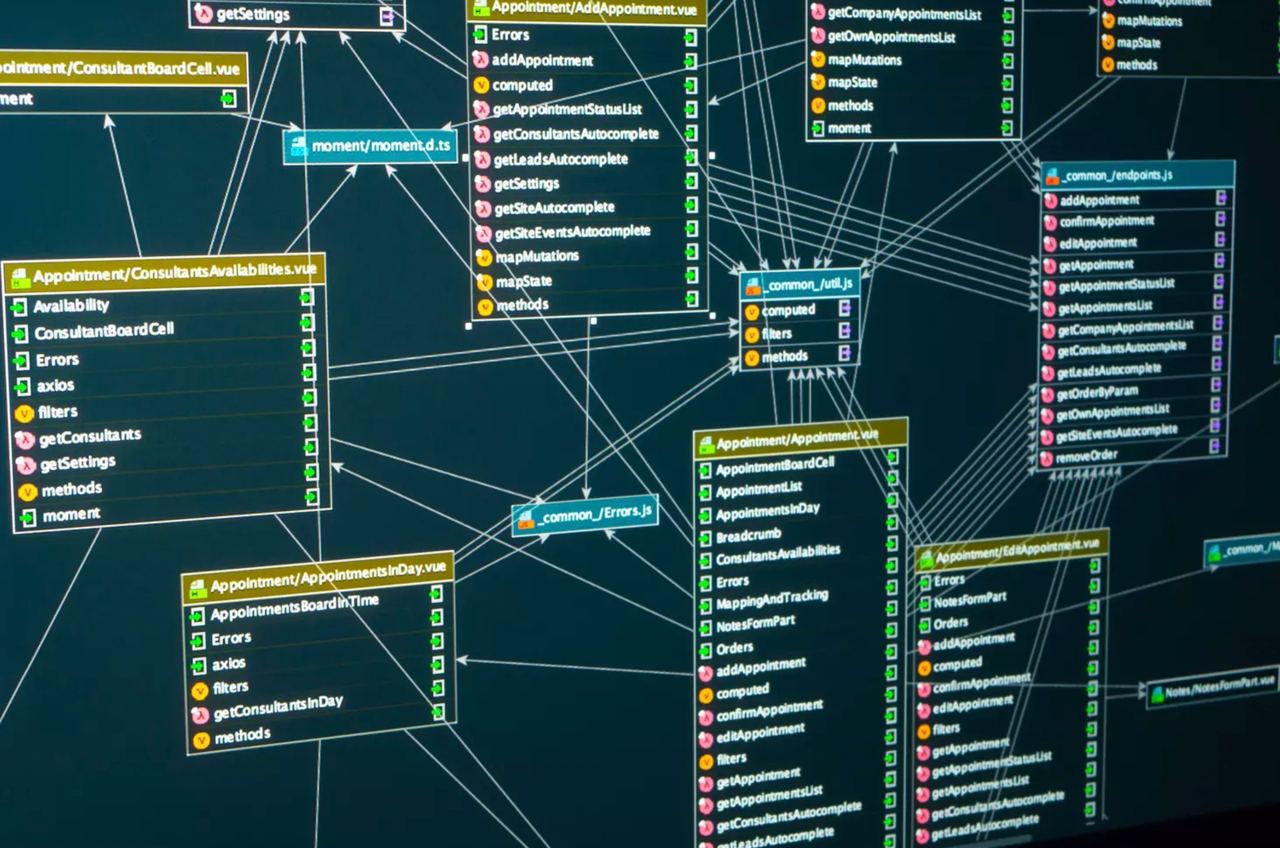
***<https://uproger.com/shpargalki-sql-2023/>***

**Подробная Шпаргалка SQL на 2023 год**



Руководство о всех SQL-запросах и примерах их использования.

Этот пост постоянно обновляется и дополняется, сохраняйте пост себе и делитесь ссыслкой с друзьями.

