Topic 7. SQL

Язык SQL подразделяется на несколько частей.
DDL – Data Definition Language (язык описания данных)
DML – Data Manipulation Language (язык манипулирования данными), который содержит следующие конструкции:
SELECT – выборка данных
INSERT – вставка новых данных
UPDATE – обновление данных
DELETE – удаление данных
Реляционная база данных (далее просто БД) представляет из себя совокупность таблиц, связанных между собой. Если говорить грубо, то БД – файл в котором данные хранятся в структурированном виде.
СУБД – Система Управления этими Базами Данных, т.е. это комплекс инструментов для работы с конкретным типом БД (MS SQL, Oracle, MySQL, Firebird, …).
Таблица представляет из себя совокупность столбцов. Столбцы, так же могут называть полями или колонками.
Таблица – это главный объект БД, все данные БД хранятся построчно в столбцах таблицы. Строки, записи – тоже синонимы.
Для каждой таблицы, как и ее столбцов задаются наименования, по которым впоследствии к ним идет обращение.

SQL — язык позволяющий осуществлять запросы в БД посредством СУБД. В конкретной СУБД, язык SQL может иметь специфичную реализацию (свой диалект).

DDL и DML — подмножество языка SQL:

Язык DDL служит для создания и модификации структуры БД, т.е. для создания/изменения/удаления таблиц и связей.

Язык DML позволяет осуществлять манипуляции с данными таблиц, т.е. с ее строками. Он позволяет делать выборку данных из таблиц, добавлять новые данные в таблицы, а так же обновлять и удалять существующие данные.

В языке SQL можно использовать 2 вида комментариев (однострочный и многострочный):

-- однострочный комментарий

и

/* многострочный комментарий */

Основные типы данных используемые в MS SQL:

Строка переменной длины (varchar(N) и nvarchar(N))- при помощи числа N, мы можем указать максимально возможную длину строки для соответствующего столбца. Например, если мы хотим сказать, что значение столбца «ФИО» может содержать максимум 30 символов, то необходимо задать ей тип nvarchar(30). Отличие varchar от nvarchar заключается в том, что varchar позволяет хранить строки в формате ASCII, где один символ занимает 1 байт, а nvarchar хранит строки в формате Unicode, где каждый символ занимает 2 байта.

Тип varchar стоит использовать только в том случае, если вы на 100% уверены, что в данном поле не потребуется хранить Unicode символы. Например, varchar можно использовать для хранения адресов электронной почты, т.к. они обычно содержат только ASCII символы.

Строка фиксированной длины (char(N) и nchar(N)) - от строки переменной длины данный тип отличается тем, что если длина строка меньше N символов, то она всегда дополняется справа до длины N пробелами и сохраняется в БД в таком виде, т.е. в базе данных она занимает ровно N символов (где один символ занимает 1 байт для char и 2 байта для типа nchar).

Целое число (int) - данный тип позволяет нам использовать в столбце только целые числа, как положительные, так и отрицательные. Для справки (сейчас это не так актуально для нас) – диапазон чисел который позволяет тип int от -2 147 483 648 до 2 147 483 647. Обычно это основной тип для задания идентификаторов.

Вещественное или действительное число (float) - простым языком, то это числа, в которых может присутствовать десятичная точка (запятая).
Дата (date) - если в столбце необходимо хранить только Дату, которая состоит из трех составляющих: Числа, Месяца и Года. Например, 15.02.2014 (15 февраля 2014 года). Данный тип можно использовать для столбца «Дата приема», «Дата рождения» и т.п., т.е. в тех случаях, когда нам важно зафиксировать только дату, или, когда составляющая времени нам не важна и ее можно отбросить или если она не известна.
Время (time) - данный тип можно использовать, если в столбце необходимо хранить только данные о времени, т.е. Часы, Минуты, Секунды и Миллисекунды. Например, 17:38:31.3231603 Например, ежедневное «Время отправления рейса».
Дата и время (datetime) - данный тип позволяет одновременно сохранить и Дату, и Время. Например, 15.02.2014 17:38:31.323.
Флаг (bit) - данный тип удобно применять для хранения значений вида «Да»/ «Нет», где «Да» будет сохраняться как 1, а «Нет» будет сохраняться как 0.

