Практическое задание 10

PostgreSQL

Разработка объектов базы данных

Цель. Разработка объектов базы данных на основе анализа предметной области, а также связей между ними.

Задачи. Освоить способы создания первичного ключа; суррогатного ключа, с помощью последовательностей; отработать ввод данных, создание связей и индексов, изменение структуры таблицы с контрольными ограничениями, создание обычных и неизменяемых представлений.

Описание предметной области.

Для магазина, торгующего картинами, разрабатывается база данных.

После системного анализа предметной области, была разработана инфологическая (ER) модель, на основании которой была создана даталогическая (реляционная) модель, содержащая пять отношений:

- таблица CUSTOMER содержит данные о покупателях;
- таблица ARTIST содержит данные о художниках;
- таблица WORK содержит данные о картинах;
- таблица TRANSACTION содержит данные о сделках;
- таблица CUSTOMER_ARTIST_INT содержит данные о предпочтениях покупателя к определенному художнику.

Последовательность выполнения.

Создать таблицы **CUSTOMER, ARTIST и CUSTOMER_ARTIST_INT**

```
test=> CREATE TABLE CUSTOMER(
test(> CustomerID int NOT NULL,
test(> Name varchar(25) NOT NULL,
test(> Street varchar(30) NULL,
test(> City varchar(35) NULL,
test(> State varchar(10) NULL,
test(> Zip varchar(10) NULL,
test(> Area_Code varchar(10) NULL,
test(> Phone_Number varchar(10) NULL);
CREATE TABLE
test=>
test=> ALTER TABLE CUSTOMER
test-> ADD CONSTRAINT CustomerPK PRIMARY KEY (CustomerID);
ALTER TABLE
test=>
test=> CREATE INDEX CustomerNameIndex ON CUSTOMER(Name);
CREATE INDEX
test=>
test=> CREATE TABLE ARTIST(
test(> ArtistID int PRIMARY KEY,
test(> Name varchar(25) NOT NULL,
test(> Nationality varchar(30) NULL,
test(> Birthdate date NULL,
test(> DeceasedDate date NULL);
CREATE TABLE
test=>
test=> CREATE INDEX ArtistNameIndex ON ARTIST(Name);
CREATE INDEX
test=>
test=> CREATE TABLE CUSTOMER_ARTIST_INT(
test(> ArtistID int NOT NULL,
test(> CustomerID int NOT NULL);
CREATE TABLE
test=>
test=> ALTER TABLE CUSTOMER_ARTIST_INT
test-> ADD CONSTRAINT CustomerArtistPK PRIMARY KEY ( ArtistID,
test(> CustomerID );
ALTER TABLE
```

Обратите внимание на два способа задания первичного ключа для таблицы – внутри оператора ее создания и с помощью оператора изменения таблицы. Обратитесь к документации по СУБД PostgreSQL и в отчете поясните особенности применения каждого из указанных способов.

Важной особенностью задания ключа для таблицы является использование уникальности комбинации нескольких ее атрибутов – композитного ключа. Какой из двух вышеописанных способов подходит для создания композитного ключа? Поясните ваш ответ в отчете.

Кроме создания таблиц, вы создаете еще два индекса, содержащих имена художников и покупателей. Обратитесь к документации по адресу postgrespro.ru/docs/postgresql/15/sql-createindex для ознакомления с теоретическим материалом, связанным с этим инструментом СУБД. Отразите основные мысли по поводу создания этих индексов в отчете.

Вы имеете возможность создать текстовый файл с расширением *.sql и поместить туда целиком весь скрипт, связанный с созданием базы

данных. Это удобно с точки зрения отработки навыков программирования баз данных и позволяет сохранить скрипт для будущих изменений. Команды консоли, позволяющие считать скрипт из файла на диске: \i [файл и путь к нему]. Кроме того, у вас есть возможность выводить результаты всех запросов в файл, с помощью следующей команды: \o [файл и путь к нему].

