Лекция 2. Введение в язык SQL

Зарождение языка SQL

Помимо определения реляционной модели **Кодд** предложил язык для работы с данными в реляционных таблицах, названный **DSL/Alpha**. Вскоре после публикации статьи Кодда в IBM была организована группа для создания прототипа языка на базе его идей. Эта группа разработала упрощенную версию DSL/Alpha, которую назвали **SQUARE**. В результате усовершенствования SQUARE появился язык **SEQUEL**, который в конце концов получил имя **SQL**.

В середине 1980-х Национальный институт стандартизации США (American National Standards Institute, ANSI) начал разрабатывать **первый стандарт** языка SQL, который был опубликован в 1986 г. Дальнейшие доработки были отражены в следующих версиях стандарта SQL (1989, 1992, 1999, 2003, 2006, 2011 и 2016 гг.). Наряду с усовершенствованием базового языка в SQL появились и новые возможности для обеспечения объектно-ориентированной функциональности.

SQL идет рука об руку с **реляционной моделью**, потому что результатом SQL-запроса является **таблица** (в данном контексте также называемая *результирующим набором*). Таким образом, в реляционной базе данных можно создать новую постоянную таблицу, просто сохранив результирующий набор запроса. Аналогично в качестве входных данных запрос может использовать как постоянные таблицы, так и результирующие наборы других запросов.

Замечание: SQL не акроним (хотя многие настаивают, что это сокращение от Structured Query Language (Структурированный язык запросов)). Название этого языка произносится по буквам (т. е. «S», «Q», «L») или как «sequel» (сиквел).

Классы SQL-выражений

Язык SQL состоит из несколько отдельных частей. В нашем курсе будут рассмотрены:

- ***SQL-выражения управления схемой данных*** (*SQL schema statements*), предназначенные для определения структур данных, хранящихся в базе данных;

- ***SQL-выражения для работы с данными*** (*SQL data statements*), предназначенные для работы со структурами данных, ранее определенными с помощью SQL-выражений управления схемой;

▪ ***SQL-выражения управления транзакциями*** (*SQL transaction statements*), предназначенные для начала, завершения и отката транзакций.

Например, новая таблица базы данных создается с помощью SQL-выражения управления схемой *create table*, а чтобы заполнить ее данными, потребуется SQL-выражение для работы с данными *insert*.

Все элементы БД, созданные посредством SQL-выражений управления схемой, хранятся в специальном наборе таблиц, который называется ***словарем данных*** (*data dictionary*). Все эти «данные о базе данных» называют ***метаданными*** (*metadata*). К таблицам словаря данных можно делать запросы с помощью оператора *select*, в точности как к созданным вами таблицам. Таким образом, текущие структуры данных, развернутые в БД во время выполнения, становятся доступными.

SQL: непроцедурный язык

Когда вы раньше работали с языками программирования, вы привыкли к описанию переменных и структур данных, использованию условной логики (if-then-else), циклическим конструкциям (do while … end) и разделению кода на небольшие многократно используемые части (объекты, функции, процедуры). Код передается компилятору, и результирующий исполняемый код делает то, что вы запрограммировали. С каким бы языком программирования вы ни работали, Java, C#, C++ или любым другим ***процедурным*** языком, вы полностью управляете действиями программы.

С SQL, однако, понадобится отказаться от привычного контроля над выполнением, потому что SQL-выражения определяют необходимые входные и выходные данные, а способ выполнения выражения зависит от компонента механизма СУБД (database engine), называемого ***оптимизатором*** (*optimizer*). Работа оптимизатора заключается в том, чтобы рассмотреть SQL-выражение и с учетом конфигурации таблиц и доступных индексов принять решение о наиболее эффективном пути выполнения запроса. Большинство СУБД позволяют программисту влиять на решения оптимизатора с помощью ***подсказок оптимизатору*** (*optimizer hints*), например, предложений по использованию конкретного индекса. Подобные тонкости в основном выполняются администраторами БД или специалистами по вопросам производительности.

Сам по себе SQL не предназначен для написания полных приложений. Если требуется создать что-то сложнее простого сценария для работы с определенными данными, понадобится интегрировать SQL с каким-либо языком программирования. Некоторые производители баз данных сделали это за вас, например, **Oracle** с языком **PL/SQL** или **Microsoft** с **TransactSQL**. Благодаря этим языкам SQL-выражения для работы с данными являются

частью грамматики языка программирования, что позволяет свободно интегрировать запросы к БД с процедурными командами. Однако при использовании не характерного для БД языка, такого как Java, для выполнения SQL-выражений понадобится специальное средство. Некоторые из этих программных средств предоставляются производителями БД, тогда как другие создаются сторонними производителями или разработчиками ПО с открытым исходным кодом.

Серверы баз данных

Реляционные базы данных продаются уже более двух десятилетий. К самым популярным продуктам относятся:

- **Oracle Database** от Oracle Corporation
- **SQL Server** от Microsoft
- **DB2 Universal Database** от IBM
- **Sybase Adaptive Server** от Sybase

Все эти серверы БД делают примерно одно и то же, хотя некоторые лучше оснащены для работы с очень большими или высокопроизводительными БД. Другие лучше ведут себя при работе с объектами, или очень большими файлами, или XML-документами и т.д. Кроме того, очень хорошо, что все эти серверы совместимы с последним стандартом ANSI SQL.