Простую базу данных (без указания дополнительных параметров) можно создать, выполнив следующую команду:

CREATE DATABASE Product

Удалить базу данных можно командой (стоит быть очень осторожным с данной командой):

DROP DATABASE Product

Создать таблицу используя описания в том виде как они есть, используя пробелы и символы кириллицы:

```
CREATE TABLE [Компьютер](
[Код] int,
[Модель] nvarchar(50),
[Скорость процессора ] int,
[Объем оперативной памяти] int,
[Размер диска] int,
[Скорость привода] nvarchar(10)
[Цена] float)
```

В данном случае нам придется заключать имена в квадратные скобки […]

В базе данных для большего удобства все наименования лучше задавать на латинице и не использовать в именах пробелы. В MS SQL обычно в данном случае каждое слово начинается с прописной буквы, например, для поля «*Скорость привода*», мы могли бы задать имя *SpeedCd*.

В некоторых СУБД более предпочтительным может быть следующий формат наименований «*SPEED_CD*», например, такой формат часто используется в БД ORACLE. Естественно при задании имя поля желательно чтобы оно не совпадало с ключевыми словами используемые в СУБД.
Очень часто для наименования поля идентификатора используется слово *ID*.

```
CREATE TABLE PC (
Code int,
Model nvarchar(50),
Speed int,
Ram int,
Hd int,
Cd nvarchar(10)
Price nvarchar(10))
```

Общая концепция языка SQL для большинства СУБД остается одинаковой (по крайней мере, об этом я могу судить по тем СУБД, с которыми мне довелось поработать). Отличие DDL в разных СУБД в основном заключаются в типах данных (здесь могут отличаться не только их наименования, но и детали их реализации), так же может немного отличаться и сама специфика реализации языка SQL (т.е. суть команд одна и та же, но могут быть небольшие различия в диалекте.

Владея основами SQL вы сможете перейти с одной СУБД на другую, т.к. вам в данном случае нужно будет только разобраться в деталях реализации команд в новой СУБД, т.е. в большинстве случаев достаточно будет просто провести аналогию.

```
CREATE TABLE PC (
Code int,
Model nvarchar2(50), -- nvarchar2 в ORACLE эквивалентен nvarchar в MS SQL
Speed int,
Ram int,
Hd int,
Cd nvarchar2(10),
Price nvarchar2(10),
PRIMARY KEY(Code))
```

Первичный ключ

При создании таблицы желательно, чтобы она имела уникальный столбец или же совокупность столбцов, которая уникальна для каждой ее строки – по данному уникальному значению можно однозначно идентифицировать запись. Такое значение называется первичным ключом таблицы.

Нормализация базы данных – дробление ее на подтаблицы.
Product(maker, model, type)
PC(code, model, speed, ram, hd, cd, price)
Laptop(code, model, speed, ram, hd, screen, price)
Printer(code, model, color, type, price).

Простой оператор SELECT

Оператор **SELECT** осуществляет выборку из базы данных.

*SELECT * FROM PC;*

При этом столбцы и строки результирующего набора не упорядочены.
Чтобы упорядочить поля результирующего набора, их следует перечислить через запятую в нужном порядке

SELECT price, speed, hd, ram, cd, model, code
FROM PC;

Если нет необходимости выводить все колонки, то нужно перечислить только те, которые нужны в результирующем запросе:

SELECT speed, ram
FROM PC;

Если требуется получить только уникальные строки, то можно использовать ключевое слово *DISTINCT*:

```
SELECT DISTINCT speed, ram
FROM PC;
```

Для упорядочивания строк используется предложение *ORDER BY* список полей, являющееся всегда последним предложением в операторе *SELECT*. При этом в списке полей могут указываться как имена полей, так и их порядковые позиции в списке предложения *SELECT*.

```
SELECT DISTINCT speed, ram
FROM PC
ORDER BY ram DESC;
```

Или

```
SELECT DISTINCT speed, ram
FROM PC
ORDER BY 2 DESC;
```

Сортировку можно проводить по возрастанию (*ASC* по умолчанию) или по убыванию (*DESC*).

Указывать номера столбцов не рекомендуется.

Горизонтальную выборку реализует предложение *WHERE*, которое записывается после предложения *FROM*. При этом в результирующий набор попадут только те строки, для каждой из которых значение равно *TRUE.* Например, запрос «получить информацию о частоте процессора и объеме оперативной памяти для компьютеров с ценой ниже $500» можно сформулировать следующим образом:

```
SELECT DISTINCT speed, ram
FROM PC
WHERE price < 500
ORDER BY 2 DESC;
```

Операции сравнения «<» (меньше чем). Кроме этой операции сравнения могут использоваться: «=» (равно), «>» (больше), «>=» (больше или равно), «<=» (меньше или равно) и «<>» (не равно).

cd = '24x' - 24-скоростной CD-ROM

color <> 'y' - Не цветной принтер

ram – 128 > 0 - Объем оперативной памяти свыше 128 Мбайт

type = 'laptop' - Типом продукции является портативный компьютер

price < 1000 - Цена меньше 1000

*Price <= speed*2* - Цена не превышает удвоенной частоты процессора

Сортировку можно выполнять по столбцам, отсутствующим в списке **SELECT**.