Потренируйтесь работать с консольными командами: \d, \d+, \d [таблица] и \d+

[таблица]. Отобразите структуру созданных трех таблиц на экране и прокомментируйте результат в отчете.

Создание суррогатных ключей с помощью последовательностей

Используйте последовательности для создания уникальных увеличивающихся значений. Соответствующие возможности отлично подходят для формирования суррогатных ключей.

```
test=> Create Sequence CustID start 100;
CREATE SEQUENCE
```

СУБД содержит возможность управлять значениями последовательности с помощью методов NextVal (выдает следующее значение в последовательности) и CurrVal (выдает текущее значение в последовательности).

```
test=> INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number)
test-> VALUES (NextVal('CustID'), 'Mary Jones', '350', '555-1234');
INSERT 0 1
```

Использованный оператор создаст в таблице CUSTOMER строку, где столбцу CustomerID будет присвоено следующее значение в последовательности CustID. Выполнив этот оператор, можно считать только что созданную строку с помощью метода CurrVaL:

```
test=> select * from customer where customerid=currval('custid');
 customerid |       name       | street | city | state | zip | area_code | phone_number
------------+------------------+--------+------+-------+-----+-----------+--------------
        103 | Tiffany Twilight |        |      |       |     | 360       | 555-1040
(1 строка)
```

Здесь метод CustID.CurrVal возвращает текущее значение в последовательности, то есть только что использованное значение.

Использование последовательностей не гарантирует корректности значений суррогатных ключей (могут быть пропущенные или повторяющиеся значения). Создайте с помощью SQL Plus следующие последовательности:

```
test=> Create Sequence ArtistID start 1;
CREATE SEQUENCE
test=> Create Sequence WorkID start 500;
CREATE SEQUENCE
test=> Create Sequence TransID start 100;
CREATE SEQUENCE
```

Ввод данных

Заполните таблицы Artist и Customer значениями:

```
test=> INSERT INTO ARTIST (ArtistID, Name, Nationality) Values (NextVal('ArtistID'), 'Tobey', 'US');
INSERT 0 1
test=> INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
test-> (NextVal('ArtistID'), 'Miro', 'Spanish');
INSERT 0 1
test=> INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
test-> (NextVal('ArtistID'), 'Frings', 'US');
INSERT 0 1
test=> INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
test-> (NextVal('ArtistID'), 'Foster', 'English');
INSERT 0 1
test=> INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
test-> (NextVal('ArtistID'), 'van Vronkin', 'US');
INSERT 0 1
test=> INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
test-> (NextVal('CustID'), 'Jeffrey Janes', '206', '555-1234');
INSERT 0 1
test=> INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
test-> (NextVal('CustID'), 'David Smith', '206', '555-443');
INSERT 0 1
test=> INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
test-> (NextVal('CustID'), 'Tiffany Twilight', '360', '555-1040');
INSERT 0 1
```

Отобразите на экране столбцы ArtistID, Name, Nationality из таблицы ARTIST; CustomerID, Name, AreaCode, PhoneNumber из таблицы CUSTOMER с помощью соответствующих запросов.

Создание связей

В PostgreSQL связи между отношениями создаются путем введения ограничений целостности по внешнему ключу. Например, следующие SQLоператоры определяют связь между таблицами CUSTOMER и CUSTOMER_ARTIST_INT и между таблицами ARTIST и

```
test=> ALTER TABLE CUSTOMER_ARTIST_INT ADD CONSTRAINT ArtistIntFK
test-> FOREIGN KEY(ArtistID) REFERENCES ARTIST ON DELETE CASCADE;
ALTER TABLE
test=> ALTER TABLE CUSTOMER_ARTIST_INT ADD CONSTRAINT CustomerIntFK
test-> FOREIGN KEY(CustomerID) REFERENCES CUSTOMER ON DELETE CASCADE;
ALTER TABLE
```

