









- [Go to your profile](#)
- [Hire a Top developer](#)
- [Apply as a developer](#)
- [Log in](#)

- [Top 3%](#)
- [Why](#)
- [Clients](#)
- [Enterprise](#)
- [Community](#)
- [Blog](#)
- [About Us](#)
- [Go to your profile](#)
- [Hire a Top developer](#)
- [Apply as a developer](#)
- [Log in](#)
- - Questions?
 - [Contact Us](#)
 - 
 - 
 - 

- Questions?
- [Contact Us](#)
- 
- 
- 

[Hire a Top developer](#)

41 Essential SQL Interview Questions *

- 1.3Kshares



[Submit an interview question](#)[Submit a question](#)

Looking for [freelance SQL jobs](#)? Design your lifestyle as a SQL developer with Toptal.



What does UNION do? What is the difference between UNION and UNION ALL?

[View the answer](#) → [Hide answer](#)



UNION merges the contents of two structurally-compatible tables into a single combined table. The difference between UNION and UNION ALL is that UNION will omit duplicate records whereas UNION ALL will include duplicate records.

It is important to note that the performance of UNION ALL will typically be better than UNION, since UNION requires the server to do the additional work of removing any duplicates. So, in cases where it is certain that there will not be any duplicates, or where having duplicates is not a problem, use of UNION ALL would be recommended for performance reasons.



List and explain the different types of JOIN clauses supported in ANSI-standard SQL.

[View the answer](#) → [Hide answer](#)



ANSI-standard SQL specifies five types of JOIN clauses as follows:

- **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.*
- **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.
- **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.
- **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.

- **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”).



Given the following tables:

```
sql> SELECT * FROM runners;
```

id	name
1	John Doe
2	Jane Doe
3	Alice Jones
4	Bobby Louis
5	Lisa Romero

```
sql> SELECT * FROM races;
```

id	event	winner_id
1	100 meter dash	2
2	500 meter dash	3
3	cross-country	2
4	triathlon	NULL

What will be the result of the query below?

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)
```

Explain your answer and also provide an alternative version of this query that will avoid the issue that it exposes.

View the answer → Hide answer



Surprisingly, given the sample data provided, the result of this query will be an empty set. The reason for this is as follows: If the set being evaluated by the SQL **NOT IN** condition contains *any* values that are null, then the outer query here will return an empty set, even if there are many runner ids that match winner_ids in the races table.

Knowing this, a query that avoids this issue would be as follows:

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races WHERE winner_id IS NOT null)
```

Note, this is assuming the standard SQL behavior that you get without modifying the default **ANSI_NULLS** setting.

Apply to Join Toptal's Developer Network and enjoy reliable, steady, remote freelance SQL jobs.

[Apply to Toptal's Freelance SQL jobs](https://www.toptal.com/sql/jobs)



Given two tables created and populated as follows:

```
CREATE TABLE dbo.envelope(id int, user_id int);
CREATE TABLE dbo.docs(idnum int, pageseq int, doctext varchar(100));
```

```
INSERT INTO dbo.envelope VALUES
```

```
(1,1),
(2,2),
(3,3);
```

```
INSERT INTO dbo.docs(idnum,pageseq) VALUES
```

```
(1,5),
(2,6),
(null,0);
```

What will the result be from the following query:

```
UPDATE docs SET doctext=pageseq FROM docs INNER JOIN envelope ON envelope.id=docs.idnum
WHERE EXISTS (
    SELECT 1 FROM dbo.docs
    WHERE id=envelope.id
);
```

Explain your answer.

View the answer → Hide answer



The result of the query will be as follows:

idnum	pageseq	doctext
1	5	5
2	6	6
NULL	0	NULL

The EXISTS clause in the above query is a red herring. It will *always* be true since ID is *not* a member of `dbo.docs`. As such, it will refer to the `envelope` table comparing itself to itself!

The `idnum` value of NULL will not be set since the join of NULL will not return a result when attempting a match with any value of `envelope`.