SELECT model FROM PC
ORDER BY price DESC;

*SELECT **DISTINCT** model FROM PC*
ORDER BY price DESC;

уже вызовет ошибку: элементы ORDER BY должны входить в список выбора, если указывается SELECT DISTINCT.

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов **AND**, **OR** или **NOT**.

Правила комбинирования всех трех истинностных значений легче запомнить, обозначив **TRUE** как 1, **FALSE** как 0.

Логические операторы при отсутствии скобок, как и арифметические операторы, выполняются в соответствии с их старшинством.

Операция *NOT* имеет наивысший приоритет

-- модели, не являющиеся ПК
-- второй предикат ничего не меняет, т.к. он добавляет условие,
-- уже учтенное в первом предикате
SELECT maker, model, type
FROM Product
WHERE NOT type='PC' OR type='Printer';

Поменять порядок выполнения логических операторов можно при помощи скобок:

-- модели, не являющиеся ПК или принтером, т.е. модели ноутбуков
SELECT maker, model, type
FROM Product
*WHERE NOT **(**type='PC' OR type='Printer'**)**;*

Следующий приоритет имеет оператор ***AND***.

-- модели ПК, выпускаемые производителем A, и любые модели производителя B
SELECT maker, model, type
FROM Product
WHERE type='PC' AND maker='A' OR maker='B';
-- модели ПК, выпускаемые производителем A или производителем B
SELECT maker, model, type
FROM Product

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения: =, >, <, >=, <=, <>.
Числа сравниваются в соответствии с их алгебраическим значением.
Строки сравниваются в соответствии с их алфавитной последовательностью.
Если a1a2…an и в1 в…вn — две последовательности символов, то первая «меньше» второй, если a1 < в1, или если a1 = в1 и a2 < в2 и т. д.
Например, ‘folder’ < ‘for’, так как первые две буквы этих строк совпадают, а третья буква строки ‘folder’ предшествует третьей букве строки ‘for’. Также справедливо неравенство ‘bar’ < ‘barber’, поскольку первая строка является префиксом второй.
Получить информацию о компьютерах, имеющих частоту процессора не менее 500 МГц и цену ниже $800:

```
SELECT *
FROM PC
WHERE speed >= 500 AND
price < 800;
```

Получить информацию обо всех принтерах, которые не являются матричными и стоят меньше $300:

```
SELECT *
FROM printer
WHERE NOT (type = 'matrix') AND
price < 300;
```

Предикат *BETWEEN* проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным словом *AND*. Естественно, как и для предиката сравнения, выражения в предикате *BETWEEN* должны быть совместимы по типам.

Предикат

exp1 BETWEEN exp2 AND exp3

равносилен предикату

exp1 >= exp2 AND exp1 <= exp3

А предикат

exp1 NOT BETWEEN exp2 AND exp3

равносилен предикату

NOT (exp1 BETWEEN exp2 AND exp3)

Если значение предиката *exp1 BETWEEN exp2 AND exp3* равно TRUE, в общем случае это отнюдь не означает, что значение предиката *exp1 BETWEEN exp3 AND exp2* тоже будет TRUE, так как первый будет интерпретироваться как предикат:

exp1 >= exp2 AND exp1 <= exp3

а второй как:

exp1 >= exp3 AND exp1 <= exp2

Найти модель и частоту процессора компьютеров стоимостью от $400 до $600:

SELECT model, speed
FROM PC
WHERE price BETWEEN 400 AND 600;

Предикат **IN** определяет, будет ли значение проверяемого выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса. Здесь табличный подзапрос это обычный оператор **SELECT**, который создает одну или несколько строк для одного столбца, совместимого по типу данных со значением проверяемого выражения. Если целевой объект эквивалентен хотя бы одному из указанных в предложении **IN** значений, истинностное значение предиката **IN** будет равно **TRUE**. Если для каждого значения X в предложении **IN** целевой объект <> X, истинностное значение будет равно **FALSE**.

