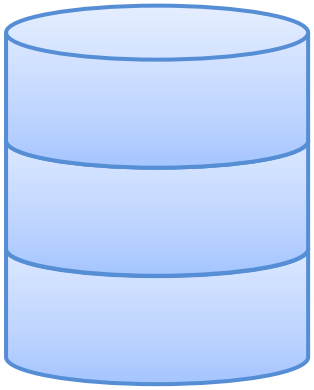
Московский Государственный Университет им. М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования



Практикум по Базам Данных

Методические материалы

Версия от 13.12.2012

Москва 2012

Содержание

Аннотация

Выполняя практикум, студенты получают знания и навыки по современным базам данных и системам управления базами данных. Они изучают основы построения систем баз данных, получают представление о моделировании данных и методах управления данными с помощью языка SQL и других средств современных СУБД. Рассматриваются реляционные базы данных и системы управления базами данных. Знания и навыки приобретаются в ходе выполнения 10 практических заданий.

В качестве базового программного средства выбрана СУБД MS SQL Server 2005. Данная СУБД имеет все типичные компоненты, характерные для программных продуктов подобного рода, позволяет освоить основные приемы работы. На занятиях практикума каждый студент по индивидуальному варианту выполняет разработку модельной корпоративной системы баз данных в среде MS SQL Server 2005, используя такие средства как SQL (T-SQL), триггеры, хранимые процедуры, ограничения прав доступа и проч. Производится оптимизация запросов средствами среды MS SQL Server.

Общие сведения

Программное обеспечение

Для выполнения заданий практикума рекомендуется использовать Microsoft SQL Server 2005 с последним пакетом обновлений (SP4) - достаточно издания Express, бесплатно доступного для загрузки. Microsoft SQL Server 2008 также может быть использован, однако он имеет некоторые отличия. Для работы необходимо установить Database Engine и Microsoft SQL Server Management Studio (или использовать сервер spprac.cs.msu.ru и выданные логин\пароль - в этом случае можно не устанавливать Database Engine). Настоятельно рекомендуется установить Microsoft SQL Server Books Online и использовать в качестве основного справочного материала. Books Online для SQL Server 2005 также доступны в Интернете по адресу: http://msdn.microsoft.com/en-us/library/ms130214(v=sql.90).aspx.

Порядок сдачи и сроки

Практические задания выполняются по индивидуальным вариантам. Номер варианта выдается на первом занятии, формулировки заданий индивидуальных вариантов доступны по адресу http://bdis.umeta.ru/db/db_course/labs/index.html.

При сдаче в установленные сроки каждое задание может быть оценено максимум на 8 баллов (4 балла – оценка за обязательную часть, 4 балла – оценка за ответы на дополнительные вопросы). Приведенная ниже таблица показывает, сколько заданий должно быть сдано студентом для того, чтобы текущее сдаваемое задание оценивалось, исходя из максимума в 8 баллов. Если заданий сдано меньше (и отсутствует уважительная причина), то максимальная оценка составляет 4 балла, а дополнительные вопросы не задаются (исключение - задание №2). Первой неделей считается неделя первого занятия (организационного собрания).

№ недели	Количество уже **сданных** заданий
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10

Время, отведенное на сдачу каждого практического задания – одно занятие. Студенты сдают задания в порядке очереди. Допускается сдача нескольких заданий на одном занятии, однако при сдаче каждого последующего задания студент оказывается в хвосте очереди, а в конце занятия все оценки за сдаваемые задания выставляются. Схема сдачи заданий в общем случае выглядит следующим образом:

- Студент показывает результаты выполнения задания. При этом все подготовленные запросы должны выполняться успешно, в противном случае задание отправляется на доработку, а оценка не выставляется.

- Преподаватель проверяет правильность написанных запросов. Баллы за основную часть могут быть снижены, если основная часть выполнена не полностью, запросы работают неверно, очень неоптимальны, или если студент не может объяснить принцип их работы.

- Также могут быть заданы вопросы по тематике практического задания. Баллы за основную часть снижаются, если студент дает неверные, неполные или неточные ответы.

- Если задание сдается вовремя, то после сдачи основной части студенту задаётся несколько дополнительных вопросов и дается время на подготовку (до конца занятия). В конце занятия выставляется оценка.

- Баллы за дополнительную часть снижаются, если дополнительные задания не выполнены, выполнены не полностью или неверно. Также баллы могут быть снижены за очень неоптимальные решения.

Исключением является лишь задание №2 – за него всегда можно получить до 8 баллов и сдавать несколько раз. Задания могут сдаваться в произвольном порядке, однако задания №4-10 могут сдаваться только после сдачи заданий №2 и №3, так как опираются на базу данных из индивидуального варианта.

Итоговая оценка

Практикум состоит из 10 практических заданий, максимальная оценка за каждое из которых составляет 8 баллов. Форма отчетности – зачет с оценкой. По результатам выполнения практических заданий в конце семестра выставляется оценка:

- **«Удовлетворительно»**, если набрано от 20 до 39 баллов включительно.
- **«Хорошо»**, если набрано от 40 до 59 баллов включительно.
- **«Отлично»**, если набрано 60 и более баллов.

В случае если до желаемой оценки студенту не хватает небольшого количества баллов, то возможно выполнение дополнительных заданий по договоренности с преподавателями. Если после зачета у студента не набирается баллов на оценку «удовлетворительно», на последующих попытках сдачи максимум за каждое задание (кроме задания №2) составляет 2 балла. Задание №2 всегда оценивается исходя из максимума в 8 баллов.

Текущие результаты будут публиковаться на Web-страничке практикума: (http://spprac.cs.msu.ru/). Просьба следить за своими оценками и сообщать преподавателям в случае обнаружения ошибок.

Практическое задание №1. Введение в SQL и MS SQL Server

Постановка задачи

Первое практическое задание заключается в знакомстве со средой Microsoft SQL Server 2005 Management Studio и написании простейших SQL-запросов с использованием оператора SELECT. Для модельной базы данных «King Corporation» должны быть составлены 4 запроса согласно индивидуальному варианту. Скрипт для создания и заполнения модельной базы данных и её описание, а также содержание индивидуальных вариантов доступны по адресу http://bdis.umeta.ru/db. После составления запросов следует убедиться в их правильности при помощи более простых запросов.

Темы для проработки

- Основные понятия реляционных и SQL-ориентированных баз данных.
- Оператор SELECT.
- Агрегатные и встроенные функции Microsoft SQL Server.
- Структура учебной базы данных «King Corporation».
- Работа в среде Microsoft SQL Server Management Studio.

Примеры

Рассмотрим простой пример: определить минимальную сумму заказа товара «SB ENERGY BAR-6 PACK».

```
SELECT MIN(total) AS min_total
      FROM item JOIN product ON item.product_id = product.product_id
      WHERE description = 'SB ENERGY BAR-6 PACK'
```

```
min_total
---------------------------------------
   2.40
```

В этом запросе используется агрегатная функция MIN, операция внутреннего соединения таблиц по условию (INNER JOIN или просто JOIN), а также ограничение WHERE.

Рассмотрим еще один пример: выбрать продавца, у которого наивысшее отношение комиссионные / зарплата. Сначала рассмотрим вариант с вложенными подзапросами:

```
SELECT first_name, last_name, commission/salary AS rate
      FROM employee
      WHERE commission IS NOT NULL
      AND commission/salary =
        (SELECT MAX(commission/salary)
           FROM employee
           WHERE commission IS NOT NULL)
```

```
first_name      last_name       rate
--------------- --------------- ---------------------------------------
KENNETH         MARTIN          1.1200000000
```

Убедиться в том, что этот запрос работает верно, можно при помощи более простого запроса:

```
SELECT first_name, last_name, commission/salary AS rate
      FROM employee
```

```
      WHERE commission IS NOT NULL
      ORDER BY commission/salary DESC
```

```
first_name      last_name       rate
--------------- --------------- ---------------------------------------
KENNETH         MARTIN          1.1200000000
KAREN           SHAW            0.9600000000
RAYMOND         PORTER          0.7200000000
LIVIA           WEST            0.6666666666
PAUL            ROSS            0.6153846153
CYNTHIA         WARD            0.4000000000
DANIEL          PETERS          0.2400000000
GREGORY         LANGE           0.2400000000
KEVIN           ALLEN           0.1875000000
MARY            TURNER          0.0000000000
```

Условие (commission IS NOT NULL) является необходмым, так как столбец commission допускает неопределённые значения.

Рассмотрим теперь другие варианты написания этого запроса. Например, можно избавиться от подзапросов, применив сортировку и ограничение количества результатов:

```
SELECT TOP 1 first_name, last_name, commission/salary AS rate
      FROM employee
      WHERE commission IS NOT NULL
      ORDER BY commission/salary DESC
```

```
first_name      last_name       rate
--------------- --------------- ---------------------------------------
KENNETH         MARTIN          1.1200000000

(1 row(s) affected)
```

Результат выполнения этого запроса совпадает с результатом выполнения исходного варианта, однако здесь кроется распространенная ошибка: в случае, если в базе данных будет более одного продавца с максимальным отношением комиссионные / зарплата, то выводиться будет всё равно лишь один из них. Этого легко избежать, применив конструкцию WITH TIES (подробнее об операторе SELECT см. Books Online). Правильный вариант запроса без использования подзапросов будет выглядеть так:

```
SELECT TOP 1 WITH TIES first_name, last_name, commission/salary as rate
      FROM employee
      WHERE commission IS NOT NULL
      ORDER BY commission/salary DESC
```

```
first_name      last_name       rate
--------------- --------------- ---------------------------------------
KENNETH         MARTIN          1.1200000000
```

Следующий пример: выбрать среднегодовую сумму заказов покупателя «REBOUND SPORTS».

```
SELECT s/ny AS average
  FROM
    (SELECT SUM(total) AS s
       FROM customer JOIN sales_order
       ON customer.customer_id = sales_order.customer_id
       WHERE name = 'REBOUND SPORTS') t1,
    (SELECT COUNT(DISTINCT YEAR(order_date)) AS ny
       FROM customer JOIN sales_order
       ON customer.customer_id = sales_order.customer_id
       WHERE name = 'REBOUND SPORTS') t2
```

```
average
---------------------------------------
1810.866666
```

В первом подзапросе подсчитывается общая сумма заказов для покупателя «REBOUND SPORTS», а во втором считается количество лет, в которые этот покупатель что-то заказывал. Здесь следует обратить внимание на ключевое слово DISTINCT, позволяющее подсчитать количество неповторяющихся значений.