Альтернативой коммерческим серверам БД являются два наиболее распространенных сервера БД с открытым исходным кодом – **PostgreSQL** и **MySQL**.

Создание и работа с базами данных

Для создания новой БД используется следующее выражение:

```
CREATE DATABASE database_name;
```

Выбор БД для использования осуществляется выражением:

```
USE database_name;
```

Далее, создадим таблицу **person**:

```
CREATE TABLE person
  (person_id SMALLINT UNSIGNED,
  fname VARCHAR(20),
  lname VARCHAR(20),
```

```
gender CHAR(1),
birth_date DATE,
street VARCHAR(30),
city VARCHAR(20),
country VARCHAR(20),
postal_code VARCHAR(20),
CONSTRAINT pk_person PRIMARY KEY (person_id)
);
```

В этом выражении должно быть понятно все, кроме последнего элемента. При описании таблицы необходимо сообщить серверу БД, какой столбец или столбцы будут играть роль первичного ключа таблицы. Осуществляется это путем создания ***ограничения*** (*constraint*) для таблицы. В описание таблицы можно добавить ограничение одного из нескольких типов. Данное ограничение является ***ограничением первичного ключа*** (*primary-key constraint*). Оно накладывается на столбец *person_id* и получает имя *pk_person*.

Существует другой тип ограничения – ***проверочное ограничение*** (*check constraint*), ограничивающее допустимые значения конкретного столбца. MySQL позволяет вводить в описание столбца проверочное ограничение следующим образом:

```
gender CHAR(1) CHECK (gender IN ('M', 'F'));
```

На большинстве серверов БД проверочные ограничения работают соответствующим образом, а сервер MySQL допускает описание проверочных ограничений, но не выполняет их проверку. Но MySQL предоставляет другой символьный тип данных – ***enum*** (*перечисление*), который вводит проверочное ограничение в описание типа. Вот как это выглядело бы для описания столбца *gender*:

```
gender ENUM('M', 'F');
```

Если требуется убедиться, что таблица **person** действительно существует, можно использовать команду *describe* (или *desc*) и посмотреть описание таблицы:

```
DESC person;
```

В некоторых случаях нам требуется изменить тип данных столбца. В Oracle и MySQL это делается следующим образом:

```
ALTER TABLE table_name MODIFY column_name new_data_type;
```

В SQL Server:

ALTER TABLE table_name ALTER COLUMN column_name new_data_type;

Для добавления столбца в таблицу используется команда:

ALTER TABLE table_name ADD column_name data_type;

Также возможно добавить сразу несколько столбцов. Например,

ALTER TABLE person ADD email char(30), telephone char(20);

Наш следующий шаг – создать таблицу **favorite_food**:

```
CREATE TABLE favorite_food
 (person_id SMALLINT UNSIGNED,
 food VARCHAR(20),
 CONSTRAINT pk_favorite_food PRIMARY KEY (person_id, food),
 CONSTRAINT fk_fav_food_person_id FOREIGN KEY (person_id)
 REFERENCES person (person_id)
 );
```

Это очень похоже на выражение *create table* для таблицы **person**, за несколькими исключениями:

• Поскольку у человека может быть несколько любимых блюд (что и стало причиной создания данной таблицы), одного столбца *person_id* для обеспечения уникальности в таблице недостаточно. Поэтому первичный ключ данной таблицы состоит из двух столбцов: *person_id* и *food*.

• Таблица **favorite_food** содержит другой тип ограничения – ***ограничение внешнего ключа*** (*foreign-key constraint*). Оно ограничивает значения столбца *person_id* таблицы **favorite_food**, позволяя ему включать только те значения, которые есть в таблице **person**.

Замечание: Если при создании таблицы ограничение внешнего ключа не было указано, его можно добавить позже с помощью оператора *alter table*.

Все серверы БД, присутствующие сегодня на рынке, обеспечивают надежный и устойчивый к ошибкам метод **формирования числовых ключей.** В MySQL надо просто включить для столбца первичного ключа свойство *auto_increment* (*автоприращение*). Обычно это делается при создании таблицы, но мы занимаемся этим сейчас, чтобы повторить еще одно SQL-выражение управления схемой, которое меняет описание существующей таблицы:

ALTER TABLE person MODIFY person_id SMALLINT UNSIGNED AUTO_INCREMENT;

При вводе данных в таблицу **person** просто задайте значение *null* для столбца *person_id*, и MySQL заполнит столбец следующим доступным числом (для столбцов с автоприращением MySQL по умолчанию начинает отсчет с 1).

Теперь, когда все расставлено по местам, пора добавить кое-какие данные. Следующее выражение создает в таблице **person** строку для Вильяма Тернера:

```
INSERT INTO person
  (person_id, fname, lname, gender, birth_date)
  VALUES (null, 'William','Turner', 'M', '1972-05-27');
```

Для удаления БД и таблицы используются следующие команды:

```
DROP DATABASE database_name;
```

```
DROP TABLE table_name;
```

Иногда возникает необходимость удалить столбец из таблицы в SQL. В MySQL для этого используется команда:

```
ALTER TABLE table_name DROP column_name;
```

В Oracle и SQL Server синтаксис команды следующий:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Литература:

1. Алан Бьюли. Изучаем SQL: пер. с англ. – СПб-М.: Символ, O'Reilly, 2007. – 310 с.
2. https://www.1keydata.com/sql/sql.html