Given these contents of the Customers table:

Id	Name	ReferredBy
1	John Doe	NULL
2	Jane Smith	NULL
3	Anne Jenkins	2
4	Eric Branford	NULL
5	Pat Richards	1
6	Alice Barnes	2

Here is a query written to return the list of customers not referred by Jane Smith:

```
SELECT Name FROM Customers WHERE ReferredBy <> 2;
```

What will be the result of the query? Why? What would be a better way to write it?

View the answer → Hide answer



Although there are 4 customers not referred by Jane Smith (including Jane Smith herself), the query will only return one: Pat Richards. All the customers who were referred by nobody at all (and therefore have NULL in their ReferredBy column) don't show up. But certainly those customers weren't referred by Jane Smith, and certainly NULL is not equal to 2, so why didn't they show up?

SQL Server uses three-valued logic, which can be troublesome for programmers accustomed to the more satisfying two-valued logic (TRUE or FALSE) most programming languages use. In most languages, if you were presented with two predicates: ReferredBy = 2 and ReferredBy <> 2, you would expect one of them to be true and one of them to be false, given the same value of ReferredBy. In SQL Server, however, if ReferredBy is NULL, neither of them are true and neither of them are false. Anything compared to NULL evaluates to the third value in three-valued logic: UNKNOWN.

The query should be written in one of two ways:

```
SELECT Name FROM Customers WHERE ReferredBy IS NULL OR ReferredBy <> 2
```

...or:

```
SELECT Name FROM Customers WHERE ISNULL(ReferredBy, 0) <> 2; -- (Or COALESCE() )
```

Watch out for the following, though!

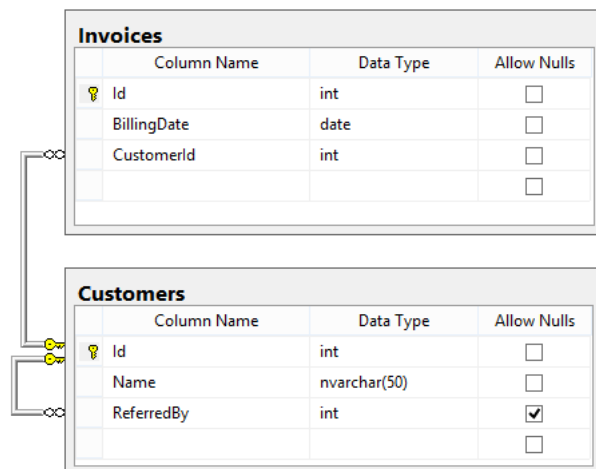
```
SELECT Name FROM Customers WHERE ReferredBy = NULL OR ReferredBy <> 2
```

This will return the same faulty set as the original. Why? We already covered that: Anything compared to NULL evaluates to the third value in the three-valued logic: UNKNOWN. That "anything" includes NULL itself! That's why SQL Server provides the IS NULL and IS NOT NULL operators to specifically check for NULL. Those particular operators will always evaluate to true or false.

Even if a candidate doesn't have a great amount of experience with SQL Server, diving into the intricacies of three-valued logic in general can give a good indication of whether they have the ability learn it quickly or whether they will struggle with it.



Considering the database schema displayed in the SQLServer-style diagram below, write a SQL query to return a list of all the invoices. For each invoice, show the Invoice ID, the billing date, the customer's name, and the name of the customer who referred that customer (if any). The list should be ordered by billing date.