В качестве последнего примера рассмотрим простейший запрос, который, тем не менее, часто составляется неверно: найти среднее значение quantity в таблице item. Очевидное решение выглядит следующим образом:

```
SELECT AVG(quantity) AS average_quantity FROM item
```

```
average_quantity
---------------------------------------
81
```

Проблема заключается в том, что результат применения агрегатной функции в SQL Server имеет тот же тип, что и её аргумент. Таким образом, в этом примере необходимо преобразовать значения целочисленного столбца quantity:

```
SELECT AVG(1.0*quantity) AS average_quantity FROM item
```

```
average_quantity
---------------------------------------
81.881918
```

То же самое необходимо помнить и при делении.

Примеры вопросов по обязательной части

- Объяснить, как работают написанные запросы.
- Рассказать про операцию соединения (JOIN) и различные её разновидности.
- Рассказать про агрегатные функции, предложения GROUP BY и HAVING.
- Как выбрать только уникальные значения какого-либо столбца?
- Как осуществить сортировку по возрастанию/убыванию по значению какого-либо столбца?
- Как агрегатные функции ведут себя по отношению к неопределённым значениям?
- Рассказать о теоретико-множественных операциях в SQL.
- Чем отличаются UNION и UNION ALL?
- Чем отличаются COUNT(*) и COUNT(field)?
- Как подсчитать количество уникальных значений столбца?
- Как можно осуществить проверку на неопределенное значение?
- Рассказать про предикат LIKE.
- Как можно выбрать только определенное количество строк?
- Чем SQL-таблица отличается от отношения?

Примеры дополнительных вопросов

- Исправить неверно работающий запрос (запросы).
- Упростить один или несколько запросов.
- Округлить результирующее значение до 3 знаков после точки.
- Округлить вещественное число до целого без нулей после точки.
- Переписать запрос, не используя функцию MAX (MIN).
- Изменить формат вывода данных (например, формат даты и времени).

- Написать или модифицировать запрос по сформулированному заданию.

Практическое задание №2. Проектирование схемы базы данных

Постановка задачи

Второе практическое задание связано с проектированием схемы базы данных. Каждый индивидуальный вариант содержит ER-диаграмму некоторой предметной области, иногда очень приблизительную (она может быть модифицирована, но не в сторону упрощения). Задачей студента является решить, для чего будет использоваться создаваемая база данных, и, исходя из этого, построить её концептуальную схему. Результатом данного практического задания является схема базы данных (в виде диаграммы, содержащей таблицы и связи между ними, без уточнения типов столбцов). Для проектирования схемы и построения диаграммы рекомендуется использовать среду Microsoft SQL Server Management Studio, хотя можно использовать и другие средства. Максимальная оценка за данное задание всегда составляет 8 баллов, а студент может сдавать схему несколько раз, исправляя замечания преподавателей. При сдаче задания студент должен обосновать соответствие созданной схемы поставленной задаче.

Темы для проработки

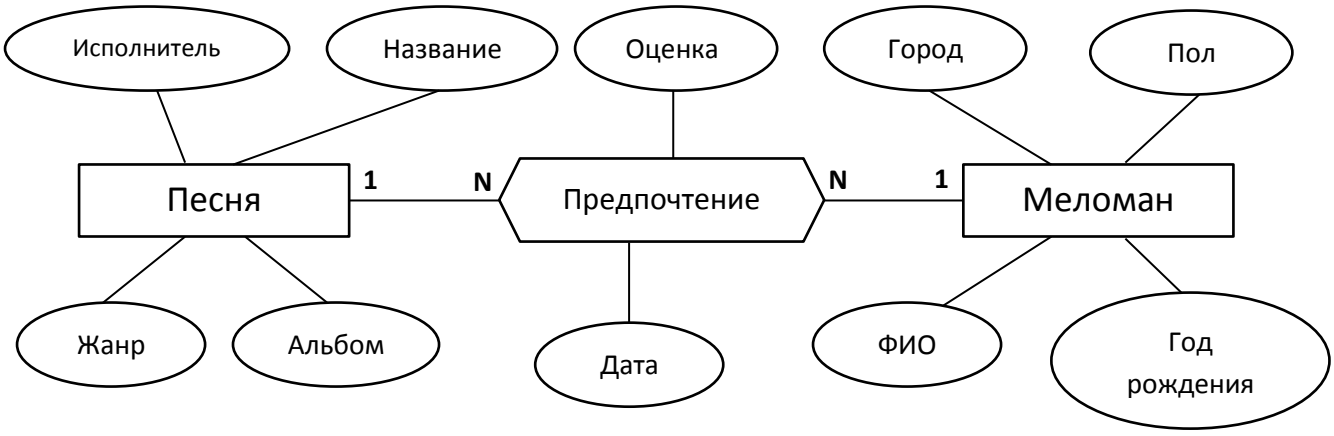
- Модель "сущность-связь" (ER-модель).
- Первичные и внешние ключи.
- Типы связей и их моделирование.
- Нормальные формы и нормализация.

Требования к схеме

- Схема должна соответствовать поставленной задаче.
- Связи между сущностями должны быть правильно смоделированы.
- Таблицы должны удовлетворять, по крайней мере, третьей нормальной форме.
- Желательно придерживаться какой-либо системы в именовании таблиц и столбцов.

Примеры

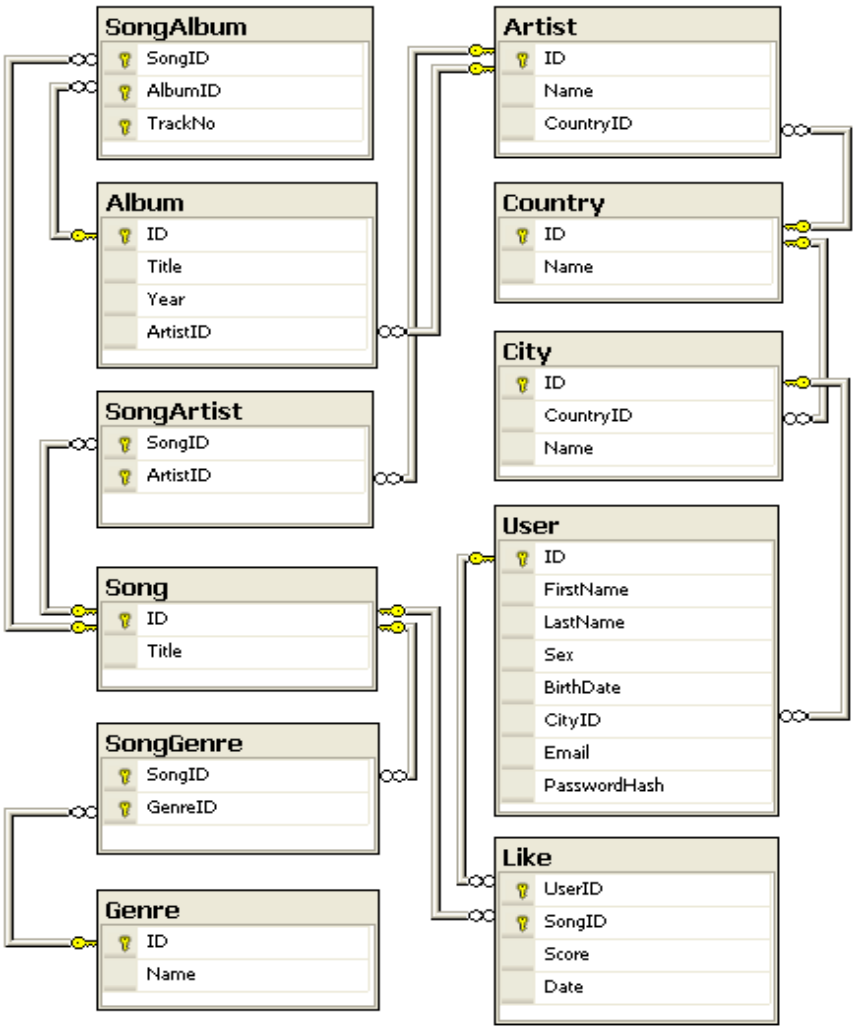
В качестве примера рассмотрим предметную область «Музыкальные предпочтения»:



Первым шагом в проектировании схемы базы данных является определение цели создания базы данных. Зададимся целью создать простую социальную сеть для меломанов, позволяющую им:

- Находить людей, которые имеют схожие музыкальные вкусы.
- Получать от системы рекомендации, какие еще песни, альбомы и исполнители могут им понравиться.
- Просматривать рейтинг исполнителей, альбомов и композиций.

Схема базы данных для этого случая может выглядеть, например, следующим образом:



Рассмотрим получившиеся таблицы и некоторые рассуждения, которые привели к приведенной схеме:

- Таблица **User** хранит данные о пользователе-меломане. Помимо базовых данных, таких как имя, фамилия, пол, дата рождения и город, эта таблица хранит также адрес электронной почты для связи и авторизации в предполагаемой социальной сети (а также хэш пароля).

- Таблицы **City** и **Country** служат справочниками для городов и стран, позволяя избежать избыточности и аномалий обновления.

- Таблица **Song** хранит уникальный идентификатор и название песни. Таблица **Genre** является справочником жанров, а таблица **SongGenre** связывает песни и жанры, моделируя связь типа «многие-ко-многим», так как одна песня может быть помечена несколькими жанрами, а также каждый жанр может включать несколько песен.