[@sqlhub](http://t.me/sqlhub) – задачи, курсы , разбор вопросов с собеседований SQL

**1. Запросы для поиска данных**

**SELECT: используется для выбора данных из базы данных**

* SELECT \* FROM table\_name;

**DISTINCT: отфильтровывает повторяющиеся значения и возвращает строки указанного столбца**

* SELECT DISTINCT column\_name;

**WHERE: используется для фильтрации записей/строк**

* SELECT column1, column2 FROM table\_name WHERE condition;
* SELECT \* FROM table\_name WHERE condition1 AND condition2;
* SELECT \* FROM table\_name WHERE condition1 OR condition2;
* SELECT \* FROM table\_name WHERE NOT condition;
* SELECT \* FROM table\_name WHERE condition1 AND (condition2 OR condition3);
* SELECT \* FROM table\_name WHERE EXISTS (SELECT column\_name FROM table\_name WHERE condition);

**ORDER BY: используется для сортировки набора результатов в порядке возрастания или убывания**

* SELECT \* FROM table\_name ORDER BY column;
* SELECT \* FROM table\_name ORDER BY column DESC;
* SELECT \* FROM table\_name ORDER BY column1 ASC, column2 DESC;

**SELECT TOP: используется для указания количества записей, возвращаемых из верхней части таблицы**

* SELECT TOP number columns\_names FROM table\_name WHERE condition;
* SELECT TOP percent columns\_names FROM table\_name WHERE condition;
* Not all database systems support SELECT TOP. The MySQL equivalent is the LIMIT clause
* SELECT column\_names FROM table\_name LIMIT offset, count;

**LIKE: оператор, используемый в предложении WHERE для поиска определенного шаблона в столбце**

WHERE CustomerName LIKE ‘a%’ Находит любые значения, которые начинаются с “a”

WHERE CustomerName LIKE ‘%a’ Находит любые значения, которые заканчиваются на “a”

WHERE CustomerName LIKE ‘%or%’ Находит любые значения, которые имеют “or” в любой позиции

WHERE CustomerName LIKE ‘\_r%’ Находит любые значения, имеющие букву “r” во второй позиции

WHERE CustomerName LIKE ‘a\_\_%’ Находит любые значения, начинающиеся с буквы “a” и имеющие длину не менее 3 символов

WHERE ContactName LIKE ‘a%o’ Находит любые значения, которые начинаются с “a” и заканчиваются “о”

**IN: оператор, который позволяет указать несколько значений в предложении WHERE**

* по сути, оператор IN является сокращением для нескольких условий OR
* SELECT column\_names FROM table\_name WHERE column\_name IN (value1, value2, …);
* SELECT column\_names FROM table\_name WHERE column\_name IN (SELECT STATEMENT);

**BETWEEN: оператор выбирает значения в заданном диапазоне включительно**

* SELECT column\_names FROM table\_name WHERE column\_name BETWEEN value1 AND value2;
* SELECT \* FROM Products WHERE (column\_name BETWEEN value1 AND value2) AND NOT column\_name2 IN (value3, value4);
* SELECT \* FROM Products WHERE column\_name BETWEEN #01/07/1999# AND #03/12/1999#;

**NULL**

* SELECT \* FROM table\_name WHERE column\_name IS NULL;
* SELECT \* FROM table\_name WHERE column\_name IS NOT NULL;

**AS: используются для присвоения временного имени таблице или столбцу**

* SELECT column\_name AS alias\_name FROM table\_name;
* SELECT column\_name FROM table\_name AS alias\_name;
* SELECT column\_name AS alias\_name1, column\_name2 AS alias\_name2;
* SELECT column\_name1, column\_name2 + ‘, ‘ + column\_name3 AS alias\_name;

**UNION: используется в SQL для объединения набора результатов двух или более операторов SELECT**

* Каждый оператор SELECT в UNION должен иметь одинаковое количество столбцов.
* Столбцы должны иметь похожие типы данных
* Столбцы в каждом операторе SELECT также должны быть в том же порядке.
* SELECT columns\_names FROM table1 UNION SELECT column\_name FROM table2;
* Оператор UNION выбирает только отдельные значения, UNION ALL разрешает дубликаты

**INTERSECT: используется для возврата записей, общих для двух операторов SELECT.**

* Обычно используется так же, как UNION
* SELECT columns\_names FROM table1 INTERSECT SELECT column\_name FROM table2;

**EXCEPT: оператор, используемый для возврата всех записей в первом операторе SELECT, которые не найдены во втором операторе SELECT.**

* Обычно используется так же, как UNION выше.
* SELECT columns\_names FROM table1 EXCEPT SELECT column\_name FROM table2;

**ANY|ALL: оператор, используемый для проверки условий подзапроса, используемых в предложениях WHERE или HAVING.**

* Оператор ANY возвращает значение true, если какие-либо значения подзапроса соответствуют условию.
* Оператор ALL возвращает значение true, если все значения подзапроса соответствуют условию
* SELECT columns\_names FROM table1 WHERE column\_name operator (ANY|ALL) (SELECT column\_name FROM table\_name WHERE condition);