View the answer → Hide answer



```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM Invoices i
JOIN Customers c ON i.CustomerId = c.Id
LEFT JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

This question simply tests the candidate's ability take a plain-English requirement and write a corresponding SQL query. There is nothing tricky in this one, it just covers the basics:

- Did the candidate remember to use a LEFT JOIN instead of an inner JOIN when joining the customer table for the referring customer name? If not, any invoices by customers not referred by somebody will be left out altogether.
- Did the candidate alias the tables in the JOIN? Most experienced T-SQL programmers always do this, because repeating the full table name each time it needs to be referenced gets tedious quickly. In this case, the query would actually break if at least the Customer table wasn't aliased, because it is referenced twice in different contexts (once as the table which contains the name of the invoiced customer, and once as the table which contains the name of the referring customer).
- Did the candidate disambiguate the Id and Name columns in the SELECT? Again, this is something most experienced programmers do automatically, whether or not there would be a conflict. And again, in this case there would be a conflict, so the query would break if the candidate neglected to do so.

Note that this query will not return Invoices that do not have an associated Customer. This may be the correct behavior for most cases (e.g., it is guaranteed that every Invoice is associated with a Customer, or unmatched Invoices are not of interest). However, in order to guarantee that all Invoices are returned no matter what, the Invoices table should be joined with Customers using LEFT JOIN:

```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM Invoices i
LEFT JOIN Customers c ON i.CustomerId = c.Id
LEFT JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```



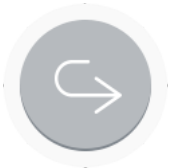
Assume a schema of Emp (Id, Name, DeptId) , Dept (Id, Name).

If there are 10 records in the Emp table and 5 records in the Dept table, how many rows will be displayed in the result of the following SQL query:

```
Select * From Emp, Dept
```

Explain your answer.

View the answer → Hide answer



The query will result in 50 rows as a “cartesian product” or “cross join”, which is the default whenever the ‘where’ clause is omitted.



Given two tables created as follows

```
create table test_a(id numeric);
create table test_b(id numeric);

insert into test_a(id) values
  (10),
  (20),
  (30),
  (40),
  (50);

insert into test_b(id) values
  (10),
  (30),
  (50);
```

Write a query to fetch values in table test_a that are and not in test_b **without** using the NOT keyword.

Note, Oracle does not support the above INSERT syntax, so you would need this instead:

```
insert into test_a(id) values (10);
insert into test_a(id) values (20);
insert into test_a(id) values (30);
```

```
insert into test_a(id) values (40);
insert into test_a(id) values (50);
insert into test_b(id) values (10);
insert into test_b(id) values (30);
insert into test_b(id) values (50);
```

View the answer → Hide answer



In SQL Server, PostgreSQL, and SQLite, this can be done using the [except](#) keyword as follows:

```
select * from test_a
except
select * from test_b;
```

In Oracle, the [minus](#) keyword is used instead. Note that if there are multiple columns, say ID and Name, the column should be explicitly stated in Oracle queries: `Select ID from test_a minus select ID from test_b`

MySQL does not support the except function. However, there is a standard SQL solution that works in all of the above engines, including MySQL:

```
select a.id
from test_a a
left join test_b b on a.id = b.id
where b.id is null;
```



Given a table TBL with a field Nbr that has rows with the following values:

1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1

Write a query to add 2 where Nbr is 0 and add 3 where Nbr is 1.

View the answer → Hide answer



This can be done as follows:

```
update TBL set Nbr = case when Nbr = 0 then Nbr+2 else Nbr+3 end;
```




Write a SQL query to find the 10th highest employee salary from an Employee table. Explain your answer.

(Note: You may assume that there are at least 10 records in the Employee table.)

View the answer → Hide answer



This can be done as follows:

```
SELECT TOP (1) Salary FROM
(
    SELECT DISTINCT TOP (10) Salary FROM Employee ORDER BY Salary DESC
) AS Emp ORDER BY Salary
```

This works as follows:

First, the `SELECT DISTINCT TOP (10) Salary FROM Employee ORDER BY Salary DESC` query will select the top 10 salaried employees in the table. However, those salaries will be listed in *descending* order. That was necessary for the first query to work, but now picking the top 1 from that list will give you the *highest* salary not the *10th highest* salary.

Therefore, the second query reorders the 10 records in *ascending* order (which the default sort order) and *then* selects the top record (which will now be the lowest of those 10 salaries).