- Таблица **Album** хранит информацию об альбомах. Так как одна песня может быть включена в несколько альбомов, а каждый альбом может содержать несколько песен, но требуется таблица **SongAlbum** для организации связи «многие-ко-многим». Эта таблица также хранит номер трека данной песни в альбоме.

- Важным столбцом в таблице **Album** является ссылка на исполнителя, если это альбом одного исполнителя. Если же это альбом-сборник, то *ArtistID* будет содержать неопределенное значение.

- Таблица **Artist** хранит данные об исполнителях, включая страну исполнителя. Исполнители связаны с песнями опять же с помощью связи «многие-ко-многим» (таблица **SongArtist**), учитывая случаи, когда у песни есть несколько исполнителей.

- Наконец, таблица **Like** хранит предпочтения пользователей: какая песня, когда и насколько понравилась - пусть Score отражает оценку пользователем песни от 1 («ничего особенного») до 5 («очень нравится»).

Практическое задание №3. Создание и заполнение таблиц в MS SQL Server

Постановка задачи

Третье практическое задание заключается в подготовке SQL-скрипта для создания таблиц согласно схеме, полученной в предыдущем задании (с уточнением типов столбцов). Необходимо определить первичные и внешние ключи, а также декларативные ограничения целостности (возможность принимать неопределенное значение, уникальные ключи, проверочные ограничения и т. д.). Таблицы следует создавать в отдельной базе данных.

Кроме того, нужно подготовить данные для заполнения созданных таблиц. Объем подготовленных данных должен составлять не менее 10 экземпляров для каждой из стержневых сущностей и 20 экземпляров для каждой из ассоциативных. На основе этих данных необходимо создать SQL-скрипт для вставки соответствующих строк в таблицы БД.

Темы для проработки

- Язык DDL, операторы CREATE TABLE и ALTER TABLE.
- Типы данных MS SQL Server.
- Декларативные ограничения целостности.
- Создание таблиц в среде Microsoft SQL Server Management Studio.
- Оператор INSERT.

Примеры

Рассмотрим скрипт для создания схемы базы данных, спроектированной в задании №2:

```
CREATE TABLE [Country]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](100) NOT NULL,
	CONSTRAINT [PK_Country] PRIMARY KEY ([ID] ASC)
);

CREATE TABLE [Song]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Title] [varchar](200) NOT NULL,
	CONSTRAINT [PK_Song] PRIMARY KEY ([ID] ASC)
);

CREATE TABLE [Genre]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](100) NOT NULL,
	CONSTRAINT [PK_Genre] PRIMARY KEY ([ID] ASC)
);

CREATE TABLE [Artist]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](100) NOT NULL,
	[CountryID] [int] NOT NULL,
	CONSTRAINT [PK_Artist] PRIMARY KEY ([ID] ASC)
```

```
);

CREATE TABLE [City]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[CountryID] [int] NOT NULL,
	[Name] [varchar](100) NOT NULL,
	CONSTRAINT [PK_City] PRIMARY KEY ([ID] ASC)
);

CREATE TABLE [Like]
(
	[UserID] [int] NOT NULL,
	[SongID] [int] NOT NULL,
	[Score] [tinyint] NOT NULL,
	[Date] [datetime] NOT NULL CONSTRAINT [DF_Like_Date] DEFAULT
(getdate()),
	CONSTRAINT [PK_Like] PRIMARY KEY
	(
		[UserID] ASC,
		[SongID] ASC
	)
);

CREATE TABLE [SongGenre]
(
	[SongID] [int] NOT NULL,
	[GenreID] [int] NOT NULL,
	CONSTRAINT [PK_SongGenre] PRIMARY KEY
	(
		[SongID] ASC,
		[GenreID] ASC
	)
);

CREATE TABLE [SongArtist]
(
	[SongID] [int] NOT NULL,
	[ArtistID] [int] NOT NULL,
	CONSTRAINT [PK_SongArtist] PRIMARY KEY
	(
		[SongID] ASC,
		[ArtistID] ASC
	)
);

CREATE TABLE [SongAlbum]
(
	[SongID] [int] NOT NULL,
	[AlbumID] [int] NOT NULL,
	[TrackNo] [tinyint] NOT NULL,
	CONSTRAINT [PK_AlbumArtist] PRIMARY KEY
	(
		[AlbumID] ASC,
		[SongID] ASC,
		[TrackNo] ASC
	)
);

CREATE TABLE [Album]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Title] [varchar](200) NOT NULL,
```

```
	[Year] [int] NOT NULL,
	[ArtistID] [int] NULL,
	CONSTRAINT [PK_Album] PRIMARY KEY ([ID] ASC)
);

CREATE TABLE [User]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [varchar](100) NOT NULL,
	[LastName] [varchar](100) NOT NULL,
	[Sex] [bit] NOT NULL,
	[BirthDate] [datetime] NOT NULL,
	[CityID] [int] NOT NULL,
	[Email] [varchar](100) NOT NULL,
	[PasswordHash] [varchar](100) NOT NULL,
	CONSTRAINT [PK_User] PRIMARY KEY ([ID] ASC)
);

CREATE UNIQUE INDEX [IX_User] ON [User]
(
	[Email] ASC
);

CREATE UNIQUE INDEX [IX_Country] ON [Country]
(
	[Name] ASC
);

CREATE UNIQUE INDEX [IX_Genre] ON [Genre]
(
	[Name] ASC
);

ALTER TABLE [Artist]
	ADD CONSTRAINT [FK_Artist_Country] FOREIGN KEY ([CountryID])
		REFERENCES [Country] ([ID]);

ALTER TABLE [City]
	ADD CONSTRAINT [FK_City_Country] FOREIGN KEY ([CountryID])
		REFERENCES [Country] ([ID]);

ALTER TABLE [Like]
	ADD CONSTRAINT [FK_Like_Song] FOREIGN KEY ([SongID])
		REFERENCES [Song] ([ID]);

ALTER TABLE [Like]
	ADD CONSTRAINT [FK_Like_User] FOREIGN KEY ([UserID])
		REFERENCES [User] ([ID]);

ALTER TABLE [Like]
	ADD CONSTRAINT [CK_Like_Score]
		CHECK ([Score] > 0 AND [Score] < 6);

ALTER TABLE [SongGenre]
	ADD CONSTRAINT [FK_SongGenre_Genre] FOREIGN KEY ([GenreID])
		REFERENCES [Genre] ([ID]);

ALTER TABLE [SongGenre]
	ADD CONSTRAINT [FK_SongGenre_Song] FOREIGN KEY ([SongID])
		REFERENCES [Song] ([ID]);

ALTER TABLE [SongArtist]
	ADD CONSTRAINT [FK_SongArtist_Artist] FOREIGN KEY ([ArtistID])
```

```
            REFERENCES [Artist] ([ID]);

ALTER TABLE [SongArtist]
      ADD CONSTRAINT [FK_SongArtist_Song] FOREIGN KEY ([SongID])
            REFERENCES [Song] ([ID]);

ALTER TABLE [SongAlbum]
      ADD CONSTRAINT [FK_SongAlbum_Album] FOREIGN KEY ([AlbumID])
            REFERENCES [Album] ([ID]);

ALTER TABLE [SongAlbum]
      ADD CONSTRAINT [FK_SongAlbum_Song] FOREIGN KEY ([SongID])
            REFERENCES [Song] ([ID]);

ALTER TABLE [SongAlbum]
      ADD CONSTRAINT [CK_SongAlbum_TrackNo]
            CHECK ([TrackNo] > 0);

ALTER TABLE [Album]
      ADD CONSTRAINT [FK_Album_Artist] FOREIGN KEY ([ArtistID])
            REFERENCES [Artist] ([ID]);

ALTER TABLE [Album]
      ADD CONSTRAINT [CK_Album_Year]
            CHECK (([Year] > 1900) AND ([Year] <= YEAR(getdate())));

ALTER TABLE [User]
      ADD  CONSTRAINT [FK_User_City] FOREIGN KEY ([CityID])
            REFERENCES [City] ([ID]);

ALTER TABLE [User]
      ADD CONSTRAINT [CK_User_BirthDate]
            CHECK (YEAR(getdate()) - YEAR([BirthDate]) >= 7);
```

Приведенный выше скрипт содержит:

- Определения таблиц, первичных и внешних ключей.
- Ограничения CHECK: например, указано, что пользователь может оценивать, насколько ему нравится песня по шкале от 1 до 5, другие значения *Score* считаются недопустимыми. Также задается ограничение и на дату рождения пользователя и год выпуска альбома.
- Значение по умолчанию для столбца *Date* в таблице **Like** – текущее время.
- Ограничения NOT NULL. Столбец *ArtistID* в таблице **Album** допускает неопределенные значения – если у альбома нет определенного исполнителя.
- Ограничение на столбец *Email* в таблице **User** – допускаются только уникальные значения. Аналогичным образом не допускается появления двух стран с одинаковым названием *Name* в таблице **Country**, а также двух жанров с одинаковым названием в таблице **Genre**.

Полезно также подготовить скрипт для удаления таблиц, так как оператор CREATE TABLE вернет ошибку, если таблица уже существует:

```
DROP TABLE [SongGenre];
DROP TABLE [SongAlbum];
DROP TABLE [SongArtist];
DROP TABLE [Like];
DROP TABLE [Album];
DROP TABLE [Song];
DROP TABLE [Artist];
DROP TABLE [User];
DROP TABLE [City];
```

```
DROP TABLE [Country];
DROP TABLE [Genre];
```

Обратите внимание, что сначала нужно удалять таблицы, на которые нет ссылок из других таблиц, иначе сработают ограничения ссылочной целостности.

