Contents

[Useful website Links 1](#_Toc452408386)

[Commonly used constraints available in SQL? 1](#_Toc452408387)

[Alias Syntax 1](#_Toc452408388)

[Like command 1](#_Toc452408389)

[ORDER BY 2](#_Toc452408390)

[GROUP BY 2](#_Toc452408391)

[Having Clause 2](#_Toc452408392)

[SQL Join Types 3](#_Toc452408393)

[UNIONS CLAUSE 3](#_Toc452408394)

[UNIONS CLAUSE 4](#_Toc452408395)

[Creating Views 4](#_Toc452408396)

[Sub Queries 5](#_Toc452408397)

### Useful website Links

* <http://www.tutorialspoint.com/sql/>
* <http://www.interviewquestionspdf.com/2014/07/sql-queries-interview-questions-answers.html>
* <http://www.interviewquestionspdf.com/2014/12/sql-server-interview-questions-and.html>

### Commonly used constraints available in SQL?

* [NOT NULL Constraint](http://www.tutorialspoint.com/sql/sql-not-null.htm): Ensures that a column cannot have NULL value.
* [DEFAULT Constraint](http://www.tutorialspoint.com/sql/sql-default.htm): Provides a default value for a column when none is specified.
* [UNIQUE Constraint](http://www.tutorialspoint.com/sql/sql-unique.htm): Ensures that all values in a column are different.
* [PRIMARY Key](http://www.tutorialspoint.com/sql/sql-primary-key.htm): Uniquely identified each rows/records in a database table.
  + Primary keys must contain UNIQUE values.
  + A primary key column cannot contain NULL values.
  + CREATE TABLE Persons  
    (  
    P\_Id int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255),  
    **PRIMARY KEY** (P\_Id)  
    )
* [FOREIGN Key](http://www.tutorialspoint.com/sql/sql-foreign-key.htm): A FOREIGN KEY in one table points to a PRIMARY KEY in another table.
  + The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
  + The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
  + CREATE TABLE Orders  
    (  
    O\_Id int NOT NULL,  
    OrderNo int NOT NULL,  
    P\_Id int,  
    PRIMARY KEY (O\_Id),  
    **FOREIGN KEY** (P\_Id) REFERENCES Persons(P\_Id)  
    )
* [CHECK Constraint](http://www.tutorialspoint.com/sql/sql-check.htm): The CHECK constraint ensures that all values in a column satisfy certain conditions.
* [INDEX](http://www.tutorialspoint.com/sql/sql-index.htm): Use to create and retrieve data from the database very quickly.

### Alias Syntax

You can rename a table or a column temporarily by giving another name known as alias.

* SELECT column1, column2 FROM table\_name AS alias\_name WHERE [condition];

### Like command

|  |  |
| --- | --- |
| **Statement** | **Description** |
| WHERE SALARY LIKE '200%' | Finds any values that start with 200 |
| WHERE SALARY LIKE '%200%' | Finds any values that have 200 in any position |
| WHERE SALARY LIKE '\_00%' | Finds any values that have 00 in the second and third positions |
| WHERE SALARY LIKE '2\_%\_%' | Finds any values that start with 2 and are at least 3 characters in length |
| WHERE SALARY LIKE '%2' | Finds any values that end with 2 |
| WHERE SALARY LIKE '\_2%3' | Finds any values that have a 2 in the second position and end with a 3 |
| WHERE SALARY LIKE '2\_\_\_3' | Finds any values in a five-digit number that start with 2 and end with 3 |

### ORDER BY

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

* SELECT column-list FROM table\_name [WHERE condition] [ORDER BY column1, column2, .. columnN] [ASC | DESC];
* SELECT \* FROM CUSTOMERS ORDER BY NAME, SALARY;

### GROUP BY

The SQL **GROUP BY**clause is used in collaboration with the SELECT statement to arrange identical data into groups.The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

SELECT column1, column2

FROM table\_name

WHERE [ conditions ]

GROUP BY column1, column2

ORDER BY column1, column2

SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;

### Having Clause

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created

* SELECT FROM WHERE GROUP BY HAVING ORDER BY by the GROUP BY clause.
* SELECT ID, NAME, AGE, ADDRESS FROM CUSTOMERS GROUP BY age HAVING COUNT(age) >= 2;
* select count(\*) , District from address group by District having District like 'Z%'
* select count(\*) , District from address group by District **where** District like 'Z%' INVALID Command,Error

### SQL Join Types

SELECT ID, NAME, AGE, AMOUNT FROM CUSTOMERS, ORDERS WHERE CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

There are different types of joins available in SQL:

* INNER JOIN (a.k.a. “simple join”): Returns all rows for which there is at least one match in BOTH tables. *This is the default type of join if no specific* JOIN *type is specified.*

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
INNER JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

* LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table, and the matched rows from the right table; i.e., the results will contain *all* records from the left table, even if the JOIN condition doesn’t find any matching records in the right table. This means that if the ON clause doesn’t match any records in the right table, the JOIN will still return a row in the result for that record in the left table, but with NULL in each column from the right table.

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

* RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table, and the matched rows from the left table. This is the exact opposite of a LEFT JOIN; i.e., the results will contain *all* records from the right table, even if the JOIN condition doesn’t find any matching records in the left table. This means that if the ON clause doesn’t match any records in the left table, the JOIN will still return a row in the result for that record in the right table, but with NULL in each column from the left table.

SELECT Orders.OrderID, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID  
ORDER BY Orders.OrderID;

* FULL JOIN (or FULL OUTER JOIN): Returns all rows for which there is a match in EITHER of the tables. Conceptually, a FULL JOIN combines the effect of applying both a LEFT JOIN and a RIGHT JOIN; i.e., its result set is equivalent to performing a UNION of the results of left and right outer queries.

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

* CROSS JOIN: Returns all records where each row from the first table is combined with each row from the second table (i.e., returns the Cartesian product of the sets of rows from the joined tables). Note that a CROSS JOIN can either be specified using the CROSS JOIN syntax (“explicit join notation”) or (b) listing the tables in the FROM clause separated by commas without using a WHERE clause to supply join criteria (“implicit join notation”).

### UNIONS CLAUSE

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows. To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]

UNION

SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

UNION

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

### UNIONS CLAUSE

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

* CREATE INDEX index\_name ON table\_name;
* CREATE INDEX index\_name ON table\_name (column\_name);
* CREATE UNIQUE INDEX index\_name on table\_name (column\_name);
* CREATE INDEX index\_name on table\_name (column1, column2);

Indexes should not be used on small tables.

* Tables that have frequent, large batch update or insert operations.
* Indexes should not be used on columns that contain a high number of NULL values.
* Columns that are frequently manipulated should not be indexed.

### Creating Views

* CREATE VIEW view\_name AS SELECT column1, column2.....FROM table\_name WHERE [condition];
* CREATE VIEW CUSTOMERS\_VIEW AS SELECT name, age FROM CUSTOMERS;

### Sub Queries

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

**SELECT Statement**

SELECT \* FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500);

**INSERT Statement**

INSERT INTO CUSTOMERS\_BKP SELECT \* FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS) ;

**UPDATE Statement**

UPDATE CUSTOMERS SET SALARY = SALARY \* 0.25 WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP WHERE AGE >= 27 );

**DELETE Statement**

DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP WHERE AGE > 27 );