Найти модель, частоту процессора и объем жесткого диска тех компьютеров, которые комплектуются накопителями 10 или 20 Гбайт:

SELECT model, speed, hd
FROM PC
WHERE hd IN (10, 20);

Найти модель, частоту процессора и объем жесткого диска компьютеров, которые комплектуются накопителями 10 Гбайт или 20 Гбайт и выпускаются производителем A:

```
SELECT model, speed, hd
FROM PC
WHERE hd IN (10, 20) AND
model IN (SELECT model
FROM product
WHERE maker = 'A');
```

Имена столбцов, указанные в предложении **SELECT**, можно переименовать. Это делает результаты более читабельными, поскольку имена полей в таблицах часто сокращают с целью упрощения набора. Ключевое слово ***AS***, используемое для переименования, согласно стандарту можно и опустить.

```
SELECT ram AS Mb, hd Gb
FROM PC
WHERE cd = '24x‘
```

Переименование особенно желательно при использовании в предложении **SELECT** выражений для вычисления значения. Эти выражения позволяют получать данные, которые не находятся непосредственно в таблицах.

Чтобы вывести объем оперативной памяти в килобайтах, можно написать:

```
SELECT ram*1024 AS Kb, hd Gb
FROM PC
WHERE cd = '24x'
```

Иногда бывает необходимо выводить поясняющую информацию рядом с соответствующим значением. Это можно сделать, добавив строковое выражение как дополнительный столбец.

```
SELECT ram, 'Mb' AS ram_units, hd, 'Gb' AS hd_units
FROM PC
WHERE cd = '24x'
```

Если же явно не указать имя для выражения, то будет принят способ именования по умолчанию, который зависит от используемой СУБД. Так, в MS Access будут использованы имена типа выражение1 и т. д.

Предикат **LIKE** сравнивает строку, указанную в первом выражении с образцом, который определен во втором выражении для вычисления значения строки. В образце разрешается использовать трафаретные символы:

- символ подчеркивания (_), который можно применять вместо любого единичного символа в проверяемом значении;
- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.
- [ ] одиночный символ из набора символов (например, [zxy]) или диапазона ([a-z]), указанных в квадратных скобках. При этом можно перечислить сразу несколько диапазонов (например, [0-9a-z]);
- ^ который в сочетании с квадратными скобками исключает из поискового образца символы из набора или диапазона.

Если проверяемое значение соответствует образцу с учетом трафаретных символов, то значение предиката равно **TRUE**.

'abc%' - Любые строки, которые начинаются с букв «abc»

'abc_' - Строки длиной строго 4 символа, причем первыми символами строки должны быть «abc»

'%z' - Любая последовательность символов, которая обязательно заканчивается символом «z»

'%Kharkiv%' - Любая последовательность символов, содержащая слово «Kharkiv» в любой позиции строки

Найти все ПК, имена моделей которых заканчиваются на букву ‘o’:

```
SELECT *
FROM PC
WHERE model LIKE '%o' ;
```

Найти все ПК, имена моделей которых заканчиваются на букву ‘o’, но не на ‘go’

```
SELECT *
FROM PC
WHERE model NOT LIKE '%go' AND
model LIKE '%o' ;
```

Если искомая строка содержит трафаретный символ, то следует задать управляющий символ в предложении **ESCAPE**. Этот управляющий символ должен использоваться в образце перед трафаретным символом, сообщая о том, что последний следует трактовать как обычный символ. Например, если в некотором поле следует отыскать все значения, содержащие символ «_», то шаблон ‘%_%’ приведет к тому, что будут возвращены все записи из таблицы. В данном случае шаблон следует записать следующим образом:

```
'%#_%' ESCAPE '#'
```

Для проверки значения на соответствие строке «25%» можно воспользоваться таким предикатом:

```
LIKE '25|%' ESCAPE '|'
```

- */* 1 */*
- *--WHERE name LIKE'5%' -- начинается с 5*
- */* 2 */*
- *--WHERE name LIKE '5[%]' -- 5%*
- */* 3 */*
- *--WHERE name LIKE '5|%' ESCAPE '|'-- 5%*
- */* 4 */*
- *--WHERE name LIKE '%5|%%' ESCAPE '|' -- 5% в любом месте строки*
- */* 5 */*
- *--WHERE name LIKE '[0-9][a-zA-Z]%' -- первая цифра, вторая буква*
- */* 6 */*
- *--WHERE name LIKE '[a-z][0-9]%' -- первая буква, вторая цифра*
- */* 7 */*
- *--WHERE name LIKE '[^0-9]%' -- начинается не на цифру.*
- */* 8 */*
- *--WHERE name LIKE '%[02468]%' -- содержит четную цифру.*
- */* 9 */*
- *--WHERE name LIKE '%[02468][13579]%' -- комбинация четная-нечетная.*