Теперь рассмотрим заполнение базы данных модельным содержимым в соответствии с постановкой задачи. SQL-скрипт для этого может выглядеть, например, так (обратите внимание на предварительную очистку таблиц и правильный порядок заполнения - чтобы не нарушались ограничения ссылочной целостности):

```
DELETE FROM [SongAlbum]
DELETE FROM [Album]
DELETE FROM [SongArtist]
DELETE FROM [SongGenre]
DELETE FROM [Like]
DELETE FROM [User]
DELETE FROM [City]
DELETE FROM [Artist]
DELETE FROM [Genre]
DELETE FROM [Song]
DELETE FROM [Country]

SET IDENTITY_INSERT [Country] ON
INSERT [Country] ([ID], [Name]) VALUES (1, 'Germany')
INSERT [Country] ([ID], [Name]) VALUES (2, 'Great Britain')
INSERT [Country] ([ID], [Name]) VALUES (3, 'USA')
INSERT [Country] ([ID], [Name]) VALUES (4, 'Russia')
INSERT [Country] ([ID], [Name]) VALUES (5, 'Canada')
INSERT [Country] ([ID], [Name]) VALUES (6, 'Finland')
INSERT [Country] ([ID], [Name]) VALUES (7, 'Sweden')
INSERT [Country] ([ID], [Name]) VALUES (8, 'Japan')
INSERT [Country] ([ID], [Name]) VALUES (9, 'Korea')
INSERT [Country] ([ID], [Name]) VALUES (10, 'Ukrain')
INSERT [Country] ([ID], [Name]) VALUES (11, 'Switzerland')
INSERT [Country] ([ID], [Name]) VALUES (12, 'France')
INSERT [Country] ([ID], [Name]) VALUES (13, 'Italy')
INSERT [Country] ([ID], [Name]) VALUES (14, 'Poland')
INSERT [Country] ([ID], [Name]) VALUES (15, 'Greece')
INSERT [Country] ([ID], [Name]) VALUES (16, 'Turkey')
INSERT [Country] ([ID], [Name]) VALUES (17, 'Estonia')
INSERT [Country] ([ID], [Name]) VALUES (18, 'Latvia')
INSERT [Country] ([ID], [Name]) VALUES (19, 'Romania')
INSERT [Country] ([ID], [Name]) VALUES (20, 'Brazil')
SET IDENTITY_INSERT [Country] OFF

SET IDENTITY_INSERT [Song] ON
INSERT [Song] ([ID], [Title]) VALUES (1, 'Drifting Sun')
INSERT [Song] ([ID], [Title]) VALUES (2, 'Fly to the Rainbow')
INSERT [Song] ([ID], [Title]) VALUES (3, 'Dark Lady')
INSERT [Song] ([ID], [Title]) VALUES (4, 'In Trance')
INSERT [Song] ([ID], [Title]) VALUES (5, 'Life is Like A River')
INSERT [Song] ([ID], [Title]) VALUES (6, 'I Saw Her Standing There')
INSERT [Song] ([ID], [Title]) VALUES (7, 'Misery')
INSERT [Song] ([ID], [Title]) VALUES (8, 'Anna (Go to Him)')
INSERT [Song] ([ID], [Title]) VALUES (9, 'Chains')
INSERT [Song] ([ID], [Title]) VALUES (10, 'Boys')
INSERT [Song] ([ID], [Title]) VALUES (11, 'Good Times, Bad Times')
INSERT [Song] ([ID], [Title]) VALUES (12, 'Babe I'm Gonna Leave You')
INSERT [Song] ([ID], [Title]) VALUES (13, 'You Shook Me')
INSERT [Song] ([ID], [Title]) VALUES (14, 'Dazed And Confused')
INSERT [Song] ([ID], [Title]) VALUES (15, 'Your Time Is Gonna Come')
INSERT [Song] ([ID], [Title]) VALUES (16, 'Keep Yourself Alive')
INSERT [Song] ([ID], [Title]) VALUES (17, 'Doing Alright')
INSERT [Song] ([ID], [Title]) VALUES (18, 'Great King Rat')
INSERT [Song] ([ID], [Title]) VALUES (19, 'My Fairy King')
INSERT [Song] ([ID], [Title]) VALUES (20, 'Liar')
SET IDENTITY_INSERT [Song] OFF

SET IDENTITY_INSERT [Genre] ON
INSERT [Genre] ([ID], [Name]) VALUES (1, 'Heavy Metal')
INSERT [Genre] ([ID], [Name]) VALUES (2, 'Hard Rock')
INSERT [Genre] ([ID], [Name]) VALUES (3, 'Rock-n-Roll')
INSERT [Genre] ([ID], [Name]) VALUES (4, 'Blues Rock')
```

```
INSERT [Genre] ([ID], [Name]) VALUES (5, 'Blues')
INSERT [Genre] ([ID], [Name]) VALUES (6, 'Rock')
INSERT [Genre] ([ID], [Name]) VALUES (7, 'Punk')
INSERT [Genre] ([ID], [Name]) VALUES (8, 'Punk Rock')
INSERT [Genre] ([ID], [Name]) VALUES (9, 'Metal')
INSERT [Genre] ([ID], [Name]) VALUES (10, 'Progressive Metal')
INSERT [Genre] ([ID], [Name]) VALUES (11, 'Pop')
INSERT [Genre] ([ID], [Name]) VALUES (12, 'Classical')
INSERT [Genre] ([ID], [Name]) VALUES (13, 'Symphonic Metal')
INSERT [Genre] ([ID], [Name]) VALUES (14, 'Drum-n-Bass')
INSERT [Genre] ([ID], [Name]) VALUES (15, 'Club')
INSERT [Genre] ([ID], [Name]) VALUES (16, 'Electro')
INSERT [Genre] ([ID], [Name]) VALUES (17, 'Folk')
INSERT [Genre] ([ID], [Name]) VALUES (18, 'Folk Metal')
INSERT [Genre] ([ID], [Name]) VALUES (19, 'Folk Rock')
INSERT [Genre] ([ID], [Name]) VALUES (20, 'Pop Punk')
SET IDENTITY_INSERT [Genre] OFF

SET IDENTITY_INSERT [Artist] ON
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (1, 'The Scorpions', 1)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (2, 'The Beatles', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (3, 'Led Zeppelin', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (4, 'Queen', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (5, 'Black Sabbath', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (6, '3 Doors Down', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (7, 'The Offspring', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (9, 'Madonna', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (10, 'The Animals', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (11, 'Pink Floyd', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (12, 'Deep Purple', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (13, 'Jimi Hendrix', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (14, 'The Who', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (15, 'Aerosmith', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (16, 'The Police', 2)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (18, 'Eagles', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (19, 'System Of A Down', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (22, 'The Misfits', 3)
INSERT [Artist] ([ID], [Name], [CountryID]) VALUES (23, 'Elvis Presley', 3)
SET IDENTITY_INSERT [Artist] OFF

SET IDENTITY_INSERT [City] ON
INSERT [City] ([ID], [CountryID], [Name]) VALUES (1, 1, 'Berlin')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (2, 2, 'Londo')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (3, 3, 'New York')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (4, 3, 'Washington')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (5, 3, 'San-Francisco')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (6, 4, 'Moscow')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (7, 4, 'St. Petersburg')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (8, 5, 'Vancouver')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (9, 6, 'Helsinki')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (10, 7, 'Stockholm')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (11, 8, 'Tokio')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (12, 9, 'Seoul')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (13, 10, 'Kiev')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (15, 11, 'Bern')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (16, 12, 'Paris')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (17, 13, 'Rome')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (18, 14, 'Warsaw')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (19, 15, 'Athens')
INSERT [City] ([ID], [CountryID], [Name]) VALUES (20, 16, 'Ankara')
SET IDENTITY_INSERT [City] OFF

SET IDENTITY_INSERT [User] ON
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (1, 'John', 'Smith', 1, convert(datetime, '1982-03-24', 102), 5,
'john@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (2, 'Mike', 'Johnson', 1, convert(datetime, '1982-04-14', 102), 4,
'mike@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (3, 'Alice', 'Robertson', 0, convert(datetime, '1985-03-17', 102), 4,
'alice@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (4, 'Mary', 'Richards', 0, convert(datetime, '1983-08-21', 102), 5,
'mary@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (5, 'Jack', 'Dare', 1, convert(datetime, '1981-08-16', 102), 3,
'jack@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
```

```
[PasswordHash]) VALUES (7, 'Tommy', 'Anderson', 1, convert(datetime, '1989-07-11', 102), 3,
'tommy@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (9, 'Janet', 'Thomson', 0, convert(datetime, '1992-09-30', 102), 4,
'janet@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (10, 'Rick', 'Jackson', 1, convert(datetime, '1982-09-19', 102), 3,
'rick@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (12, 'Joanna', 'Rosenberg', 0, convert(datetime, '1993-05-13', 102), 5,
'joanna@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (14, 'Jill', 'Jordan', 0, convert(datetime, '1995-04-10', 102), 4,
'jill@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (15, 'Ivan', 'Petrov', 1, convert(datetime, '1987-03-22', 102), 6,
'ivan@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (16, 'Sergey', 'Ivanov', 1, convert(datetime, '1988-05-25', 102), 7,
'sergey@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (18, 'Petr', 'Sidorov', 1, convert(datetime, '1988-03-25', 102), 6,
'petr@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (19, 'Anna', 'Ivanova', 0, convert(datetime, '1987-04-21', 102), 7,
'anna@mail.com', '')
INSERT [User] ([ID], [FirstName], [LastName], [Sex], [BirthDate], [CityID], [Email],
[PasswordHash]) VALUES (20, 'Katerina', 'Petrova', 0, convert(datetime, '1981-09-12', 102), 7,
'katerina@mail.com', '')
SET IDENTITY_INSERT [User] OFF

INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (1, 1, 4, convert(datetime, '2010-06-
24', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (1, 4, 5, convert(datetime, '2010-07-
02', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (1, 7, 5, convert(datetime, '2010-07-
13', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (1, 14, 3, convert(datetime, '2010-08-
15', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (2, 3, 3, convert(datetime, '2010-09-
07', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (2, 6, 1, convert(datetime, '2010-10-
10', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (2, 18, 4, convert(datetime, '2010-11-
14', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (3, 4, 4, convert(datetime, '2010-12-
01', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (3, 11, 5, convert(datetime, '2010-12-
10', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (3, 12, 5, convert(datetime, '2011-01-
14', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (3, 18, 5, convert(datetime, '2011-03-
24', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (3, 19, 5, convert(datetime, '2011-04-
06', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (4, 5, 4, convert(datetime, '2011-05-
11', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (4, 7, 1, convert(datetime, '2011-06-
17', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (4, 13, 5, convert(datetime, '2011-09-
08', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (4, 16, 4, convert(datetime, '2011-10-
14', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (5, 3, 5, convert(datetime, '2012-03-
05', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (5, 8, 5, convert(datetime, '2012-05-
14', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (5, 14, 3, convert(datetime, '2012-06-
24', 102))
INSERT [Like] ([UserID], [SongID], [Score], [Date]) VALUES (7, 10, 5, convert(datetime, '2012-06-
25', 102))

INSERT [SongGenre] ([SongID], [GenreID]) VALUES (1, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (1, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (2, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (2, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (3, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (3, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (4, 1)
```

```
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (4, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (5, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (5, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (6, 3)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (7, 3)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (8, 3)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (9, 3)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (10, 3)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (11, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (11, 4)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (12, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (12, 4)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (13, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (13, 4)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (14, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (14, 4)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (15, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (15, 4)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (16, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (16, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (17, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (17, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (18, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (18, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (19, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (19, 2)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (20, 1)
INSERT [SongGenre] ([SongID], [GenreID]) VALUES (20, 2)

INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (1, 1)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (2, 1)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (3, 1)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (4, 1)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (5, 1)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (6, 2)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (7, 2)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (8, 2)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (9, 2)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (10, 2)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (11, 3)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (12, 3)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (13, 3)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (14, 3)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (15, 3)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (16, 4)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (17, 4)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (18, 4)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (19, 4)
INSERT [SongArtist] ([SongID], [ArtistID]) VALUES (20, 4)

SET IDENTITY_INSERT [Album] ON
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (1, 'Lonesome Crow', 1972, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (2, 'Fly to the Rainbow', 1974, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (3, 'In Trance', 1975, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (4, 'Virgin Killer', 1976, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (6, 'Taken by Force', 1977, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (7, 'Lovedrive', 1979, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (8, 'Animal Magnetism ', 1980, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (9, 'Blackout', 1982, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (10, 'Love at First Sting', 1984, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (11, 'Savage Amusement', 1988, 1)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (12, 'Please Please Me', 1963, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (13, 'With The Beatles', 1963, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (14, 'A Hard Day's Night', 1964, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (15, 'Beatles For Sale', 1964, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (16, 'Help!', 1965, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (17, 'Rubber Soul', 1965, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (18, 'Revolver', 1966, 2)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (19, 'Led Zeppelin', 1969, 3)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (20, 'Led Zeppelin II', 1969, 3)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (21, 'Led Zeppelin III', 1970, 3)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (24, 'Queen', 1973, 4)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (25, 'Queen II', 1974, 4)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (26, 'Black Sabbath', 1970, 5)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (27, 'Paranoid', 1970, 5)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (28, 'The Better Life', 2000, 6)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (29, 'Away from the Sun', 2002, 6)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (30, 'Seventeen Days', 2005, 6)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (31, 'The Offspring', 1989, 7)
```

```
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (32, 'Ignition', 1982, 7)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (34, 'Madonna', 1983, 9)
INSERT [Album] ([ID], [Title], [Year], [ArtistID]) VALUES (35, 'The Animals', 1964, 10)
SET IDENTITY_INSERT [Album] OFF

INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (1, 2, 3)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (2, 2, 7)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (3, 3, 1)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (4, 3, 2)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (5, 3, 3)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (6, 12, 1)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (7, 12, 2)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (8, 12, 3)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (9, 12, 4)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (10, 12, 5)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (11, 19, 1)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (12, 19, 2)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (13, 19, 3)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (14, 19, 4)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (15, 19, 5)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (16, 24, 1)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (17, 24, 2)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (18, 24, 3)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (19, 24, 4)
INSERT [SongAlbum] ([SongID], [AlbumID], [TrackNo]) VALUES (20, 24, 5)
```

