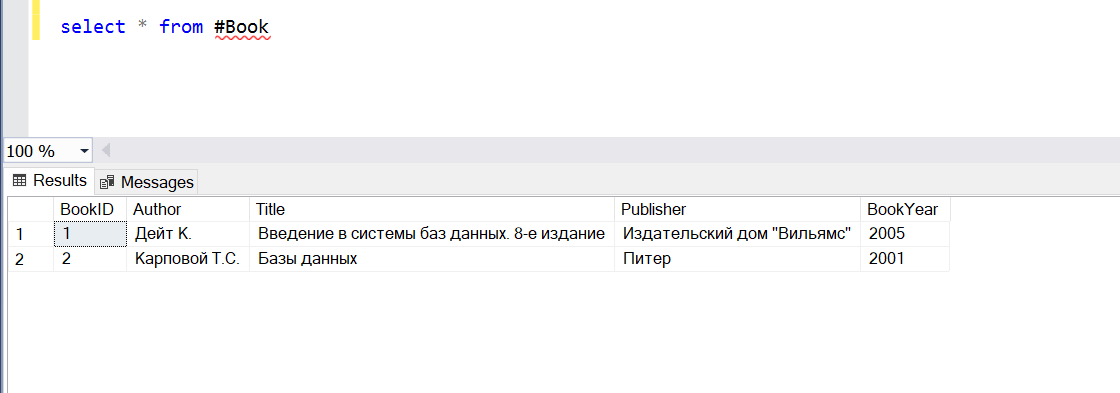


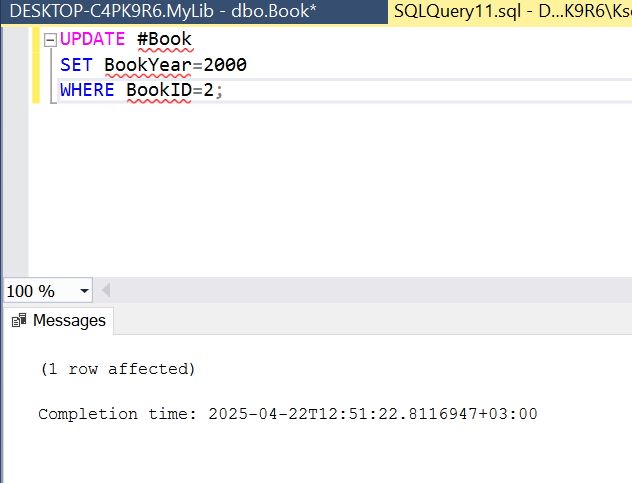


Задание. По аналогии с рассмотренным примером создайте временную таблицу #Book и внесите в нее записи из таблицы Book о книгах, выпущенных после 2000 года.

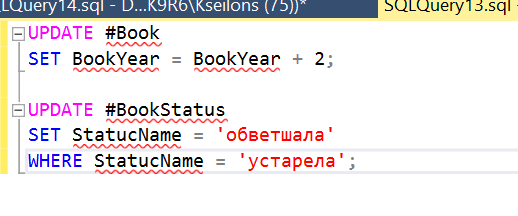


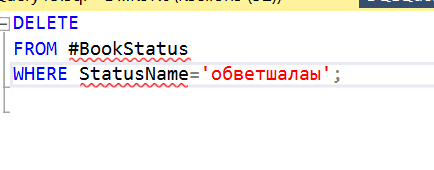




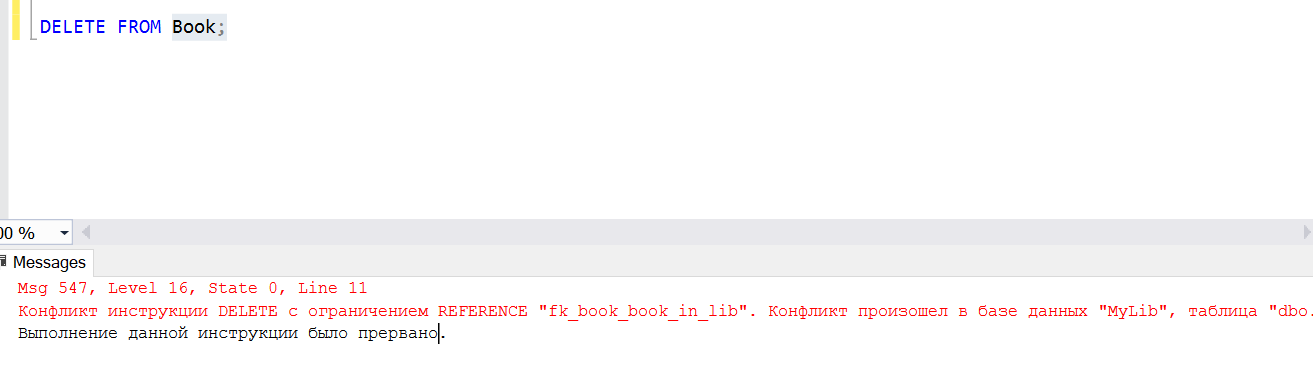


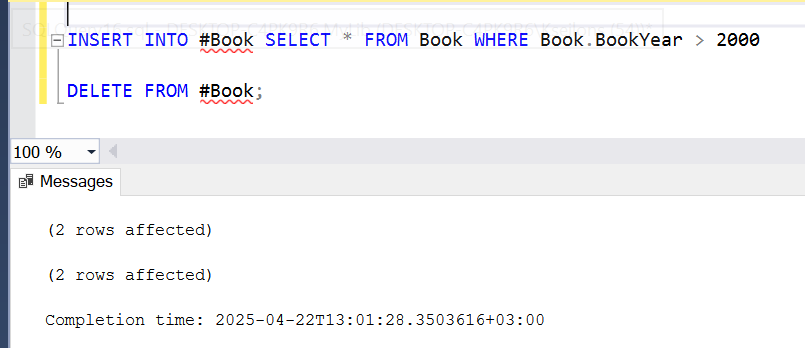
Задание. Напишите два запроса, первый из которых увеличивает год издания всех книг в таблице #Book на 2, а второй изменяет в таблице #BookStatus название статуса c «устарела» на «обветшала»





Задание. Проделайте этот эксперимент и проанализируйте результат. Почему не было проблем при удалении записей из таблицы #Book?





Почему не было проблем при удалении записей из таблицы #Book. Потому что на таблицу #Book не ссылались записи из других таблиц, в отличие от Book, на которую ссылаются записи из BookInLib.