**GROUP BY: оператор, часто используемый с агрегатными функциями (COUNT, MAX, MIN, SUM, AVG) для группировки набора результатов по одному или нескольким столбцам.**

* SELECT column\_name1, COUNT(column\_name2) FROM table\_name WHERE condition GROUP BY column\_name1 ORDER BY COUNT(column\_name2) DESC;

**HAVING: Оператор SQL HAVING является указателем на результат выполнения агрегатных функций. Агрегатной функцией в языке SQL называется функция, возвращающая какое-либо одно значение по набору значений столбца. Такими функциями являются:**[**SQL COUNT()**](http://2sql.ru/functions/sql-count/)**,**[**SQL MIN()**](http://2sql.ru/functions/sql-min/)**,**[**SQL MAX()**](http://2sql.ru/functions/sql-max/)**,**[**SQL AVG()**](http://2sql.ru/functions/sql-avg/)**,**[**SQL SUM()**](http://2sql.ru/functions/sql-sum/)**.**

* SELECT COUNT(column\_name1), column\_name2 FROM table GROUP BY column\_name2 HAVING COUNT(column\_name1) > 5;

**WITH: запрос, который часто используется для извлечения иерархических данных**

* WITH RECURSIVE cte AS (  
    SELECT c0.\* FROM categories AS c0 WHERE id = 1 # Starting point  
    UNION ALL  
    SELECT c1.\* FROM categories AS c1 JOIN cte ON c1.parent\_category\_id = cte.id  
  )  
  SELECT \*  
  FROM cte

**2. Запросы для модификации данных**

**INSERT INTO: используется для вставки новых записей/строк в таблицу**

* INSERT INTO table\_name (column1, column2) VALUES (value1, value2);
* INSERT INTO table\_name VALUES (value1, value2 …);

**UPDATE: используется для изменения существующих записей в таблице**

* UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;
* UPDATE table\_name SET column\_name = value;

**DELETE: используется для удаления существующих записей/строк в таблице**

* DELETE FROM table\_name WHERE condition;
* DELETE \* FROM table\_name;

**3. Считаем количество записей**

**COUNT: возвращает количество вхождений**

* SELECT COUNT (DISTINCT column\_name);

**MIN() and MAX(): MIN() и MAX(): возвращает наименьшее/наибольшее значение выбранного столбца.**

* SELECT MIN (column\_names) FROM table\_name WHERE condition;
* SELECT MAX (column\_names) FROM table\_name WHERE condition;

**AVG(): возвращает среднее значение числового столбца**

* SELECT AVG (column\_name) FROM table\_name WHERE condition;

**SUM(): возвращает общую сумму числового столбца**

* SELECT SUM (column\_name) FROM table\_name WHERE condition;

**4. Join запросы SQL**

**INNER JOIN: возвращает записи, имеющие совпадающее значение в обеих таблицах**

* SELECT column\_names FROM table1 INNER JOIN table2 ON table1.column\_name=table2.column\_name;
* SELECT table1.column\_name1, table2.column\_name2, table3.column\_name3 FROM ((table1 INNER JOIN table2 ON relationship) INNER JOIN table3 ON relationship);

**LEFT (OUTER) JOIN: возвращает все записи из левой таблицы (таблица1) и соответствующие записи из правой таблицы (таблица2)**

* SELECT column\_names FROM table1 LEFT JOIN table2 ON table1.column\_name=table2.column\_name;

**RIGHT (OUTER) JOIN: возвращает все записи из правой таблицы (таблица2) и соответствующие записи из левой таблицы (таблица1)**

* SELECT column\_names FROM table1 RIGHT JOIN table2 ON table1.column\_name=table2.column\_name;

**FULL (OUTER) JOIN**

* SELECT column\_names FROM table1 FULL OUTER JOIN table2 ON table1.column\_name=table2.column\_name;

**Self JOIN: обычные джоины**

* SELECT column\_names FROM table1 T1, table1 T2 WHERE condition;

**5. Запросы на просмотр**

**CREATE: create a view**

* CREATE VIEW view\_name AS SELECT column1, column2 FROM table\_name WHERE condition;

**SELECT: retrieve a view**

* SELECT \* FROM view\_name;

**DROP: drop a view**

* DROP VIEW view\_name;

**6. Altering запросы**

**ADD: add a column**

* ALTER TABLE table\_name ADD column\_name column\_definition;

**MODIFY: change data type of column**

* ALTER TABLE table\_name MODIFY column\_name column\_type;

**DROP: delete a column**

* ALTER TABLE table\_name DROP COLUMN column\_name;

**7. Создание таблицы sql**

**CREATE: create a table**

* CREATE TABLE table\_name (  
  column1 datatype,  
  column2 datatype,  
  column3 datatype,  
  column4 datatype,  
  );

[**SQL. Обобщенное табличное выражение**](http://t.me/sqlhub)

WITH CTE\_Name (column1, column2)

AS

-- Define the CTE query.

