Основы языка SQL

I. Стандарты языка SQL

Первые попытки практического применения основных положений реляционной модели данных (в т.ч. и операций РА) сразу же показали, что нужен ***специальный язык для работы с реляционными БД***.

Причем требовался ***простой непроцедурный язык***, пригодный даже для пользователей, которые не имеют навыков программирования.

В результате к концу 1970-х годов появилось несколько версий такого языка:

- SEQUEL и SQUARE – фирма IBM;
- SQL – СУБД Oracle;

Первый стандарт по языку SQL вышел в США (1986 г.), что способствовало достижению ***совместимости*** разных СУБД.

В 1989 г. этот документ получил статус ***международного стандарта.***

Затем с участием ANSI и ISO были приняты и опубликованы стандарты SQL:1992, SQL:1999 и SQL:2003.

Для каждого нового стандарта характерно:

- ❖улучшение синтаксиса;
- ❖расширение типов данных;
- ❖появление дополнительных возможностей языка.

Подавляющее большинство существующих СУБД ***поддерживают*** эти стандарты, однако ***с разной степенью соответствия***.
Здесь важно понимать, что в стандартах фиксируются только ***основные концепции языка*** после их признания большинством специалистов.

Параллельно существует большое количество ***диалектов*** этого языка.

Причем дополнительные возможности (расширения) языка SQL первоначально появляются именно в диалектах, развиваемых

II. Язык запросов по образцу (QBE)

Язык QBE (Query-By-Example) – это ***визуальное средство***, которое помогает непрофессиональным пользователям при работе с БД.

Средства QBE позволяют на экране компьютера заполнить бланк (шаблон) запроса, в т.ч. путем перетаскивания (Drag-and-Drop) отдельных элементов графического интерфейса.

При этом эквивалентная директива языка SQL ***генерируется автоматически***.

Есть возможность просмотреть ее и даже исправить.

Средства QBE присутствуют во многих

III. Возможности современного языка SQL

a) Средства определения данных (DDL)

$$\begin{Bmatrix} CREATE \\ DROP \\ ALTER \end{Bmatrix} \begin{Bmatrix} TABLE \\ VIEW \\ INDEX \end{Bmatrix}$$

VIEW – представление (виртуальная таблица)

b) Средства манипулирования данными (DML) SELECT, DELETE, INSERT, UPDATE

c) Средства управления транзакциями

- COMMIT – завершить транзакцию, т.е. зафиксировать ее результаты
- ROLLBACK – откатить транзакцию
- SAVEPOINT – сохранить промежуточную точку (на случай отката транзакции)

d) Средства административного

$$\begin{Bmatrix} CREATE \\ DROP \\ ALTER \end{Bmatrix} \begin{Bmatrix} DATABASE/SCHEMA \\ DBAREA \\ USER \end{Bmatrix}$$

GRANT – предоставить права

REVOKE – отменить права

DBAREA – область хранения данных

управление защитой данных

е) Процедурные расширения языка

Поддерживают ***средства программирования*** для построения сложных процедур работы с БД (например, хранимые процедуры).

Здесь пока не достигнут высокий уровень стандартизации.

Поэтому практически каждая СУБД использует

IV. Создание таблиц БД с помощью языка SQL

В простейшем виде команда **CREATE TABLE** имеет следующий синтаксис:

CREATE TABLE tab_name ({ col_def | tab_cnstr } [, …])

В этой синтаксической формуле применяются следующие обозначения:

- фигурные скобки { } определяют ***обязательный элемент***;
- вертикальная черта | означает ***выбор*** одного из приведенных вариантов;
- квадратные скобки [ ] определяют ***необязательный элемент***;
- многоточие … указывает возможность ***неоднократного повторения*** конструкции.

Элемент **col_def** обозначает ***определение отдельной колонки таблицы***.

Этот элемент имеет следующий синтаксис:

col_def ::= { col_name data_type }

[**DEFAULT** const_expr] [col_cnstr] [, …]

Прежде всего, нужно ***обязательно*** определить имя колонки (**col_name**), а также тип данных (**data_type**) для этой колонки.

Современные СУБД позволяют обрабатывать данные разных типов:

- INT, SMALLINT — целые числа;
- NUMERIC, DECIMAL — числа с фиксированной точкой;
- REAL, FLOAT — числа с плавающей точкой;
- CHAR, VARCHAR — строки символов постоянной и переменной длины;
- MONEY, SMALLMONEY — денежные значения;

Необязательное ключевое слово **DEFAULT** определяет значение по умолчанию – **const_expr**. Это значение будет использовано, если при вводе записи явно не указано другое значение.

Кроме того, для колонки можно определить набор ограничений – **col_cnstr**.

Эти ограничения ***повышают качество данных***, которые хранятся в БД, а также поддерживают ***ссылочную целостность*** для взаимосвязанных таблиц.

Может использоваться ***несколько видов***

а) Обязательные значения

Это ограничение применяется в случае, если для некоторого столбца в каждой строке таблицы требуется наличие конкретного значения (**NOT NULL**).

Например, каждый сотрудник ***обязательно*** занимает ту или иную должность (Position).