Примеры вопросов по обязательной части

- Объяснить, что делают написанные запросы.
- В чем различие типов CHAR и VARCHAR? VARCHAR и NVARCHAR?
- Что такое внешний ключ?
- Какие существуют способы поддержания ссылочной целостности?
- Что такое уникальный ключ?
- Что такое нормализация?
- Рассказать о нормальных формах.
- Что такое IDENTITY?
- Рассказать о значениях по умолчанию и неопределенных значениях.
- Рассказать о вычисляемых столбцах.
- Как можно представить значение булевского типа?
- Как можно хранить даты и время?
- Рассказать о числовых типах данных.
- Каким образом можно вставить несколько строк с помощью одного оператора INSERT?
- Как ведет себя оператор INSERT, если в списке столбцов перечислены не все столбцы?

Примеры дополнительных вопросов

- Добавить какие-либо ограничения целостности.
- Добавить IDENTITY.
- Исправить выявленные при проверке недочеты.

Практическое задание №4. Операторы манипулирования данными языка SQL

Постановка задачи

Четвертое практическое задание посвящено манипулированию данными с помощью операторов SQL. В ходе выполнения четвертого практического задания необходимо:

- Составить SQL-скрипты для выполнения выборок, указанных в индивидуальном варианте. Кроме этого, нужно подготовить еще 3-4 выборки, которые имеют осмысленное значение для предметной области, и также составить для них SQL-скрипты.

- Сформулировать 3-4 запроса на изменение и удаление из базы данных. Запросы должны быть сформулированы в терминах предметной области. Среди запросов обязательно должны быть такие, которые будут вызывать срабатывание ограничений целостности. Составить SQL-скрипты для выполнения этих запросов.

Темы для проработки

- Оператор SELECT.
- Оператор UPDATE.
- Оператор DELETE.
- Декларативные ограничения целостности.

Примеры

Рассмотрим несколько примеров, показывающих различные способы построения запросов и некоторые полезные возможности MS SQL Server 2005:

Вывести имена и фамилии меломанов, которым нравятся только песни, которые не нравятся больше никому.

```
SELECT FirstName, LastName
      FROM [User] WHERE [User].ID IN
            (SELECT UserID FROM [Like]
                  EXCEPT
             SELECT UserID FROM [Like] WHERE SongID IN
                  (SELECT SongID
                        FROM [Like]
                        GROUP BY SongID
                        HAVING COUNT(DISTINCT UserID) > 1))
```

```
FirstName            LastName
-------------------- ----------------------
Tommy                Anderson
```

Определить самый популярный музыкальный жанр среди меломанов из Нью-Йорка.

```
SELECT TOP 1 WITH TIES
      Genre.[Name] AS Genre
      FROM Song
            LEFT JOIN [Like] ON Song.ID = [Like].SongID
            JOIN SongGenre ON Song.ID = SongGenre.SongID
            JOIN Genre ON Genre.ID = SongGenre.GenreID
```

```
      WHERE [Like].UserID IN
            (SELECT [User].ID
                  FROM [User] JOIN City ON [User].CityID = City.ID
                  WHERE City.[Name] = 'New York')
      GROUP BY Genre.[Name]
      ORDER BY ISNULL(SUM(Score), 0) DESC
```

```
Genre
---------------------------------------
Rock-n-Roll
```

Подсчитать, сколько людей из Америки слушают «The Beatles».

```
SELECT COUNT(DISTINCT [Like].UserID) AS [Count]
      FROM [Like] JOIN Song ON [Like].SongID = Song.ID
                  JOIN SongArtist ON Song.ID = SongArtist.SongID
                  JOIN Artist ON Artist.ID = SongArtist.ArtistID
      WHERE
            Artist.[Name] = 'The Beatles' AND
            [Like].UserID IN
                  (SELECT [User].ID
                        FROM [User]
                              JOIN City ON [User].CityID = City.ID
                              JOIN Country ON Country.ID = City.CountryID
                        WHERE Country.[Name] = 'USA')
```

```
Count
-----------
4
```

Определить название самой популярную песни жанра «Rock-n-Roll» в 2012 году.

```
WITH SongScores2012 AS
(SELECT Song.Title, SUM(Score) AS TotalScore
      FROM Song
            LEFT JOIN [Like] ON [Like].SongID = Song.ID
            JOIN SongGenre ON Song.ID = SongGenre.SongID
            JOIN Genre ON Genre.ID = SongGenre.GenreID
      WHERE YEAR([Like].Date) = 2012 AND Genre.[Name] = 'Rock-n-Roll'
      GROUP BY Song.ID, Song.Title)

SELECT Title
      FROM SongScores2012
      WHERE TotalScore = (SELECT MAX(TotalScore) FROM SongScores2012)
```

```
Title
--------------------------
Boys
Anna (Go to Him)
```

Приведем еще пример выборки, осмысленной для предметной области «Музыкальные предпочтения»: порекомендовать пользователю «john@mail.com» 3 самых популярных песни (с исполнителями) жанра «Heavy Metal», которые он еще не слышал. Подобный запрос, можно составить, например, таким образом:

```
SELECT TOP 3 WITH TIES
      (SELECT Artist.[Name] + '; '
            FROM Artist JOIN SongArtist ON Artist.ID = SongArtist.ArtistID
            WHERE SongArtist.SongID = S.ID FOR XML PATH('')) AS 'Artist(s)',
            S.[Title]
      FROM Song S LEFT JOIN [Like] ON [Like].SongID = S.ID
      WHERE EXISTS(
            SELECT * FROM SongGenre JOIN Genre ON SongGenre.GenreID = Genre.ID
```

```
            WHERE Genre.[Name] = 'Heavy Metal' AND SongGenre.SongID = S.ID
      )
      AND S.ID NOT IN
            (SELECT SongID FROM [Like]
                  JOIN [User] ON [Like].UserID = [User].ID
                  WHERE [User].Email = 'john@mail.com')
      GROUP BY S.[Title], S.[ID]
      ORDER BY ISNULL(SUM(Score), 0) DESC
```

```
Artist(s)                  Title
-------------------------- ------------------------------------------------------------
Queen;                     Great King Rat
The Scorpions;             Dark Lady
Queen;                     My Fairy King
```