(

SELECT column1, column2

FROM Table1

WHERE column1 IS NOT NULL

)

-- Define the outer query referencing the CTE name.

SELECT \*

FROM CTE\_Name

GROUP BY column1, column2

ORDER BY column1, column2;

GO

star\_border

Поиск текста в SP

SET QUOTED\_IDENTIFIER ON

SET ANSI\_NULLS ON

GO

CREATE PROCEDURE [dbo].[Find\_Text\_In\_SP]

@StringToSearch VARCHAR(100),

@StringToSearch2 VARCHAR(100) = '',

@StringToSearch3 VARCHAR(100) = '',

@Name VARCHAR(100) = ''

AS

SET @StringToSearch = '%' +@StringToSearch + '%'

SET @StringToSearch2 = '%' +@StringToSearch2 + '%'

SET @StringToSearch3 = '%' +@StringToSearch3 + '%'

SET @Name = '%' +@Name + '%'

SELECT ROUTINE\_NAME, LEN(OBJECT\_DEFINITION(OBJECT\_ID(ROUTINE\_NAME))) AS SP\_Length

FROM INFORMATION\_SCHEMA.ROUTINES

WHERE OBJECT\_DEFINITION(OBJECT\_ID(ROUTINE\_NAME)) LIKE @stringtosearch

AND OBJECT\_DEFINITION(OBJECT\_ID(ROUTINE\_NAME)) LIKE @StringToSearch2

AND OBJECT\_DEFINITION(OBJECT\_ID(ROUTINE\_NAME)) LIKE @StringToSearch3

AND (ROUTINE\_TYPE='PROCEDURE' OR ROUTINE\_TYPE='FUNCTION')

AND ROUTINE\_NAME LIKE @Name

ORDER BY routine\_name

GO

**OFFSET FETCH**

Пропустить первые 10 строк из отсортированного набора результатов и вернуть остальные строки.

SELECT column1, column2 FROM table\_name ORDER BY column1 OFFSET 10 ROWS;

- Пропустить первые 10 строк из отсортированного набора результатов и вернуть следующие 5 строк.

SELECT column1, column2 FROM table\_name ORDER BY column1 OFFSET 10 ROWS FETCH NEXT 5 ROWS ONLY;

**Переименование a Table**

exec sp\_rename '[schema.old\_table\_name]', 'new\_table\_name'

**Переименование a Column**

exec sp\_rename 'table\_name.[oldColumName]' , 'newColumName', 'COLUMN'

**SCOPE\_IDENTITY**

-- returns the last IDENTITY value inserted into an IDENTITY column in the same scope

-- returns the last identity value generated for any table in the current session and the current scope

-- A scope is a module; a Stored Procedure, trigger, function, or batch

SELECT SCOPE\_IDENTITY()

**НАЙТИ, КАКОЙ ТАБЛИЦЕ ПРИНАДЛЕЖИТ ОГРАНИЧЕНИЕ**

SELECT

OBJECT\_NAME(o.parent\_object\_id)

FROM

sys.objects o

WHERE

o.name = 'MyConstraintName' AND o.parent\_object\_id <> 0

**TRY-CATCH**

BEGIN TRY

BEGIN TRANSACTION

-- Do something here

COMMIT TRANSACTION

END TRY

BEGIN CATCH

DECLARE

@ErrorMessage NVARCHAR(4000),

@ErrorSeverity INT,

@ErrorState INT;

SELECT

@ErrorMessage = ERROR\_MESSAGE(),

@ErrorSeverity = ERROR\_SEVERITY(),

@ErrorState = ERROR\_STATE();

RAISERROR (

@ErrorMessage,

@ErrorSeverity,

@ErrorState

);

ROLLBACK TRANSACTION

END CATCH

**Условия с переменными WHERE CLAUSE**

-- using '=' operator

WHERE Column = IIF(@Variable IS NULL ,@Variable, Column)

-- using 'LIKE, IN, etc.'