Ограничениям даны имена ArtistIntFK и CustomerIntFK. Эти имена не играют особой роли для PostgreSQL и могут выбираться разработчиком. Обратите внимание, что для родительской таблицы указан только столбец, являющийся внешним ключом. СУБД предполагает, что внешний ключ будет связан с первичным ключом родительской таблицы, поэтому указывать столбец первичного ключа нет необходимости. Фраза ON DELETE CASCADE указывает на то, что при удалении строк из родительской таблицы соответствующие строки дочерних таблиц должны быть также удалены. Слово *cascade* (каскад) используется здесь потому, что удаление идет каскадом от родительской таблицы к дочерней.

Введите эти операторы в редактор SQL Shell (psql) и заполните несколько строк таблицы пересечения. Теперь, если вы удалите данные о покупателе или художнике, соответствующие строки в таблице пересечения будут также удалены.

Создайте таблицы WORK и TRANSACTION. Обратите внимание, что в определениях ограничений по внешнему ключу отсутствует фраза ON DELETE CASCADE. Так, ограничение ArtistFK сделает невозможным удаление тех строк в таблице ARTIST, которые имеют дочерние строки и таблице WORK.

Ограничения WorkFK и CustomerFK функционируют сходным образом.

```
test=> CREATE TABLE WORK (
test(> WorkID int PRIMARY KEY,
test(> Description varchar(1000) NULL,
test(> Title varchar(25) NOT NULL,
test(> Copy varchar(8) NOT NULL,
test(> ArtistID int NOT NULL);
CREATE TABLE
test=>
test=> ALTER TABLE WORK ADD CONSTRAINT ArtistFK
test-> FOREIGN KEY (ArtistID) REFERENCES ARTIST;
ALTER TABLE
test=>
test=> CREATE TABLE TRANSACTION (
test(> TransactionID int PRIMARY KEY,
test(> DateAcquired date NOT NULL,
test(> AcquisitionPrice numeric(7,2) NULL,
test(> PurchaseDate date NULL,
test(> SalesPrice numeric(7,2) NULL,
test(> CustomerID int NULL,
test(> WorkID int NOT NULL);
CREATE TABLE
test=>
test=> ALTER TABLE TRANSACTION ADD CONSTRAINT WorkFK
test-> FOREIGN KEY (WorkID) REFERENCES WORK;
ALTER TABLE
test=>
test=> ALTER TABLE TRANSACTION ADD CONSTRAINT CustomerFK
test-> FOREIGN KEY (CustomerID) REFERENCES CUSTOMER;
ALTER TABLE
```

Оператор ALTER можно также использовать для удаления ограничения.

Создание индексов

Создайте индекс по столбцу Name таблицы CUSTOMER. Индексу дано имя CustNameIdx. Имя не играет роли для PostgreSQL. Чтобы создать уникальный индекс, перед ключевым словом INDEX используют ключевое слово UNIQUE.

Например, чтобы гарантировать, что ни одно произведение не будет записано дважды в таблицу WORK, можно создать уникальный индекс по столбцам

```
test=> CREATE INDEX CustNameIdx ON CUSTOMER(Name);
CREATE INDEX
test=>
test=> CREATE UNIQUE INDEX WorkUniqueIndex on Work (Title, Copy, ArtistID);
CREATE INDEX
```

Изменение структуры таблиц, контрольные ограничения

После создания таблицы ее структуру можно изменять с помощью оператора ALTER TABLE. Будьте, однако, осторожны с этим оператором, поскольку при его использовании возможна потеря данных.

Модифицируем таблицу ARTIST. Мы установили для столбцов

(дата рождения) и DeceasedDate (дата смерти) тип данных Date. Допустим, что пользователям базы данных не нужно, чтобы и этих столбцах хранилась полная дата, а нужен только год рождения или смерти художника. Предположим также, что из представленных в галерее художников нет ни одного, кто бы родился или умер ранее 1400 года или позже 2100 года.