Not all databases support the TOP keyword. For example, MySQL and PostgreSQL use the LIMIT keyword, as follows:

```
SELECT Salary FROM
(
    SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT 10
) AS Emp ORDER BY Salary LIMIT 1;
```

Or even more concisely, in MySQL this can be:

```
SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT 9,1;
```

And in PostgreSQL this can be:

```
SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT 1 OFFSET 9;
```



Write a SQL query using UNION ALL (**not** UNION) that uses the WHERE clause to eliminate duplicates. Why might you want to do this?

View the answer → Hide answer



You can avoid duplicates using UNION ALL and still run much faster than UNION DISTINCT (which is actually same as UNION) by running a query like this:

```
SELECT * FROM mytable WHERE a=X UNION ALL SELECT * FROM mytable WHERE b=Y AND a!=X
```

The key is the AND a!=X part. This gives you the benefits of the UNION (a.k.a., UNION DISTINCT) command, while avoiding much of its performance hit.



Given the following tables:

```
SELECT * FROM users;
```

user_id	username
1	John Doe
2	Jane Don
3	Alice Jones
4	Lisa Romero

```
SELECT * FROM training_details;
```

user_training_id	user_id	training_id	training_date
1	1	1	"2015-08-02"
2	2	1	"2015-08-03"
3	3	2	"2015-08-02"
4	4	2	"2015-08-04"
5	2	2	"2015-08-03"
6	1	1	"2015-08-02"
7	3	2	"2015-08-04"
8	4	3	"2015-08-03"
9	1	4	"2015-08-03"
10	3	1	"2015-08-02"
11	4	2	"2015-08-04"
12	3	2	"2015-08-02"
13	1	1	"2015-08-02"
14	4	3	"2015-08-03"

Write a query to to get the list of users who took the a training lesson more than once in the same day, grouped by user and training lesson, each ordered from the most recent lesson date to oldest date.

View the answer → Hide answer



```
SELECT
    u.user_id,
    username,
    training_id,
    training_date,
    count( user_training_id ) AS count
FROM users u JOIN training_details t ON t.user_id = u.user_id
GROUP BY u.user_id,
    username,
    training_id,
    training_date
HAVING count( user_training_id ) > 1
ORDER BY training_date DESC;
```

user_id	username	training_id	training_date	count
4	Lisa Romero	2	August, 04 2015 00:00:00	2
4	Lisa Romero	3	August, 03 2015 00:00:00	2
1	John Doe	1	August, 02 2015 00:00:00	3
3	Alice Jones	2	August, 02 2015 00:00:00	2



What is an execution plan? When would you use it? How would you view the execution plan?

View the answer → Hide answer



An execution plan is basically a road map that graphically or textually shows the data retrieval methods chosen by the SQL server's query optimizer for a stored procedure or ad hoc query. Execution plans are very useful for helping a developer understand and analyze the performance characteristics of a query or stored procedure, since the plan is used to execute the query or stored procedure.

In many SQL systems, a textual execution plan can be obtained using a keyword such as EXPLAIN, and visual representations can often be obtained as well. In Microsoft SQL Server, the Query Analyzer has an option called "Show Execution Plan" (located on the Query drop down menu). If this option is turned on, it will display query execution plans in a separate window when a query is run.



List and explain each of the ACID properties that collectively guarantee that database transactions are processed reliably.

[View the answer](#) → [Hide answer](#)



ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. They are defined as follows:

- **Atomicity.** Atomicity requires that each transaction be “all or nothing”: if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
- **Consistency.** The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.
- **Isolation.** The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method (i.e. if it uses strict - as opposed to relaxed - serializability), the effects of an incomplete transaction might not even be visible to another transaction.
- **Durability.** Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.



Given a table `dbo.users` where the column `user_id` is a unique numeric identifier, how can you efficiently select the first 100 odd `user_id` values from the table?

(Assume the table contains well over 100 records with odd `user_id` values.)