WHERE (@Variable IS NULL OR Column LIKE '%' + @Variable + '%' )

**ВСТАВИТЬ РАЗДЕЛЕННУЮ ЗАПЯТОЙ СТРОКУ В ТАБЛИЦУ**

DECLARE @String = '1, 4, 3'

DECLARE @Tbl TABLE(ID INT);

INSERT INTO @Tbl

(

ID

)

(SELECT value

FROM STRING\_SPLIT(@String, ',')

WHERE RTRIM(value) <> '');

**UPDATE WITH JOIN**

UPDATE Table1

SET Table1.Column = B.Column

FROM Table1 A

INNER JOIN Table2 B

ON A.ID = B.ID

**DELETE WITH JOIN**

DELETE A

FROM Table1 A

INNER JOIN Table2 B

ON B.Id = A.Id

WHERE A.Column = 1 AND B.Column = 2

**UPDATE/INSERT IDENTITY COLUMN**

SET IDENTITY\_INSERT YourTable ON

-- UPDATE/INSERT STATEMENT HERE

SET IDENTITY\_INSERT YourTable OFF

**Находим Foreign Key ограничения ссылок таблицы**

SELECT

OBJECT\_NAME(f.parent\_object\_id) TableName,

COL\_NAME(fc.parent\_object\_id,fc.parent\_column\_id) ColName

FROM

sys.foreign\_keys AS f

INNER JOIN

sys.foreign\_key\_columns AS fc

ON f.OBJECT\_ID = fc.constraint\_object\_id

INNER JOIN

sys.tables t

ON t.OBJECT\_ID = fc.referenced\_object\_id

WHERE

OBJECT\_NAME (f.referenced\_object\_id) = 'Table\_Name'

**Парсим JSON файл в таблицу**

-- JSON Data sample:

-- {

-- "label": "test ",

-- "value": 1

-- },

-- {

-- "label": "test2 ",

-- "value": 2

-- }

DECLARE @tbl TABLE (id INT, label VARCHAR(500));

DECLARE @json VARCHAR(max);

SELECT @json = BulkColumn

FROM OPENROWSET (BULK 'C:\jsonFile.json', SINGLE\_CLOB) as j

INSERT INTO @tbl (id, label)

SELECT [value], label

FROM OPENJSON(@json)

WITH ([value] int,

label nvarchar(max))

SELECT \* FROM @tbl

**Добавляем FK в существующую колонку**

ALTER TABLE [Table1]

ADD CONSTRAINT FK\_Table2\_Id FOREIGN KEY (Table1\_Id)

REFERENCES Table2(Table2\_Id);

**Список всех пользовательских функций по типу**

SELECT [Name], [Definition], [Type\_desc]

FROM sys.sql\_modules m

INNER JOIN sys.objects o

ON m.object\_id=o.object\_id

WHERE [Type\_desc] like '%function%'

**Обновляем и изменяем часть строки**

UPDATE dbo.[Table]

SET Value = REPLACE(Value, '123\', '')

WHERE ID <=4

**Генерируем случайные INT SQL**

---- Create the variables for the random number generation

DECLARE @Random INT;

DECLARE @Upper INT;

DECLARE @Lower INT

---- This will create a random number between 1 and 999

SET @Lower = 1 ---- The lowest random number

SET @Upper = 999 ---- The highest random number

SELECT @Random = ROUND(((@Upper - @Lower -1) \* RAND() + @Lower), 0)

SELECT @Random

**Создаем случайные ДАТЫ между двумя диапазонами**

DECLARE @FromDate DATE = '2019-09-01';

DECLARE @ToDate DATE = '2019-12-31';

SELECT DATEADD(DAY, RAND(CHECKSUM(NEWID()))\*(1+DATEDIFF(DAY, @FromDate, @ToDate)), @FromDate)

**Получаем Список таблиц**

SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE='BASE TABLE'

**Проверить, существует ли таблица в базе данных**

IF EXISTS(SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA = 'dbo' AND TABLE\_NAME = 'Table')

BEGIN

-- exists

END

**Сгенерировать 6-значный уникальный номер**

SELECT LEFT(CAST(RAND()\*1000000000+999999 AS INT),6) AS OTP

**Ищем table name**

SELECT \* FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_NAME LIKE '%%'