Рассмотрим теперь примеры запросов для удаления и модификации данных. Например, удалим из таблицы предпочтений строки, набравшие в сумме меньше 3 очков:

```
DELETE FROM [Like]
      WHERE SongID IN
            (SELECT SongID FROM [Like]
                  GROUP BY SongID
                  HAVING SUM(Score) < 3
            )
```

Напишем теперь запрос на удаление данных, который не срабатывает из-за ограничений ссылочной целостности:

```
DELETE FROM Genre
      WHERE [Name] = 'Heavy Metal'
```

```
Msg 547, Level 16, State 0, Line 1
The DELETE statement conflicted with the REFERENCE constraint "FK_SongGenre_Genre". The conflict
occurred in database "MusicFans", table "dbo.SongGenre", column 'GenreID'.
The statement has been terminated.
```

Рассмотрим теперь модификацию данных. В качестве первого примера напишем запрос, который заменит все имена и фамилии пользователей на инициалы:

```
UPDATE [User] SET
   FirstName = SUBSTRING(FirstName, 1, 1),
   LastName = SUBSTRING(LastName, 1, 1)

SELECT ID, FirstName, LastName FROM [User]
```

```
ID          FirstName          LastName
----------- ------------------ --------------------
1           J                  S
2           M                  J
3           A                  R
4           M                  R
5           J                  D
7           T                  A
9           J                  T
10          R                  J
12          J                  R
14          J                  J
15          I                  P
16          S                  I
18          P                  S
19          A                  I
20          K                  P
```

Продемонстрировать работу проверочных ограничений можно, например, таким запросом:

```
UPDATE [Like] SET Score = 6 WHERE SongID = 1
```

```
Msg 547, Level 16, State 0, Line 1
The UPDATE statement conflicted with the CHECK constraint "CK_Like_Score". The conflict occurred
in database "MusicFans", table "dbo.Like", column 'Score'.
The statement has been terminated.
```

В качестве последнего примера рассмотрим нарушение ограничения уникального ключа:

```
UPDATE [User] SET Email = 'john@mail.com' WHERE ID = 2
```

```
Msg 2601, Level 14, State 1, Line 1
Cannot insert duplicate key row in object 'dbo.User' with unique index 'IX_User'.
The statement has been terminated.
```

Примеры вопросов по обязательной части

- Объяснить, как работают написанные запросы.
- Примеры вопросов по оператору SELECT см. в задании №1.

Примеры дополнительных вопросов

- Исправить неверно работающий запрос (запросы).
- Упростить один или несколько запросов.
- Написать или модифицировать запрос по сформулированному заданию.

Практическое задание №5. Создание и использование представлений

Постановка задачи

Пятое практическое задание посвящено созданию и использованию представлений: требуется составить SQL-скрипты для создания 4 представлений согласно индивидуальному варианту.

Темы для проработки

- Оператор CREATE VIEW.
- Вставка и модификация данных через представления.

Примеры

Создадим несколько полезных представлений для нашей базы данных. Пусть в первом представлении требуется показывать следующие данные: название песни, суммарная оценка, количество людей, которым нравится эта песня и средняя оценка.

```
IF EXISTS(
      SELECT * FROM sys.views
      WHERE [name] = 'SongScores' AND
            schema_id = SCHEMA_ID('dbo'))
DROP VIEW SongScores
GO
CREATE VIEW SongScores AS
(SELECT
      Song.Title,
      ISNULL(SUM([Like].Score), 0) AS TotalScore,
      COUNT([Like].UserID) AS Fans,
      ISNULL(AVG(1.0*[Like].Score), 0) AS AverageScore
      FROM Song
            LEFT JOIN [Like] ON Song.ID = [Like].SongID
      GROUP BY Song.ID, Song.Title)
GO
SELECT * FROM SongScores
```

```
Title                          TotalScore  Fans        AverageScore
------------------------------ ----------- ----------- ------------------
Drifting Sun                   4           1           4.000000
Fly to the Rainbow             0           0           0.000000
Dark Lady                      8           2           4.000000
In Trance                      9           2           4.500000
Life is Like A River           4           1           4.000000
I Saw Her Standing There       0           0           0.000000
Misery                         6           2           3.000000
Anna (Go to Him)               5           1           5.000000
Chains                         0           0           0.000000
Boys                           5           1           5.000000
Good Times, Bad Times          5           1           5.000000
Babe I'm Gonna Leave You       5           1           5.000000
You Shook Me                   5           1           5.000000
Dazed And Confused             6           2           3.000000
Your Time Is Gonna Come        0           0           0.000000
Keep Yourself Alive            4           1           4.000000
Doing Alright                  0           0           0.000000
Great King Rat                 9           2           4.500000
My Fairy King                  5           1           5.000000
Liar                           0           0           0.000000
```

В этом примере представление удаляется, если оно существовало, а затем создается, и производится выборка из него, чтобы проверить правильность данных.

Создадим теперь представление, содержащее название исполнителя, количество альбомов с его участием, количество песен с участием этого исполнителя.

```
IF EXISTS(
      SELECT * FROM sys.views
      WHERE [name] = 'Artists' AND
               schema_id = SCHEMA_ID('dbo'))
DROP VIEW Artists
GO
CREATE VIEW Artists AS
(SELECT
      Artist.[Name] AS Artist,
      COUNT(DISTINCT Album.ID) AS AlbumCount,
      COUNT(DISTINCT SongArtist.SongID) AS SongCount
      FROM Artist
            LEFT JOIN SongArtist ON Artist.ID = SongArtist.ArtistID
            LEFT JOIN SongAlbum ON SongArtist.SongID = SongAlbum.SongID
            LEFT JOIN Album ON Album.ID = SongAlbum.AlbumID
      GROUP BY Artist.[Name], Artist.ID)
GO
SELECT * FROM Artists
```

```
Artist                            AlbumCount  SongCount
--------------------------------- ----------- -----------
The Scorpions                     2           5
The Beatles                       1           5
Led Zeppelin                      1           5
Queen                             1           5
Black Sabbath                     0           0
3 Doors Down                      0           0
The Offspring                     0           0
Madonna                           0           0
The Animals                       0           0
Pink Floyd                        0           0
Deep Purple                       0           0
Jimi Hendrix                      0           0
The Who                           0           0
Aerosmith                         0           0
The Police                        0           0
Eagles                            0           0
System Of A Down                  0           0
The Misfits                       0           0
Elvis Presley                     0           0
```

В качестве следующего представления рассмотрим распределение активности меломанов по месяцам года: требуется вывести год, месяц, когда была какая-то активность и количество песен, которые понравились меломанам за этот месяц.

```
IF EXISTS(
      SELECT * FROM sys.views
      WHERE [name] = 'Activity' AND
               schema_id = SCHEMA_ID('dbo'))
DROP VIEW Activity
GO
CREATE VIEW Activity AS
(SELECT
      DATENAME(YYYY, [Like].Date) AS 'Year',
      DATENAME(MM, [Like].Date) AS 'Month', COUNT([Like].SongID) AS SongCount
      FROM [Like]
      GROUP BY DATENAME(YYYY, [Like].Date), DATENAME(MM, [Like].Date))
GO
SELECT * FROM Activity
```

```
Year                          Month                         SongCount
----------------------------- ----------------------------- -----------
2011                          April                         1
2010                          August                        1
```

```
2010                    December                2
2011                    January                 1
2010                    July                    2
2010                    June                    1
2011                    June                    1
2012                    June                    2
2011                    March                   1
2012                    March                   1
2011                    May                     1
2012                    May                     1
2010                    November                1
2010                    October                 1
2011                    October                 1
2010                    September               1
2011                    September               1
```

В качестве последнего представления рассмотрим следующее: для каждого пользователя вывести имя, фамилию, город и количество меломанов из этого города, которым нравится хотя бы одна песня из предпочтений этого пользователя.

```
IF EXISTS(
      SELECT * FROM sys.views
      WHERE [name] = 'Fans' AND
              schema_id = SCHEMA_ID('dbo'))
DROP VIEW Fans
GO
CREATE VIEW Fans AS
(SELECT FirstName,
            LastName,
            Country.[Name] AS Country,
            City.[Name] AS City,
            (SELECT COUNT(DISTINCT UserID)
                  FROM [Like]
                  WHERE
                  SongID IN
                        (SELECT SongID FROM [Like] WHERE [Like].UserID = U.ID)
                  AND [Like].UserID <> U.ID
            ) AS Fans
      FROM [User] U
            JOIN City ON U.CityID = City.ID
            JOIN Country ON City.CountryID = Country.ID)
GO
SELECT * FROM Fans
```

```
FirstName           LastName            Country             City                Fans
------------------- ------------------- ------------------- ------------------- -----------
John                Smith               USA                 San-Francisco       3
Mike                Johnson             USA                 Washington          2
Alice               Robertson           USA                 Washington          2
Mary                Richards            USA                 San-Francisco       1
Jack                Dare                USA                 New York            2
Tommy               Anderson            USA                 New York            0
Janet               Thomson             USA                 Washington          0
Rick                Jackson             USA                 New York            0
Joanna              Rosenberg           USA                 San-Francisco       0
Jill                Jordan              USA                 Washington          0
Ivan                Petrov              Russia              Moscow              0
Sergey              Ivanov              Russia              St. Petersburg      0
Petr                Sidorov             Russia              Moscow              0
Anna                Ivanova             Russia              St. Petersburg      0
Katerina            Petrova             Russia              St. Petersburg      0
```

Примеры вопросов по обязательной части

- Объяснить, как работают написанные запросы.
- Рассказать о CHECK OPTION.
- Рассказать о модификации данных через представления.