[View the answer](#) → [Hide answer](#)



```
SELECT TOP 100 user_id FROM dbo.users WHERE user_id % 2 = 1 ORDER BY user_id
```



How can you select all the even number records from a table? All the odd number records?

[View the answer](#) → [Hide answer](#)



To select all the **even** number records from a table:

```
Select * from table where id % 2 = 0
```

To select all the **odd** number records from a table:

```
Select * from table where id % 2 != 0
```



What are the NVL and the NVL2 functions in SQL? How do they differ?

[View the answer](#) → [Hide answer](#)



Both the NVL(exp1, exp2) and NVL2(exp1, exp2, exp3) functions check the value exp1 to see if it is null.

With the NVL(exp1, exp2) function, if exp1 is *not* null, then the value of exp1 is returned; otherwise, the value of exp2 is returned, but case to the same data type as that of exp1.

With the NVL2(exp1, exp2, exp3) function, if exp1 is *not* null, then exp2 is returned; otherwise, the value of exp3 is returned.



What is the difference between the RANK() and DENSE_RANK() functions? Provide an example.

[View the answer](#) → [Hide answer](#)



The *only* difference between the `RANK()` and `DENSE_RANK()` functions is in cases where there is a “tie”; i.e., in cases where multiple values in a set have the same ranking. In such cases, `RANK()` will assign non-consecutive “ranks” to the values in the set (resulting in gaps between the integer ranking values when there is a tie), whereas `DENSE_RANK()` will assign consecutive ranks to the values in the set (so there will be no gaps between the integer ranking values in the case of a tie).

For example, consider the set {25, 25, 50, 75, 75, 100}. For such a set, `RANK()` will return {1, 1, 3, 4, 4, 6} (note that the values 2 and 5 are skipped), whereas `DENSE_RANK()` will return {1, 1, 2, 3, 3, 4}.



What is the difference between the `WHERE` and `HAVING` clauses?

View the answer → Hide answer



When `GROUP BY` is not used, the `WHERE` and `HAVING` clauses are essentially equivalent.

However, when `GROUP BY` is used:

- The `WHERE` clause is used to filter records from a result. The filtering occurs before any groupings are made.
- The `HAVING` clause is used to filter values from a group (i.e., to check conditions after aggregation into groups has been performed).



Suppose we have a `Customer` table containing the following data:

CustomerID	CustomerName
1	Prashant Kaurav
2	Ashish Jha
3	Ankit Varma
4	Vineet Kumar
5	Rahul Kumar

Write a single SQL statement to concatenate all the customer names into the following single semicolon-separated string:

Prashant Kaurav; Ashish Jha; Ankit Varma; Vineet Kumar; Rahul Kumar

View the answer → Hide answer



```
SELECT CustomerName+ ' ; '
FROM Customer
FOR XML PATH('')
```

This is close, but will have an undesired trailing ;. One way of fixing that could be:

```
SELECT top 1
LTRIM(STUFF((SELECT &#39;; &#39; + c1.CustomerName FROM Customer c1 FOR XML PATH (&#39;&#39;)), 1, 1,&#39;&#39;)) as SSV
from Customer c2;
```

In PostgreSQL one can also use this syntax to achieve the fully correct result:

```
SELECT array_to_string(array_agg(CustomerName), ' ; '::text)
FROM Customer
```



Given a table Employee having columns empName and empId, what will be the result of the SQL query below?

```
select empName from Employee order by 2 desc;
```

View the answer → Hide answer



“Order by 2” is only valid when there are at least two columns being used in select statement. However, in this query, even though the Employee table has 2 columns, the query is only selecting 1 column name, so “Order by 2” will cause the statement to throw an error while executing the above sql query.



What will be the output of the below query, given an Employee table having 10 records?

```
BEGIN TRAN  
TRUNCATE TABLE Employees  
ROLLBACK  
SELECT * FROM Employees
```

View the answer → Hide answer



This query will return 10 records as TRUNCATE was executed in the transaction. TRUNCATE does not itself keep a log but BEGIN TRANSACTION keeps track of the TRUNCATE command.