**Поиск между двумя датами**

--convert to date to ignore time

SELECT \* FROM Table T

WHERE CONVERT(DATE,T.DateColumn) BETWEEN COALESCE(CONVERT(DATE,@DateFrom), CONVERT(DATE,T.DateColumn)) AND COALESCE(CONVERT(DATE,@DateTo), CONVERT(DATE,T.DateColumn))

**Формат дат**

--Output: 21/03/2022

SELECT FORMAT (getdate(), 'dd/MM/yyyy ') as date

--Output: 21/03/2022, 11:36:14

SELECT FORMAT (getdate(), 'dd/MM/yyyy, hh:mm:ss ') as date

--Output: Wednesday, March, 2022

SELECT FORMAT (getdate(), 'dddd, MMMM, yyyy') as date

--Output: Mar 21 2022

SELECT FORMAT (getdate(), 'MMM dd yyyy') as date

--Output: 03.21.22

SELECT FORMAT (getdate(), 'MM.dd.yy') as date

--Output: 03-21-22

SELECT FORMAT (getdate(), 'MM-dd-yy') as date

--Output: 11:36:14 AM

SELECT FORMAT (getdate(), 'hh:mm:ss tt') as date

--Output: 03/21/2022

SELECT FORMAT (getdate(), 'd','us') as date

**Триггеры**

create trigger t1 on table1

after insert

as

begin

insert into Audit

(Column)

select 'Insert New Row with Key' + cast(t.Id as nvarchar(10)) + 'in table1'

from table1 t where Id IN (select Id from inserted)

end

go

**Найти все таблицы, содержащие столбец с указанным именем**

SELECT c.name AS 'ColumnName'

,t.name AS 'TableName'

FROM sys.columns c

JOIN sys.tables t ON c.object\_id = t.object\_id

WHERE c.name LIKE '%COLUMN\_NAME%'

ORDER BY TableName

,ColumnName;

**Скрипт для создания отбрасываемых всех таблиц с префиксом**

SELECT 'DROP TABLE ' + TABLE\_NAME + ''

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_NAME LIKE 'PREFIX\_%'

**Скрипт для изменения таблиц, чтобы удалить все ограничения**

DECLARE @SQL varchar(4000)=''

SELECT @SQL =

@SQL + 'ALTER TABLE ' + s.name+'.'+t.name + ' DROP CONSTRAINT [' + RTRIM(f.name) +'];' + CHAR(13)

FROM sys.Tables t

INNER JOIN sys.foreign\_keys f ON f.parent\_object\_id = t.object\_id

INNER JOIN sys.schemas s ON s.schema\_id = f.schema\_id

WHERE t.name LIKE 'PREFIX\_%'

--EXEC (@SQL)

PRINT @SQL

**Cursor**

**Перебрать набор данных**

-- two variables to hold product name and list price (gonna be used on the loop)

DECLARE

@product\_name VARCHAR(MAX),

@list\_price DECIMAL;

--defines the result set for the cursor

DECLARE cursor\_product CURSOR

FOR SELECT

product\_name,

list\_price

FROM

dbo.products;

-- open cursor

OPEN cursor\_product;

--fetch a row from the cursor into one or more variables

FETCH NEXT FROM cursor\_product INTO

@product\_name,

@list\_price;

-- loop through the cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

-- use current product\_name and list\_price from current index of the cursor in the loop

PRINT @product\_name + CAST(@list\_price AS varchar);

-- fetch next row from the cursor

FETCH NEXT FROM cursor\_product INTO

@product\_name,

@list\_price;

END;

-- close cursor

CLOSE cursor\_product;

-- deallocate the cursor to release it

DEALLOCATE cursor\_product;

Join Multiple Tables

SELECT

comments.body,

posts.title,

users.first\_name,

users.last\_name

FROM comments

INNER JOIN posts on posts.id = comments.post\_id

INNER JOIN users on users.id = comments.user\_id

ORDER BY posts.title;

Aggregate Functions

SELECT COUNT(id) FROM users;

SELECT MAX(age) FROM users;

SELECT MIN(age) FROM users;

SELECT SUM(age) FROM users;

SELECT UCASE(first\_name), LCASE(last\_name) FROM users;

Group By

SELECT age, COUNT(age) FROM users GROUP BY age;

SELECT age, COUNT(age) FROM users WHERE age > 20 GROUP BY age;

SELECT age, COUNT(age) FROM users GROUP BY age HAVING count(age) >=2;