IS [NOT] NULL позволяет проверить отсутствие (наличие) значения в полях таблицы.
Так, если требуется найти записи в таблице PC, для которых в столбце price отсутствует значение

```
SELECT *
FROM PC
WHERE price IS NULL
```

Характерной ошибкой является написание предиката в виде:

```
WHERE price = NULL;
```

Агрегатные функции
COUNT(*) - Возвращает количество строк источника записей
SUM - Возвращает сумму значений в указанном столбце
AVG - Возвращает среднее значение в указанном столбце
MIN - Возвращает минимальное значение в указанном столбце
MAX - Возвращает максимальное значение в указанном столбце
Все эти функции возвращают единственное значение. При этом функции **COUNT**, **MIN** и **MAX** применимы к данным любого типа, в то время как **SUM** и **AVG** используются только для данных числового типа.

Найти минимальную и максимальную цену на персональные компьютеры:

```
SELECT MIN(price) AS Min_price,
   MAX(price) AS Max_price
FROM PC;
```

Результатом будет единственная строка, содержащая агрегатные значения

Найти имеющееся в наличии количество компьютеров, выпущенных производителем A:

```
SELECT COUNT(*) AS Qty
FROM PC
WHERE model IN(SELECT model
FROM Product
WHERE maker = 'A');
```

Найти количество различных моделей, выпускаемых производителем A:

```
SELECT COUNT(model) AS Qty_model
FROM Product
WHERE maker = 'A';
```

Предложение *GROUP BY* используется для определения групп выходных строк, к которым могут применяться агрегатные функции *(COUNT, MIN, MAX, AVG и SUM).* Если при наличии предложения *GROUP BY*, в предложении *SELECT* отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы. Эту возможность, наряду с ключевым словом *DISTINCT*, можно использовать для исключения дубликатов строк в результирующем наборе.
Для каждой модели ПК определить их количество и среднюю стоимость:

```
SELECT model, COUNT(model) AS Qty_model,
    AVG(price) AS Avg_price
FROM PC
GROUP BY model;
```

Все строки с одинаковыми значениями *model* (номер модели) образуют группу, и на выходе *SELECT* вычисляются количество значений и средняя цена для каждой группы.

Использование в запросе нескольких источников

Простое перечисление таблиц через запятую практически не используется, поскольку оно соответствует реляционной операции, которая называется декартовым произведением. То есть в результирующем наборе каждая строка из одной таблицы будет сочетаться с каждой строкой из другой. Например, для

A

a	b
1	2
2	1

B

c	d
2	4
3	3

*SELECT **
FROM A, B;

a	b	c	d
1	2	2	4
1	2	3	3
2	1	2	4
2	1	3	3

Поэтому перечисление таблиц, как правило, используется совместно с условием соединения строк из разных таблиц, указываемым в предложении **WHERE**.

*SELECT **
FROM A, B
WHERE a = c;

a	b	c	d
2	1	2	4

то есть соединяются только те строки таблиц, у которых в указанных столбцах находятся равные значения.

Найти номер модели и производителя ПК, имеющих цену менее $600:

```
SELECT DISTINCT PC.model, maker
FROM PC, Product
WHERE PC.model = Product.model AND
price < 600;
```

JOIN — оператор языка sql, который является реализацией операции соединения. Операция соединения предназначена для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор.
Отличительными особенностями операции соединения являются следующее:
- в схему таблицы-результата входят столбцы обеих исходных таблиц (операндов), то есть схема результата является «сцеплением» схем операндов;
- каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда.

Определение того, какие именно исходные строки войдут в результат и в каких сочетаниях, зависит от типа операции соединения и от явно заданного *условия соединения*. При необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно).

```
SELECT
  field_name [,... n]
FROM
  Table1
  {INNER | {LEFT | RIGHT | FULL} OUTER | CROSS } JOIN
  Table2
    {ON <condition> | USING (field_name [,... n])}
```

В большинстве СУБД при указании слов *LEFT, RIGHT, FULL* слово *OUTER* можно опустить. Слово *INNER* также в большинстве СУБД можно опустить.

В общем случае СУБД при выполнении соединения проверяет условие (предикат) *condition*. Если названия столбцов по которым происходит соединение таблиц совпадают, то вместо *ON* можно использовать *USING*. Для *CROSS JOIN* условие не указывается.