1. What is the difference between single-row functions and multiple-row functions?
2. What is the group by clause used for?

View the answer → Hide answer



1. Single-row functions work with single row at a time. Multiple-row functions work with data of multiple rows at a time.
2. The group by clause combines all those records that have identical values in a particular field or any group of fields.



What is the difference between char and varchar2?

View the answer → Hide answer



When stored in a database, `varchar2` uses only the allocated space. E.g. if you have a `varchar2(1999)` and put 50 bytes in the table, it will use 52 bytes.

But when stored in a database, `char` always uses the maximum length and is blank-padded. E.g. if you have `char(1999)` and put 50 bytes in the table, it will consume 2000 bytes.



Write an SQL query to display the text CAPONE as:

C
A
P
O
N
E

Or in other words, an SQL query to transpose text.

View the answer → Hide answer



```
Declare @a nvarchar(100)='capone';
Declare @length INT;
Declare @i INT=1;
SET @length=LEN(@a)
while @i<=@length
BEGIN
print(substring(@a,@i,1));
set @i=@i+1;
END
```

In Oracle SQL, this can be done as follows:

```
SELECT SUBSTR('CAPONE', LEVEL, 1)
FROM DUAL CONNECT BY LEVEL <= LENGTH('CAPONE');
```



Can we insert a row for identity column implicitly?

View the answer → Hide answer



Yes, like so:

```
SET IDENTITY_INSERT TABLE1 ON  
  
INSERT INTO TABLE1 (ID,NAME)  
SELECT ID,NAME FROM TEMPTB1  
  
SET IDENTITY_INSERT OFF
```



Given this table:

Testdb=# Select * FROM "Test"."EMP";

```
ID  
----  
1  
2  
3  
4  
5  
(5 rows)
```

What will be the output of below snippet?

```
Select SUM(1) FROM "Test"."EMP";  
Select SUM(2) FROM "Test"."EMP";  
Select SUM(3) FROM "Test"."EMP";
```

View the answer → Hide answer



```
5  
10  
15
```



Table is as follows:

ID	C1	C2	C3
1	Red	Yellow	Blue
2	NULL	Red	Green
3	Yellow	NULL	Violet

Print the rows which have 'Yellow' in one of the columns C1, C2, or C3, but without using OR.

View the answer → Hide answer



```
SELECT * FROM table
WHERE 'Yellow' IN (C1, C2, C3)
```



Write a query to insert/update Col2's values to look exactly opposite to Col1's values.

Col1	Col2
1	0
0	1
0	1
0	1
1	0

Col1	Col2
0	1
1	0
1	0

View the answer → Hide answer



```
update table set col2 = case when col1 = 1 then 0 else 1 end
```

Or if the type is numeric:

```
update table set col2 = 1 - col1
```



How do you get the last id without the max function?

View the answer → Hide answer



In MySQL:

```
select id from table order by id desc limit 1
```

In SQL Server:

```
select top 1 id from table order by id desc
```



What is the difference between IN and EXISTS?

[View the answer](#) → [Hide answer](#)



IN:

- Works on List result set
- Doesn't work on subqueries resulting in Virtual tables with multiple columns
- Compares every value in the result list
- Performance is comparatively SLOW for larger resultset of subquery

EXISTS:

- Works on Virtual tables
- Is used with co-related queries
- Exits comparison when match is found
- Performance is comparatively FAST for larger resultset of subquery



Suppose in a table, seven records are there.

The column is an identity column.

Now the client wants to insert a record after the identity value 7 with its identity value starting from 10.

Is it possible? If so, how? If not, why not?