Тогда столбец Position должен определяться следующим образом:

Position VARCHAR(10) **NOT NULL**

При установке такого ограничения СУБД препятствует появлению в этом столбце

b) Простой первичный ключ

Определяется с помощью спецификатора **PRIMARY KEY**. Например:

Object_ID **INTEGER PRIMARY KEY**

При этом для столбца Sub_ID автоматически гарантируется уникальность значений, а также становятся запрещенными неопределенные значения (NULL).

В таблице можно определить ***только***

c) Простой альтернативный ключ

Иногда в дополнение к первичному ключу необходимо иметь альтернативные ключи, которые обеспечивают уникальность значений для других столбцов.

В этом случае применяется спецификатор **UNIQUE**. Например:

Object_name VARCHAR(20) **NOT NULL UNIQUE**

d) Проверочные ограничения

С помощью спецификатора **CHECK**(log_expr) можно задать ограниченный диапазон возможных значений для некоторого столбца.

Логическое выражение **log_expr** может объединять несколько условий контроля при вводе данных. Например:

Kurs **INTEGER NOT NULL**

е) Простой внешний ключ

Объявляется в дочерней (подчиненной) таблице с помощью конструкции

[FOREIGN KEY] REFERENCES ref_table
[(ref_col)]

Если в родительской таблице **ref_table** ссылка осуществляется на первичный ключ, то параметр **ref_col** можно не указывать.

Этот параметр является обязательным при

Ограничения на уровне всей таблицы

В директиве CREATE TABLE такие ограничения объявляются с помощью синтаксического элемента **tab_cnstr**.

Этот элемент может присутствовать в общем списке вместе с определениями отдельных столбцов.

Такой вариант ограничений обычно применяется в случае ***составных ключей***.

При этом используются те же самые

Объявление составного первичного или альтернативного ключа:

{ PRIMARY KEY | UNIQUE } { (col_name [, …]) }

Объявление составного внешнего ключа:

FOREIGN KEY (col_name [, …])
REFERENCES ref_tab [(ref_col [, …])]

Пример создания таблицы с составным первичным ключом:

CREATE TABLE Рецепты (

Код_блюда **INTEGER**, Код_продукта **INTEGER**,

Вес_брутто **REAL**, Вес_нетто **REAL**,

V. Выборка данных с помощью языка SQL

Команда **SELECT** позволяет извлечь данные из одной или нескольких таблиц, а также (в случае необходимости) вычислить по этим данным производные значения.

При формировании этой команды ***описывается только необходимый результат***, т.е. набор выходных данных в виде таблицы.

Когда СУБД начинает выполнять конкретную команду SELECT, с помощью ***оптимизатора запросов*** строится определенная

В структуре этой мощной команды, которая имеет достаточно сложный синтаксис, можно выделить несколько ***основных разделов***:

SELECT ⟨ Список_выбора ⟩

[**INTO** ⟨ Новая_таблица ⟩]

FROM ⟨ Набор_источников_данных ⟩

[**WHERE** ⟨ Условия_отбора_записей ⟩]

[**GROUP BY** ⟨ Ключи_группировки ⟩]

[**HAVING** ⟨ Условия_отбора_групп ⟩]

[**ORDER BY** ⟨ Ключи_сортировки ⟩]

Обязательными являются только два раздела:

- **SELECT**, где указываются столбцы, которые должны присутствовать в выходной таблице;
- **FROM**, где задается перечень таблиц и других

Рассмотрим наиболее распространенные варианты применения команды **SELECT**.

1) Полное отображение таблицы

SELECT * FROM ⟨ имя_исх_таб ⟩

Здесь символ * означает «все столбцы».

2) Отображение конкретных столбцов таблицы

SELECT ⟨ список_столбцов ⟩

FROM ⟨ имя_исх_таб ⟩

В списке столбцов могут присутствовать ***выражения*** языка SQL, что означает включение ***вычисляемых полей*** в выходную таблицу.

В эти выражения обычно входят константы и названия полей (только числовых типов), а также арифметические операции.

Для построения сложных выражений разрешено применять круглые скобки.

3) Выборка записей по заданному условию

SELECT ⟨ список_столбцов ⟩

FROM ⟨ имя_исх_таб ⟩

WHERE ⟨ условие_отбора ⟩

В условие отбора могут входить простые операции сравнения: =, <, > и т.п.

Более сложные условия строятся с помощью логических операций **AND**, **OR** или **NOT**.

Кроме того, в условии отбора можно:

- указать шаблон поиска (LIKE);
- проверить принадлежность к диапазону (BETWEEN) или множеству (IN).

4) Сортировка результатов запроса

SELECT ⟨ список_столбцов ⟩
FROM ⟨ имя_исх_таб ⟩
ORDER BY ⟨ Ключи_сортировки ⟩

Каждый ключ сортировки содержит название столбца выходной таблицы (обязательно), а также указатель порядка сортировки:

- ASC – возрастающий порядок (по умолчанию);
- DESC – убывающий порядок.