Пока эти столбцы имеют пустые значения, что позволяет нам менять тип данных, не удаляя сами столбцы.

```
test=> ALTER TABLE ARTIST
test-> ALTER COLUMN BirthDate DROP DEFAULT,
test-> ALTER COLUMN BirthDate TYPE Numeric(4) USING date_part('Year', birthdate)::numeric(4,0);
ALTER TABLE
test=>
test=>
test=> ALTER TABLE ARTIST
test-> ALTER COLUMN DeceasedDate DROP DEFAULT,
test-> ALTER COLUMN DeceasedDate TYPE Numeric(4) USING date_part('Year', DeceasedDate)::numeric(4,0);
ALTER TABLE
```

Следующие два оператора устанавливают пределы значений столбцов BirthDate и DeceasedDate:

```
test=> ALTER TABLE ARTIST ADD CONSTRAINT BDLimit CHECK (BirthDate BETWEEN 1400 AND 2100);
ALTER TABLE
test=> ALTER TABLE ARTIST ADD CONSTRAINT DDLimit CHECK (DeceasedDate BETWEEN 1400 AND 2100);
ALTER TABLE
```

Выполним команды обновления:

```
test=> UPDATE ARTIST SET BirthDate = 1870 WHERE Name = 'Miro';
UPDATE 1
test=> UPDATE ARTIST SET BirthDate = 1270 WHERE Name = 'Tobey';
ОШИБКА:  новая строка в отношении "artist" нарушает ограничение-проверку "bdlimit"
DETAIL:  Ошибочная строка содержит (1, Tobey, US, 1270, null).
```

Первое обновление пройдет успешно, а второе нарушит ограничение и поэтому не будет выполнено. Попробуйте самостоятельно запустить эти операторы и посмотрите, каковы будут результаты.

Представления

Важное ограничение SQL-представлений состоит в том, что они могут содержать не более одного многозначного пути.

Определим представление, соединяющее три таблицы, с наложенным условием на столбцы AcquisitionPrice и CustomerID. Это *представления соединения* – они базируются на соединениях.

```
test=> CREATE VIEW ExpensiveArt AS
test-> SELECT Name, Copy, Title
test-> FROM ARTIST, WORK, TRANSACTION
test-> WHERE ARTIST.ArtistID = WORK.ArtistID AND
test-> WORK.WorkID = TRANSACTION.WorkID AND
test-> AcquisitionPrice > 10000 AND
test-> CustomerID IS NULL;
CREATE VIEW
test=> CREATE materialized VIEW V2 AS SELECT * FROM ARTIST;
SELECT 5
```

Вообще говоря, простые представления допускают обновление данных. Если вам необходимо зафиксировать содержимое представления до момента, когда вы сами захотите его обновить, то вы можете создать materialized представление. Оно хранит не только данные, но и запрос, по которым было сформировано, и в нужный вам момент вы сможете его обновить.

Для обновления данных в простом представлении иногда можно использовать SQL-оператор UPDATE, однако в общем случае оператор UPDATE не может использоваться. Для этих целей потребуется написать специальный триггер, называемый замещающим триггером (триггер INSTEAD OF).

Контрольные задания к л.р. № 3-4

1. Вывести на экран информацию о таблицах пользователя
2. Вывести на экран информацию о содержимом одной из таблиц
3. Вывести на экран информацию о физических атрибутах таблицы Artist Ввести данные для следующих ситуаций:
4. На склад поступили репродукции картин 2-х русских художников. Как минимум одно произведение - первого

художника и два произведения - второго *(требуются ваши уникальные данные о реальных художниках, повторы в группе недопустимы).*

5. Сначала появился первый покупатель и приобрел картину первого художника. Затем появился другой покупатель и приобрел репродукции второго художника. Сделки были оформлены.
6. Внесите данные в соответствующие таблицы, в том числе отобразите предпочтения покупателя к картинам определенных художников.
7. Последовательно отобразить содержимое всех имеющихся в базе данных таблиц.