[View the answer](#) → [Hide answer](#)



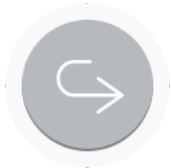
Yes, it is possible, using a DBCC command:

```
create table tableA
(id int identity,
 name nvarchar(50)
)
insert into tableA values ('ram')
insert into tableA values ('rahim')
insert into tableA values ('roja')
insert into tableA values ('rahman')
insert into tableA values ('rani')
insert into tableA values ('raja')
insert into tableA values ('raga')
select * From tableA
DBCC CHECKIDENT(tableA, RESEED, 9)
insert into tableA values ('roli')
insert into tableA values ('rosy')
insert into tableA values ('raka')
insert into tableA values ('rahul')
insert into tableA values ('rihan')
insert into tableA values ('bala')
insert into tableA values ('gala')
```



How can you use a CTE to return the fifth highest (or *Nth* highest) salary from a table?

View the answer → Hide answer



```
Declare @N int
set @N = 5;
WITH CTE AS
(
    SELECT Name, Salary, EmpID, RN = ROW_NUMBER()
        OVER (ORDER BY Salary DESC)
    FROM Employee
)
SELECT Name, Salary, EmpID
FROM CTE
WHERE RN = @N
```



Imagine a single column in a table that is populated with either a single digit (0-9) or a single character (a-z, A-Z). Write a SQL query to print 'Fizz' for a numeric value or 'Buzz' for alphabetical value for all values in that column.

Example:

['d', 'x', 'T', 8, 'a', 9, 6, 2, 'V']

...should output:

['Buzz', 'Buzz', 'Buzz', 'Fizz', 'Buzz', 'Fizz', 'Fizz', 'Fizz', 'Buzz']

View the answer → Hide answer



```
SELECT col, case when upper(col) = lower(col) then 'Fizz' else 'Buzz' end as FizzBuzz from table;
```



How do you get the Nth-highest salary from the Employee table without a subquery or CTE?

View the answer → Hide answer



```
SELECT salary from Employee order by salary DESC LIMIT 2,1
```

This will give the third-highest salary from the Employee table. Accordingly we can find out Nth salary using `LIMIT (N-1),1`.

But MS SQL Server doesn't support that syntax, so in that case:

```
SELECT salary from Employee order by salary DESC  
OFFSET 2 ROWS  
FETCH NEXT 1 ROW ONLY
```

OFFSET's parameter corresponds to the (N-1) above.



Given the following table named A:

x
2
-2
4
-4
-3
0
2

Write a single query to calculate the sum of all positive values of x and the sum of all negative values of x.

View the answer → Hide answer



```
select sum(case when x>0 then x else 0 end)sum_pos,sum(case when x<0 then x else 0 end)sum_neg from a;
```



Given the table mass_table:

weight
5.67
34.567
365.253
34

Write a query that produces the output:

weight	kg	gms
--------	----	-----

weight	kg	gms
5.67	5	67
34.567	34	567
365.253	365	253
34	34	0

View the answer → Hide answer



```
select weight, trunc(weight) as kg, nvl(substr(weight - trunc(weight), 2), 0) as gms
from mass_table;
```



Consider the Employee table below.

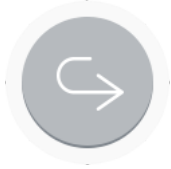
Emp_Id	Emp_name	Salary	Manager_Id
10	Anil	50000	18
11	Vikas	75000	16
12	Nisha	40000	18
13	Nidhi	60000	17
14	Priya	80000	18
15	Mohit	45000	18
16	Rajesh	90000	–
17	Raman	55000	16
18	Santosh	65000	17

Write a query to generate below output:

Manager_Id	Manager	Average_Salary_Under_Manager
------------	---------	------------------------------

16	manager_Id	Rajesh	65000	ge_Salary_Under_Manager
17		Raman	62500	
18		Santosh	53750	

View the answer → Hide answer



```
select b.emp_id as "Manager_Id",
       b.emp_name as "Manager",
       avg(a.salary) as "Average_Salary_Under_Manager"
from Employee a,
     Employee b
where a.manager_id = b.emp_id
group by b.emp_id, b.emp_name
order by b.emp_id;
```



How do you copy data from one table to another table ?