5) Применение агрегатных (итоговых) функций

При выборке данных из таблиц БД агрегатные функции позволяют произвести ***статистическую обработку***, что важно для подсчета ***итогов***.

Функция	Результат
COUNT( )	количество значений
SUM( )	сумма значений
AVG( )	среднее значение
MIN( )	минимальное значение
MAX( )	максимальное значение

Кроме специального случая COUNT(*), каждая из этих функций работает с ***отдельным столбцом***, указанным в аргументе функции.

Пример 1. Усреднение значений для заданного столбца:

SELECT AVG(⟨имя_столбца⟩)
FROM ⟨ имя_исх_таб ⟩

Пример 2. Подсчет общего числа записей в выходной таблице:

SELECT COUNT(*)
FROM ⟨ имя_исх_таб ⟩
WHERE ⟨ условия_отбора ⟩

Пример 3. Подсчет неповторяющихся значений в заданном столбце:

SELECT COUNT(DISTINCT ⟨имя_столбца⟩)
FROM ⟨ имя_исх_таб ⟩

6) Запросы с группировкой

Часто при анализе табличных данных требуется выполнить их ***группировку***, т.е. сделать так, чтобы в одну группу попадали записи с одинаковыми значениями для заданных атрибутов (***ключи группировки***).

В этом случае применяется следующая команда:

SELECT ⟨ список_столбцов ⟩
FROM ⟨ имя_исх_таб ⟩
GROUP BY ⟨ Ключи_группировки ⟩

Логика работы запросов с группировкой требует ***тесной связи*** между разделами **SELECT** и **GROUP BY.**

В частности, любой элемент списка столбцов в разделе **SELECT** должен иметь ***единственное значение*** для каждой группы.

Следовательно, в этом списке могут быть только:

- имена столбцов, которые являются ключами группировки;
- агрегатные функции;
- выражения, состоящие из перечисленных выше элементов.

Пример. Пусть имеется таблица ПРЕПОДАВАТЕЛИ, где содержатся следующие данные: Таб_номер, ФИО, Должность, Зарплата, Кафедра.

С помощью этих данных требуется по каждой кафедре определить количество

Решение этой задачи дает следующий запрос:

```
SELECT Кафедра, COUNT(*), SUM(Зарплата)
FROM   ПРЕПОДАВАТЕЛИ
GROUP BY  Кафедра
```

7) Вложенные запросы (подзапросы)

В разделе **WHERE** оператора **SELECT** может присутствовать ***вложенный запрос***.

Результат выполнения этого внутреннего запроса (подзапроса) передается внешнему запросу.

Пример. Список преподавателей, которые получают зарплату выше средней,

```
SELECT   ФИО, Должность
FROM   ПРЕПОДАВАТЕЛИ
WHERE   Зарплата >
( SELECT   AVG(Зарплата)
FROM   ПРЕПОДАВАТЕЛИ )
```

8) Многотабличные запросы

Для выборки данных из нескольких таблиц применяется механизм ***соединения*** этих таблиц (операция **JOIN**).

При ***внутреннем соединении*** исходные таблицы можно через запятую указать в разделе **FROM**.

В дополнение к этому, раздел **WHERE** должен содержать ***условия для соединения строк***

Пример 1.

По таблицам КАФЕДРЫ и ПРЕПОДАВАТЕЛИ нужно получить общий список преподавателей с указанием названия кафедры, на которой работает преподаватель, и телефона этой кафедры.

Выполняем следующий запрос:

```
SELECT  ФИО, Название, Телефон
FROM    ПРЕПОДАВАТЕЛИ  a, КАФЕДРЫ  b
WHERE   a.Код_каф = b.Код_каф
```

В этом запросе для сокращенного обозначения таблиц используются их ***псевдонимы*** – **a** и **b**.

Начиная с СУБД Oracle 9i и стандарта SQL:1992, стало возможным в разделе **FROM** оператора **SELECT** использовать следующую конструкцию:

⟨left_tab⟩ ⟨join_type⟩ **⟨right_tab⟩ ON** ⟨join_cond⟩

Эта конструкция применяется для соединения таблиц **⟨left_tab⟩** и **⟨right_tab⟩**, причем условия **⟨join_cond⟩** для соединения строк переносятся внутрь раздела **FROM.**

Дополнительные условия для отбора записей по другим критериям остаются в разделе **WHERE**, что делает текст запроса ***более***

Синтаксический элемент **⟨join_cond⟩** может принимать следующие значения:

- [INNER] JOIN — внутреннее соединение (применяется по умолчанию);
- LEFT [OUTER] JOIN — левое внешнее соединение;
- RIGHT [OUTER] JOIN — правое внешнее соединение;
- FULL [OUTER] JOIN — полное внешнее соединение.

Пример 2.

Запрос из примера 1 можно записать так:

SELECT ФИО, Название, Телефон
FROM ПРЕПОДАВАТЕЛИ
NATURAL JOIN КАФЕДРЫ

Ключевое слово **NATURAL** означает, что соединение должно проводиться по равенству значений в столбцах с ***одинаковыми названиями***.

Это слово позволяет полностью исключить из запроса условия соединения строк.