- Рассказать о вставке данных через представления.
- Примеры вопросов по оператору SELECT см. в задании №1.

Примеры дополнительных вопросов

- Исправить неверно работающий запрос (запросы).
- Упростить один или несколько запросов.
- Продемонстрировать изменение и вставку данных через представления.
- Написать или модифицировать запрос по сформулированному заданию.

Практическое задание №6. Управление транзакциями в MS SQL Server

Постановка задачи

Шестое практическое задание посвящено управлению транзакциями в MS SQL Server: необходимо подготовить SQL-скрипты для проверки наличия аномалий (потерянных изменений, грязных чтений, неповторяющихся чтений, фантомов) при параллельном исполнении транзакций на различных уровнях изолированности SQL/92 (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE). Подготовленные скрипты должны работать с одной из таблиц, созданных в практическом задании №3.

Для проверки наличия аномалий потребуются два параллельных сеанса, операторы в которых выполняются пошагово.

- Установить в обоих сеансах уровень изоляции READ UNCOMMITTED. Выполнить сценарии проверки наличия аномалий потерянных изменений и грязных чтений.

- Установить в обоих сеансах уровень изоляции READ COMMITTED. Выполнить сценарии проверки наличия аномалий грязных чтений и неповторяющихся чтений.

- Установить в обоих сеансах уровень изоляции REPEATABLE READ. Выполнить сценарии проверки наличия аномалий неповторяющихся чтений и фантомов.

- Установить в обоих сеансах уровень изоляции SERIALIZABLE. Выполнить сценарий проверки наличия фантомов.

Темы для проработки

- Понятие транзакции, свойства транзакций.
- Аномалии доступа к БД: потерянные изменения, грязные чтения, неповторяющиеся чтения, фантомы.
- Уровни изолированности SQL/92: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE.
- Управление транзакциями в MS SQL Server.

Примеры

В качестве примера исследуем наличие аномалии грязного чтения на уровне READ UNCOMMITTED. Установим уровень изолированности READ UNCOMMITTED и покажем, что в этом случае допускаются грязные чтения:

№	Сеанс 1	Сеанс 2
1	`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`	`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`
2	`BEGIN TRANSACTION`	`BEGIN TRANSACTION`

3	`SELECT [Year] FROM Album WHERE ID = 1`
	
	`Year`
	`-----------`
	`1972`	

4		`UPDATE Album SET [Year] = 1974` `WHERE ID = 1`

5	`SELECT [Year] FROM Album WHERE ID = 1`
	
	`Year`
	`-----------`
	`1974`	

6		`ROLLBACK`

7	`SELECT [Year] FROM Album WHERE ID = 1`
	
	`Year`
	`-----------`
	`1972`	

8	`COMMIT`	

Из этого примера видно, что уровень изолированности READ UNCOMMITTED допускает грязные чтения. Остальные сценарии предлагается проверить самостоятельно.

Примеры вопросов по обязательной части

- Рассказать об аномалиях доступа к БД.
- Перечислить аномалии, возникающие на каждом из уровней изолированности.
- Рассказать о свойствах транзакций.
- Рассказать об управлении транзакциями.
- Что такое тупики? Как бороться с тупиками?
- На каком уровне изолированности возможны тупики?
- Как обеспечивается изолированность транзакций в СУБД?
- Как бороться с проблемой фантомов?
- Что такое гранулированные блокировки?
- Как можно избежать блокировки при конфликте Read-Write?
- Что такое журнал транзакций?
- Как обеспечивается постоянство хранения (durability) в СУБД?

Примеры дополнительных вопросов

- Продемонстрировать откат транзакции при возникновении ошибок.
- Продемонстрировать возникновение тупика.
- Исправить неверные сценарии проверки аномалий.

Практическое задание №7. Управление доступом в MS SQL Server

Постановка задачи

Целью седьмого практического задания является освоение способов управления доступом в Microsoft SQL Server. При выполнении задания необходимо:

- Выдать пользователю test\test доступ к базе данных (назначить ему роль уровня базы данных public).

- Составить и выполнить скрипты присвоения новому пользователю прав доступа к таблицам, созданным в практическом задании №3. При этом права доступа к различным таблицам должны быть различными, а именно:

    - По крайней мере, для одной таблицы новому пользователю присваиваются права SELECT, INSERT, UPDATE в полном объеме.

    - По крайней мере, для одной таблицы новому пользователю присваиваются права SELECT и UPDATE только избранных столбцов.

    - По крайней мере, для одной таблицы новому пользователю присваивается только право SELECT.

- Присвоить новому пользователю право доступа (SELECT) к представлению, созданному в практическом задании №5.

- Создать стандартную роль уровня базы данных, присвоить ей право доступа (UPDATE на некоторые столбцы) к представлению, созданному в практическом задании №5, назначить новому пользователю созданную роль.

- Выполнить от имени нового пользователя некоторые выборки из таблиц и представления, подготовленные в практических заданиях №4 и №5. Убедиться в правильности контроля прав доступа.

- Выполнить от имени нового пользователя операторы изменения таблиц с ограниченными правами доступа. Убедиться в правильности контроля прав доступа.

Темы для проработки

- Средства управления доступом в MS SQL Server.
- Аутентификация и авторизация.
- Роли и привилегии.
- Директивы GRANT, DENY и REVOKE.

Примеры вопросов по обязательной части

- В чем различия имени входа (логина) и пользователя?

- Рассказать о ролях уровня сервера.
- Рассказать о ролях уровня базы данных.
- Можно ли создать свою роль уровня сервера?
- Для чего нужны роли?
- Что такое схема?
- Рассказать о роли уровня базы данных public.
- Рассказать про директивы GRANT, DENY и REVOKE.
- Как разрешить пользователю предоставлять разрешение другим пользователям?
- Как добавить нового пользователя в текущую базу данных?
- Как создать новый логин?

Примеры дополнительных вопросов

- Исправить ошибки в обязательной части.
- Сменить владельца базы данных.
- Сменить пароль для имени входа.
- Сменить базу данных по умолчанию для имени входа.
- Определить роль с заданными правами.

Практическое задание №8. Использование метаданных о структуре БД

Постановка задачи

Восьмое практическое задание посвящено выборке метаданных о структуре базы данных. Необходимо составить следующие запросы на выборку сведений о таблицах, представлениях, триггерах, созданных в процессе выполнения практических заданий:

- Выбрать имена всех таблиц, созданных назначенным пользователем базы данных.

- Выбрать имя таблицы, имя столбца таблицы, признак того, допускает ли данный столбец NULL-значения, название типа данных столбца таблицы, размер этого типа данных - для всех таблиц, созданных назначенным пользователем базы данных и всех их столбцов.

- Выбрать название ограничения целостности (первичные и внешние ключи), имя таблицы, в которой оно находится, признак того, что это за ограничение ('PK' для первичного ключа и 'F' для внешнего) - для всех ограничений целостности, созданных назначенным пользователем базы данных.

- Выбрать название внешнего ключа, имя таблицы, содержащей внешний ключ, имя таблицы, содержащей его родительский ключ - для всех внешних ключей, созданных назначенным пользователем базы данных.

- Выбрать название представления, SQL-запрос, создающий это представление - для всех представлений, созданных назначенным пользователем базы данных.

- Выбрать название триггера, имя таблицы, для которой определен триггер - для всех триггеров, созданных назначенным пользователем базы данных.

Темы для проработки

- Структура системных представлений в Microsoft SQL Server.

Примеры

В качестве примера выведем список всем ограничений целостности типа CHECK для нашей базы данных в следующем формате: название таблицы, название ограничения, проверяемое условие:

```
SELECT
      so2.[name] AS 'Table',
      so1.[name] AS 'Constraint',
      sc.[text] AS 'Condition'
FROM
      sys.sysobjects so1
            JOIN sys.sysobjects so2 ON so1.parent_obj = so2.id
            JOIN sys.syscomments sc ON so1.id = sc.id
WHERE so1.xtype = 'C'
```

```
Table         Name                  Condition
------------- --------------------- ------------------------------------------------------------
Like          CK_Like_Score         ([Score]>(0) AND [Score]<(6))
SongAlbum     CK_SongAlbum_TrackNo  ([TrackNo]>(0))
Album         CK_Album_Year         ([Year]>(1900) AND [Year]<=datepart(year,getdate()))
User          CK_User_BirthDate     ((datepart(year,getdate())-datepart(year,[BirthDate]))>=(7))
```

Примеры вопросов по обязательной части

- В чем разница между sys и INFORMATION_SCHEMA?
- Как обеспечить максимальную переносимость скриптов, использующих метаданные, на другие SQL-ориентированные СУБД?

Примеры дополнительных вопросов

- Исправить ошибки в подготовленных выборках.
- Выбрать имена баз данных и файлы, соответствующие им на диске.
- Выбрать названия ролей и имена пользователей, входящих в эти роли.
- Вывести текст запроса, создающего какое-либо системное представление.
- Составить различные другие выборки по заданию преподавателей.

Практическое задание №9. Создание и использование триггеров

Постановка задачи

Девятое практическое задание посвящено созданию и использованию триггеров для поддержания целостности данных. Необходимо составить скрипт для создания триггера согласно индивидуальному варианту, а также подготовить несколько запросов для проверки и демонстрации работы триггера.

Темы для проработки

- Функции триггеров.
- Типы триггеров и условия их срабатывания.
- Создание и использование триггеров в MS SQL Server.