View the answer → Hide answer



```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```



Find the SQL statement below that is equal to the following: SELECT name FROM customer WHERE state = 'VA';

1. SELECT name IN customer WHERE state IN ('VA');
2. SELECT name IN customer WHERE state = 'VA';
3. SELECT name IN customer WHERE state = 'V';
4. SELECT name FROM customer WHERE state IN ('VA');

View the answer → Hide answer



4. SELECT name FROM customer WHERE state IN ('VA');



How to find a duplicate record?

1. duplicate records with one field
2. duplicate records with more than one field

View the answer → Hide answer



1. duplicate records with one field

```
SELECT name, COUNT(email)
FROM users
GROUP BY email
HAVING COUNT(email) > 1
```

2. duplicate records with more than one field

```
SELECT name, email, COUNT(*)
FROM users
GROUP BY name, email
HAVING COUNT(*) > 1
```

* There is more to interviewing than tricky technical questions, so these are intended merely as a guide. Not every “A” candidate worth hiring will be able to answer them all, nor does answering them all guarantee an “A” candidate. At the end of the day, [hiring remains an art, a science — and a lot of work.](#)

Submit an interview question

Submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.

Name

Email

All fields are required

☐ I agree with the Terms and Conditions of Toptal, LLC's [Privacy Policy](#)

Thanks for submitting your question.

Our editorial staff will review it shortly. Please note that submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.

Work with leading US companies on [SQL freelance jobs](#). Design your career as a SQL developer with Toptal.

[View full profile »](#)

[Mark Perg](#)

Israel

With a background from the Israeli elite intelligence unit and over 15 years of experience in software development and cybersecurity, Mark has everything in his skill set to deliver high-quality projects developed for your needs. As a full-stack engineer with experience in developing, architecting, and designing web applications with security and privacy by design approaches, Mark provides tailored high-quality solutions for your needs.

[SQL](#)[Java](#)[JavaScript](#)[Spring Boot](#)+9 more

[Hire Mark](#)

[View full profile »](#)

[Jonas Cristens](#)

Belgium

Jonas is an SQL/ETL developer and data scientist, and for the past few years, he's worked at Exellys—a top Belgium consultancy company—mainly with insurance and private banking companies. As an SQL developer, Jonas has automated financial reporting for a national bank and built a data quality framework along with multiple dashboards, and as a data scientist, he's analyzed customer behavior and financial transactions using machine learning.

[SQL](#)

[Hire Jonas](#)

[View full profile »](#)

[Levi Self](#)

United States

Levi has nearly a decade of experience in applied data science in a variety of industries with a concentration in the insurance industry. He's passionate about solving challenging problems that others find difficult or impossible. He's comfortable working independently and collaborating on teams. He is most at home in small startups with experience in enterprise as well.

[SQL](#)[Python](#)[RRStudio](#) [Shiny](#)+9 more

[Hire Levi](#)

Toptal connects the [top 3%](#) of freelance talent all over the world.

Join the Toptal community.

[Hire a Top developer](#)

or

[Apply as a developer](#)

Highest In-Demand Talent

- [iOS Developers](#)
- [Front-End Developers](#)
- [UX Designers](#)
- [UI Designers](#)
- [Financial Modeling Consultants](#)
- [Interim CFOs](#)
- [Digital Project Managers](#)

About

- [Top 3%](#)
- [Clients](#)
- [Freelance Developers](#)
- [Freelance Designers](#)
- [Freelance Finance Experts](#)
- [Freelance Project Managers](#)

- [Freelance Product Managers](#)
- [Specialized Services](#)
- [About Us](#)

Contact

- [Contact Us](#)
- [Press Center](#)
- [Careers](#)
- [FAQ](#)

Social



Hire the top 3% of freelance talent™

- © Copyright 2010 - 2020 Toptal, LLC
- [Privacy Policy](#)
- [Website Terms](#)

By continuing to use this site you agree to our [Cookie Policy](#).

Got it

Work with leading tech companies on freelance SQL jobs. [Apply to Toptal's Freelance SQL jobs](#)