Виды оператора JOIN

Для дальнейших пояснений будут использоваться следующие

City (Города)

Id	Name
1	Москва
2	Санкт-Петербург
3	Казань

Person (Люди)

Name	CityId
Андрей	1
Леонид	2
Сергей	1
Григорий	4

INNER JOIN

Оператор *внутреннего соединения INNER JOIN* соединяет две таблицы. Порядок таблиц для оператора неважен.
Каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной «соединённой» строки проверяется условие соединения (вычисляется предикат соединения). Если условие истинно, в таблицу-результат добавляется соответствующая «соединённая»

```
SELECT *
FROM
  Person
  INNER JOIN
  City
    ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Леонид	2	2	Санкт-Петербург
Сергей	1	1	Москва

OUTER JOIN

Соединение двух таблиц, в результат которого в обязательном порядке входят все строки либо одной, либо обеих таблиц.

LEFT OUTER JOIN

Оператор *левого внешнего соединения* *LEFT OUTER JOIN* соединяет две таблицы. Порядок таблиц для оператора важен. Результата логически формируется следующим образом:

1. В результат включается внутреннее соединение (*INNER JOIN*) левой и правой таблиц по предикату *p*.
2. Затем в результат добавляются те записи левой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких записей поля, соответствующие правой таблице, заполняются значениями *NULL*.

```
SELECT *
FROM
  Person -- Левая таблица
  LEFT OUTER JOIN
  City   -- Правая таблица
    ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Леонид	2	2	Санкт-Петербург
Сергей	1	1	Москва
Григорий	4	NULL	NULL

RIGHT OUTER JOIN

Оператор *правого внешнего соединения RIGHT OUTER JOIN* соединяет две таблицы. Порядок таблиц для оператора важен. Результата логически формируется следующим образом:

1. В результат включается внутреннее соединение (*INNER JOIN*) левой и правой таблиц по предикату *p*.
2. Затем в результат добавляются те записи правой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких записей поля, соответствующие левой таблице, заполняются значениями *NULL*.

```
SELECT *
FROM
  Person -- Левая таблица
  RIGHT OUTER JOIN
  City   -- Правая таблица
    ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Сергей	1	1	Москва
Леонид	2	2	Санкт-Петербург
NULL	NULL	3	Казань

FULL OUTER JOIN

Оператор *полного внешнего соединения FULL OUTER JOIN* соединяет две таблицы. Порядок таблиц для оператора неважен. Результата логически формируется следующим образом:

1. В результат включается внутреннее соединение (*INNER JOIN*) первой и второй таблиц по предикату *p*.
2. В результат добавляются те записи первой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких записей поля, соответствующие второй таблице, заполняются значениями *NULL*.
3. В результат добавляются те записи второй таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких записей поля, соответствующие первой таблице, заполняются значениями *NULL*.

```
SELECT *
FROM
  Person
  FULL OUTER JOIN
  City
    ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Сергей	1	1	Москва
Леонид	2	2	Санкт-Петербург
NULL	NULL	3	Казань
Григорий	4	NULL	NULL

CROSS JOIN

Оператор *перекрёстного соединения*, или *декартово произведение CROSS JOIN* соединяет две таблицы. Порядок таблиц для оператора неважен. Результата логически формируется следующим образом. Каждая строка одной таблицы соединяется с каждой строкой второй таблицы, давая тем самым в результате все возможные сочетания строк двух таблиц.

```
SELECT *
FROM
  Person
  CROSS JOIN
  City
```

```
SELECT *
FROM
  Person,
  City
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Андрей	1	2	Санкт-Петербург
Андрей	1	3	Казань
Леонид	2	1	Москва
Леонид	2	2	Санкт-Петербург
Леонид	2	3	Казань
Сергей	1	1	Москва
Сергей	1	2	Санкт-Петербург
Сергей	1	3	Казань
Григорий	4	1	Москва
Григорий	4	2	Санкт-Петербург
Григорий	4	3	Казань

Если в предложении *WHERE* добавить условие соединения (предикат *p*), то есть ограничения на сочетания кортежей, то результат эквивалентен операции *INNER JOIN* с таким же условием:

```
SELECT *
FROM
  Person,
  City
WHERE
  Person.CityId = City.Id
```

Таким образом,

```
t1 CROSS JOIN t2 WHERE p
```

и

```
t1 INNER JOIN t2 ON p
```

синтаксически являются альтернативными формами записи одной и той же логической операции внутреннего соединения по предикату *p*. Синтаксис CROSS JOIN + WHERE для операции соединения называют устаревшим, его не рекомендует стандарт SQL ANSI. В чистом виде декартово произведение практически не используется, оно, как правило, является промежуточным результатом выполнения операции горизонтальной проекции (выборки) при наличии в операторе SELECT предложения WHERE.

Объединение

Для объединения запросов используется служебное слово **UNION**:

```
<запрос 1>
UNION [ALL]
<запрос 2>
```

Предложение **UNION** приводит к появлению в результирующем наборе всех строк каждого из запросов. При этом, если определен параметр **ALL**, то сохраняются все дубликаты выходных строк, в противном случае в результирующем наборе присутствуют только уникальные строки. Заметим, что можно связывать вместе любое число запросов. Кроме того, с помощью скобок можно задавать порядок объединения.

Операция объединения может быть выполнена только при выполнении следующих условий:

- количество выходных столбцов каждого из запросов должно быть одинаковым;
- выходные столбцы каждого из запросов должны быть совместимы между собой (в порядке их следования) по типам данных;
- в результирующем наборе используются имена столбцов, заданные в первом запросе;
- предложение **ORDER BY** применяется к результату соединения, поэтому оно может быть указано только в конце всего составного запроса.

Найти номера моделей и цены ПК и портативных

```
SELECT model, price
FROM PC
UNION
SELECT model, price
FROM Laptop
ORDER BY price DESC;
```

model	Price
1750	1200
1752	1150
1298	1050
1233	980
1321	970
1233	950
1121	850
1298	700
1232	600
1233	600
1232	400
1232	350
1260	350

Найти тип продукции, номер модели и цену ПК и портативных компьютеров:

```
SELECT Product.type, PC.model, price
FROM PC INNER JOIN
     Product ON PC.model = Product.model
UNION
SELECT Product.type, Laptop.model, price
FROM Laptop INNER JOIN
     Product ON Laptop.model = Product.model
ORDER BY price DESC;
```

Type	Model	price
Laptop	1750	1200
Laptop	1752	1150
Laptop	1298	1050
PC	1233	980
Laptop	1321	970
PC	1233	950
PC	1121	850
Laptop	1298	700
PC	1232	600
PC	1233	600
PC	1232	400

Пересечение и разность

В стандарте языка SQL имеются предложения оператора **SELECT** для выполнения операций пересечения и разности результатов запросов-операндов. Этими предложениями являются **INTERSECT [ALL]** (пересечение) и **EXCEPT [ALL]** (разность), которые работают аналогично предложению **UNION**. В результирующий набор попадают только те строки, которые присутствуют в обоих запросах (INTERSECT) или только те строки первого запроса, которые отсутствуют во втором (EXCEPT). При этом оба запроса, участвующих в операции, должны иметь одинаковое число столбцов, и соответствующие столбцы должны иметь одинаковые (или неявно приводимые) типы данных. Имена столбцов результирующего набора формируются из заголовков первого запроса. Если не используется ключевое слово **ALL** (по умолчанию подразумевается DISTINCT), то при выполнении операции автоматически устраняются дубликаты строк.

```
SELECT name FROM Ships
INTERSECT
SELECT ship FROM Outcomes;
```

name
California
Kirishima
South Dakota
Tennessee
Washington

```
SELECT ship FROM Outcomes
INTERSECT
SELECT name FROM Ships;
```

ship
California
Kirishima
South Dakota
Tennessee
Washington

" Найти корабли из таблицы Outcomes, которые отсутствуют в таблице Ships ".

```
SELECT ship FROM Outcomes
EXCEPT
SELECT name FROM Ships;
```

Операция разности не является коммутативной, поэтому если переставить местами запросы, то мы получим решение совсем другой задачи: "Найти корабли из таблицы Ships, которые отсутствуют в таблице Outcomes".

ship
Bismarck
Duke of York
Fuso
Hood
King George V
Prince of Wales
Rodney
Schamhorst
West Virginia
Yamashiro

Выражение CASE – условный оператор языка SQL
Данный оператор позволяет осуществить проверку условий и возвратить в зависимости от выполнения того или иного условия тот или иной результат.

Оператор CASE имеет 2 формы:

Первая форма:	**Вторая форма:**
CASE WHEN условие_1 THEN возвращаемое_значение_1 ... WHEN условие_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END	CASE проверяемое_значение WHEN сравниваемое_значение_1 THEN возвращаемое_значение_1 ... WHEN сравниваемое_значение_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END

```
SELECT
  ID,Name,Salary,

  CASE
    WHEN Salary>=3000 THEN 'ЗП >= 3000'
    WHEN Salary>=2000 THEN '2000 <= ЗП < 3000'
    ELSE 'ЗП < 2000'
  END SalaryTypeWithELSE,

  CASE
    WHEN Salary>=3000 THEN 'ЗП >= 3000'
    WHEN Salary>=2000 THEN '2000 <= ЗП < 3000'
  END SalaryTypeWithoutELSE