Примеры

В качестве примера реализуем следующее ограничение целостности, имеющее смысл для предметной области «музыкальные предпочтения»: пользователь может оценивать не более 5 песен в сутки. Создать такой триггер можно, например, следующим образом:

```
CREATE TRIGGER LimitLikes
  ON [Like] AFTER INSERT AS
    IF EXISTS(
      SELECT [Like].UserID
        FROM [Like]
        WHERE
          [Like].UserID IN (SELECT UserID FROM inserted) AND
            CONVERT(VARCHAR, [Like].[Date], 105) IN
              (SELECT CONVERT(VARCHAR, inserted.[Date], 105)
           FROM inserted
           WHERE inserted.UserID = [Like].UserID)
        GROUP BY [Like].UserID, CONVERT(VARCHAR, [Like].[Date], 105)
        HAVING COUNT([Like].SongID) > 5)
    BEGIN
      ROLLBACK
    END
```

Этот триггер будет срабатывать после вставки записей в таблицу Like (AFTER INSERT) и не позволит добавлять новые строки, если лимит в 5 оценок в сутки превышен. Проверим теперь работу триггера:

```
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 1, 5)
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 2, 5)
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 3, 5)
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 4, 5)
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 5, 5)

-- Шестая оценка превышает лимит
INSERT [Like] ([UserID], [SongID], [Score]) VALUES (10, 6, 5)

Msg 3609, Level 16, State 1, Line 7
The transaction ended in the trigger. The batch has been aborted.
```

Примеры вопросов по обязательной части

- Объяснить принцип работы написанного триггера.
- Какие бывают типы триггеров?
- Когда может срабатывать триггер?
- В каком порядке срабатывают триггеры?
- Можно ли менять порядок срабатывания триггеров?
- Сработает ли триггер, если оператор, выполненный пользователем, не затрагивает ни одну строку таблицы?

Примеры дополнительных вопросов

- Исправить ошибки в работе триггера.
- Модифицировать триггер каким-либо образом.

Практическое задание №10. Использование индексов и средств оптимизации запросов в MS SQL Server

Постановка задачи

Десятое практическое задание посвящено ускорению выполнения запросов при помощи использования индексов. Для выполнения задания необходим достаточно большой объем данных, чтобы использование индексов было целесообразным (порядка 1000000 строк в каждой таблице). Необходимо заполнить соответствующие таблицы и подготовить два запроса:

- Запрос к одной таблице, содержащий фильтрацию по нескольким полям.
- Запрос к нескольким связанным таблицам, содержащий фильтрацию по нескольким полям.

Для каждого из этих запросов необходимо провести следующие шаги:

- Получить план выполнения запроса без использования индексов.
- Получить статистику (IO и Time) выполнения запроса без использования индексов.
- Создать нужные индексы, позволяющие ускорить запрос.
- Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.
- Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.
- Оценить эффективность выполнения оптимизированного запроса.

Темы для проработки

- Понятие индекса, типы индексов, внутренняя организация индексов.
- Работа с индексами в среде MS SQL Server Management Studio.
- Использование индексов для повышения скорости выполнения запросов.

Примеры

Рассмотрим простейший пример использования индексов. Для того чтобы использование индексов было целесообразным, необходимо подготовить достаточно большое количество данных. Создадим таблицы **UserLarge** и **UserLargeIndex** для этих целей:

```
SELECT * INTO UserLarge FROM [User]

SELECT * INTO UserLargeIndex FROM [User]
```

Далее, увеличим количество данных в этих таблицах (может занять длительное время):

```
INSERT INTO [UserLarge]
      SELECT FirstName, LastName, Sex, BirthDate, CityID, Email, PasswordHash
            FROM [User]

INSERT INTO [UserLargeIndex]
      SELECT FirstName, LastName, Sex, BirthDate, CityID, Email, PasswordHash
            FROM [User]

GO 100000
```

Рассмотрим теперь следующий простой запрос:

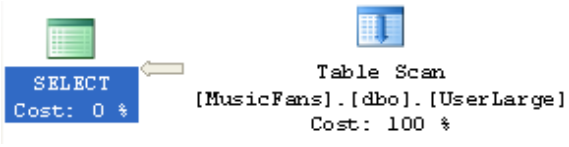
```
SELECT FirstName, LastName
      FROM [UserLarge]
            WHERE [FirstName] = 'John' AND [LastName] = 'Smith'
```

Не создавая пока никаких индексов, оценим производительность и план выполнения запроса. Для того, чтобы оценить время выполнения и количество операций ввода-вывода, включим вывод статистики:

```
SET STATISTICS TIME ON
SET STATISTICS IO ON
```

Также необходимо включить отображение плана выполнения запроса. Запрос без использования индексов дает следующие показатели и план выполнения:

```
Table 'UserLarge'. Scan count 1, logical reads 11113, physical reads 0, read-ahead reads 11113,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 188 ms,  elapsed time = 6156 ms.
```



Стоит заметить, что при оценке производительности запросов бывает полезным очищать кэш SQL Server:

```
CHECKPOINT
GO
DBCC DROPCLEANBUFFERS
GO
```

Теперь создадим индекс в таблице **UserLargeIndex** и сравним стоимость выполнения одного и того же запроса к таблицам **UserLarge** (без индексов) и **UserLargeIndex** (с индексом):

```
CREATE NONCLUSTERED INDEX [IX_FirstNameLastName] ON [dbo].[UserLargeIndex]
(
      [FirstName] ASC,
      [LastName] ASC,
)
```

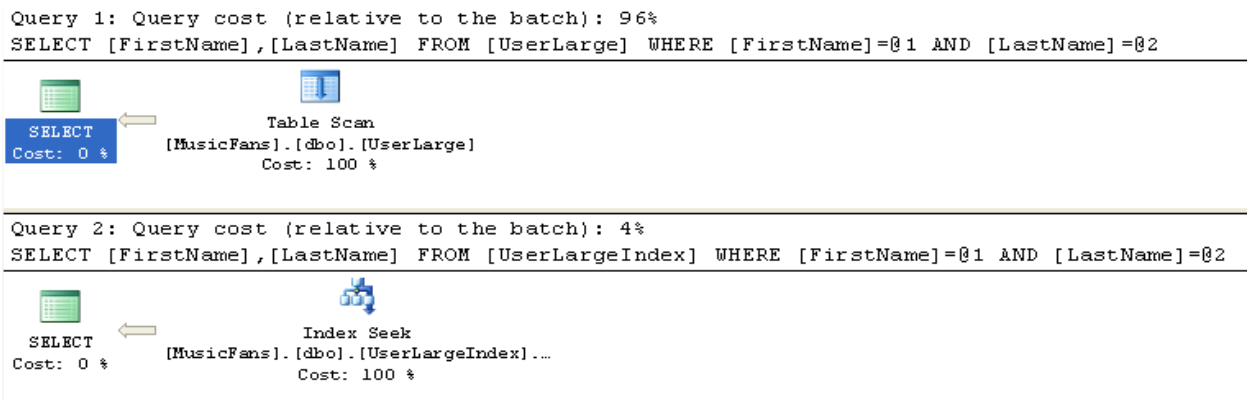
Выполним запрос к таблице **UserLargeIndex** и оценим статистику:

```
SELECT FirstName, LastName
      FROM [UserLargeIndex]
            WHERE [FirstName] = 'John' AND [LastName] = 'Smith'

Table 'UserLargeIndex'. Scan count 1, logical reads 327, physical reads 3, read-ahead reads 324,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.


SQL Server Execution Times:
     CPU time = 47 ms,  elapsed time = 2703 ms.
```

Сравним теперь стоимость запросов к таблицам **UserLarge** и **UserLargeIndex**:

```
SELECT FirstName, LastName
      FROM [UserLarge]
```

```
            WHERE [FirstName] = 'John' AND [LastName] = 'Smith'

SELECT FirstName, LastName
      FROM [UserLargeIndex]
            WHERE [FirstName] = 'John' AND [LastName] = 'Smith'
GO
```



Из этого примера видно, что индекс позволяет существенно ускорить выполнение запросов, позволяя избежать дорогостоящих операций сканирования таблицы.

Примеры вопросов по обязательной части

- Что такое кластерный индекс? В чем его отличие от некластерного?
- Можно ли создать неуникальный кластерный индекс?
- В чем отличие первичного ключа и уникального индекса?
- В каких случаях имеет смысл создавать индексы? Какие колонки следует включать в индекс и почему?
- Какие существуют способы внутренней организации индексов?
- Рассказать о проблеме фрагментации индексов. Как бороться с фрагментацией?
- Имеет ли значение порядок указания колонок при создании индекса?
- В чем разница между Index Scan и Index Seek?

Примеры дополнительных вопросов

- Исправить ошибки в подготовленных выборках.
- Могут ли индексы ухудшить производительность? Если да, то продемонстрировать это.
- На что влияет порядок сортировки (ASC\DESC) при создании индекса? Продемонстрировать это.
- Объяснить и продемонстрировать разницу в скорости выполнения запросов в зависимости от того, входят ли все колонки из списка выборки в некластерный индекс.

Литература и материалы

1. Кузнецов С. Д. Основы баз данных. – М.: Бином. Лаборатория знаний, Интернет-университет информационных технологий. 2007
2. Microsoft SQL Server 2005 Books Online. [HTML] http://msdn.microsoft.com/en-us/library/ms130214%28v=sql.90%29.aspx
3. Кузнецов С. Д. Основы современных баз данных. [HTML] http://citforum.ru/database/osbd/contents.shtml
4. Гарсиа-Молина Г., Ульман Дж., Уидом Д. Системы баз данных Полный курс. – М.: Вильямс, 2004
5. Ульман Дж., Уидом Д. Основы реляционных баз данных. – М.: Лори. 2006.
6. Коннолли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение. Теория и практика. – М.: Вильямс. 2003.
7. Станек У. Р. Microsoft SQL Server 2005. Справочник администратора – М.: Русская редакция. 2008.
8. Нильсен П. SQL Server 2005. Библия пользователя. – М.: Вильямс. 2008

Web-ресурсы

1. http://spprac.cs.msu.ru/
2. http://bdis.umeta.ru/db/