FROM Employees
```

ID	Name	Salary	SalaryTypeWithELSE	SalaryTypeWithoutELSE
1000	Иванов И.И.	5000	ЗП >= 3000	ЗП >= 3000
1001	Петров П.П.	1500	ЗП < 2000	NULL
1002	Сидоров С.С.	2500	2000 <= ЗП < 3000	2000 <= ЗП < 3000
1003	Андреев А.А.	2000	2000 <= ЗП < 3000	2000 <= ЗП < 3000
1004	Николаев Н.Н.	1500	ЗП < 2000	NULL
1005	Александров А.А.	2000	2000 <= ЗП < 3000	2000 <= ЗП < 3000

WHEN-условия проверяются последовательно, сверху-вниз. При достижении первого удовлетворяющего условия дальнейшая проверка прерывается и возвращается значение, указанное после слова THEN, относящегося к данному блоку WHEN.
Если ни одно из WHEN-условий не выполняется, то возвращается значение, указанное после слова ELSE (что в данном случае означает «ИНАЧЕ ВЕРНИ …»).
Если ELSE-блок не указан и не выполняется ни одно WHEN-условие, то возвращается NULL.
И в первой, и во второй форме ELSE-блок идет в самом конце конструкции CASE, т.е. после всех WHEN-условий.

Допустим, на новый год решили премировать всех сотрудников и попросили вычислить сумму бонусов по следующей схеме:
Сотрудникам ИТ-отдела выдать по 15% от ЗП;
Сотрудникам Бухгалтерии по 10% от ЗП;
Всем остальным по 5% от ЗП.

```
SELECT
  ID,Name,Salary,DepartmentID,

  -- для наглядности выведем процент в виде строки
  CASE DepartmentID -- проверяемое значение
    WHEN 2 THEN '10%' -- 10% от ЗП выдать Бухгалтерам
    WHEN 3 THEN '15%' -- 15% от ЗП выдать ИТ-шникам
    ELSE '5%' -- всем остальным по 5%
  END NewYearBonusPercent,
FROM Employees
```

ID	Name	Salary	DepartmentID	NewYearBonusPercent
1000	Иванов И.И.	5000	1	5%
1001	Петров П.П.	1500	3	15%
1002	Сидоров С.С.	2500	2	10%
1003	Андреев А.А.	2000	3	15%
1004	Николаев Н.Н.	1500	3	15%
1005	Александров А.А.	2000	NULL	5%

Здесь делается последовательная проверка значения DepartmentID с WHEN-значениями. При достижении первого равенства DepartmentID с WHEN-значением, проверка прерывается и возвращается значение, указанное после слова THEN, относящегося к данному блоку WHEN.
Соответственно, значение блока ELSE возвращается в случае, если DepartmentID не совпал ни с одним WHEN-значением.
Если блок ELSE отсутствует, то в случае несовпадения DepartmentID ни с одним WHEN-значением будет возвращено NULL.
Вторую форму CASE несложно представить при помощи первой формы:

```
SELECT
  ID,Name,Salary,DepartmentID,

  CASE
    WHEN DepartmentID=2 THEN '10%' -- 10% от ЗП выдать Бухгалтерам
    WHEN DepartmentID=3 THEN '15%' -- 15% от ЗП выдать ИТ-шникам
    ELSE '5%' -- всем остальным по 5%
  END NewYearBonusPercent,
```

Если вы делаете первые шаги в SQL, то сосредоточьтесь в первую очередь, именно на изучении базовых конструкций, т.к. владея базой, все остальное вам понять будет гораздо легче, и к тому же самостоятельно.

Список литературы

1. Дейт К.Дж. Введение в системы баз данных, 6-е издание. - К.; М.; СПб.: Издательский дом "Вильямс", 2000
2. Ульман Дж.Д., Уидом Дж. Введение в системы баз данных. - М.: Издательство "Лори", 2000
3. Muthusamy Anantha Kumar. SQL Server and Collation, 2004
4. Мартин Грабер. Справочное руководство по SQL. - М.: Издательство "Лори", 1997
5. Kalen Delaney. Inside Microsoft SQL Server 2005: The Storage Engine (Microsoft Press, 2006) ISBN 978-0735621053
6. Codd E.F. The Relational Model for Database Management Version 2. - Reading, Mass.: Addison-Wesley, 1989
7. Джо Селко. Программирование на SQL для профессионалов. - 2-е издание - М.: Издательство "Лори", 2004
8. Itzik Ben-Gan. Inside Microsoft SQL Server 2008: T-SQL Querying: Microsoft Press, 2009
9. https://habrahabr.ru/post/255825/