accepted

### [Key Difference between DBMS and RDBMS](https://www.udemy.com/blog/differences-between-dbms-and-rdbms/):

**The key difference is that RDBMS (relational database management system) applications store data in a tabular form, while DBMS applications store data as files**.

Does that mean there are no tables in a DBMS?

*here can be, but there will be no “relation” between the tables*, like in a RDBMS. In DBMS, data is generally stored in either a hierarchical form or a navigational form. This means that a single data unit will have one parent node and zero, one or more children nodes. It may even be stored in a graph form, which can be seen in the network model.

In a RDBMS, the tables will have an identifier called primary key. Data values will be stored in the form of tables. The relationships between these data values will be stored in the form of a table as well. Every value stored in the relational database is accessible. This value can be updated by the system. The data in this system is also physically and logically independent.

## ****You can say that a RDBMS is an extension of a DBMS****, even if there are many differences between the two. Most software products in the market today are both DBMS and RDBMS compliant. Essentially, they can maintain databases in a (relational) tabular form as well as a file form, or both. This means that today a RDBMS application is a DBMS application, and vice versa. However, there are still major differences between a relational database system for storing data and a plain database system.

These are called **ACID** properties.

## What is Normalization?

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

**Atomicity:**

**If in a transaction multiple instructions are given then either nothing will executes or all will be executed**

By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves following two operations.  
—**Abort**: If a transaction aborts, changes made to database are not visible.  
—**Commit**: If a transaction commits, changes made are visible.

* **Consistency** – if your database is consistent before transation then it should be consistent after the transaction as well

**Befoe transact A and B then transafer from A to B 5**

**After tarnsaction account A** -5 to B +5

**Then after transaction A+B and before transaction should be same A+B**

The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

* **Durability** – if any change has been committed then that should not be change

The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

* **Isolation** – There should not be any impact on any transaction if multiple transaction is going on simultaneously

In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

# SQL Views

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

The Syntax to create a sql view is

CREATE VIEW view\_name

AS

SELECT column\_list

FROM table\_name [WHERE condition];

# [Is a view in the database updatable?](https://stackoverflow.com/questions/3777918/is-a-view-in-the-database-updatable)

The basic criteria is it has to be an updateable view in the opinion of the database engine, that is to say can the engine uniquely identify the row(s) to be updated and secondly are the fields updateable. If your view has a calculated field or represents the product of a parent/child join then the default answer is probably no.

In the past it wasn't possible to update any views. The main purpose of a view is to *look* at data, hence the name. It could also have been called a *stored query*.

Today, many database engines support to update views. It's bound to restrictions, some updates are virtually impossible (eg. calculated columns, group by etc).

## What is Normalization?

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

**Triggers**

Triggers are database object. Basically these are special type of stored procedure that are automatically fired/executed when a DDL or DML command statement related with the trigger is executed.

## Types of Triggers

In Sql Server we can create four types of triggers

## 1. Data Definition Language (DDL) triggers, (After Trigger(after SQL Server finish the execution of the action successfully,  If the record/row insertion fails, SQL Server will not fire the After Trigger.), Instead of Trigger(fires before SQL Server starts the execution of the action If the record/row insertion fails, SQL Server will fire the Instead of Trigger.))

**2.**Data Manipulation Language (DML) triggers,

**3.** CLR triggers

**4.**Logon triggers.

## What is GraphQL? [#](https://graphql.github.io/blog/graphql-a-query-language/#what-is-graphql)

A GraphQL query is a string that is sent to a server to be interpreted and fulfilled, which then returns JSON back to the client.

{

user(id: 4802170) {

id

name

isViewerFriend

profilePicture(size: 50) {

uri

width

height

}

friendConnection(first: 5) {

totalCount

friends {

id

name

}

}

}

}

**{**

**"data": {**

**"user": {**

**"id": "4802170",**

**"name": "Lee Byron",**

**"isViewerFriend": true,**

**"profilePicture": {**

**"uri": "cdn://pic/4802170/50",**

**"width": 50,**

**"height": 50**

**},**

**"friendConnection": {**

**"totalCount": 14,**

**"friends": [**

**{**

**"id": "305249",**

**"name": "Stephen Schwink"**

**},**

**{**

**"id": "3108935",**

**"name": "Nathaniel Roman"**

**},**

**{**

**"id": "9020247",**

**"name": "William Sanville"**

**},**

**{**

**"id": "13957785",**

**"name": "Alex Langenfeld"**

**},**

**{**

**"id": "37000641",**

**"name": "Nick Schrock"**

**}**

**]**

**}**

**}**

**}**

**}**

**Radis Caching databse: Redis is also NoSQL Database**

Each of the shards will have a **maximum size** of 2 GB. In this case, the **maximum**dataset **size** that you will be able to store in the **database** is 2 GB.

**Set**

**Get**

**Mongo DB is NoSQL**

## When to use Redis?

* Caching

Caching using MongoDB simply doesn't make a lot of sense. It would be too slow.

If you don't care that much about scaling.

* You should know that redis database size is limited by the amount of RAM in the machine. Any larger than that and you have to think clustering which is manual and intensi
* Redis does not make any assumptions based on your data.
* Redis provides a bunch of useful data structures (e.g. Sets, Hashes, Lists), but you have to explicitly define how you want to store you data. To put it in a nutshell, Redis and MongoDB can be used in order to achieve similar things. Redis is simply faster, but not suited for prototyping. That's one use case where you would typically prefer MongoDB. Besides that, Redis is **really**flexible. The underlying data structures it provides are the building blocks of high-performance DB systems.
* If you need **really** high performance.

# When to use MongoDB

* Prototyping, Startups, Hackathons
* When you need to change your schema quickly.

Use Redis if performance is important and you are willing to spend time optimizing and organizing your data. - Use MongoDB if you need to build a prototype without worrying too much about your DB.

Redis and MongoDB are both non-relational databases but they're of different categories.

Redis is a Key/Value database, and it's using In-memory storage which makes it super fast. It's a good candidate for caching stuff and temporary data storage(in memory) and as the most of cloud platforms (such as Azure,AWS) support it, it's memory usage is scalable.But if you're gonna use it on your machines with limited resources, consider it's memory usage.

MongoDB on the other hand, is a document database. It's a good option for keeping large texts, images, videos, etc and almost anything you do with databases except transactions.For example if you wanna develop a blog or social network, MongoDB is a proper choice. It's scalable with scale-out strategy. It uses disk as storage media, so data would be persisted.

## When1 to use Redis?

1.Caching

2.Caching using MongoDB simply doesn't make a lot of sense. It would be too slow.

If you don't care that much about scaling.



# When to use MongoDB

Prototyping, Start-ups, Hackathons

When you need to change your schema quickly.

Redis is an **in memory** data store, that can persist it's state to disk (to enable recovery after restart). However, being an in-memory data store means the size of the data store (on a single node) cannot exceed the total memory space on the system (physical RAM + swap space). In reality, it will be much less that this, as Redis is sharing that space with many other processes on the system, and if it exhausts the system memory space it will likely be killed off by the operating system.

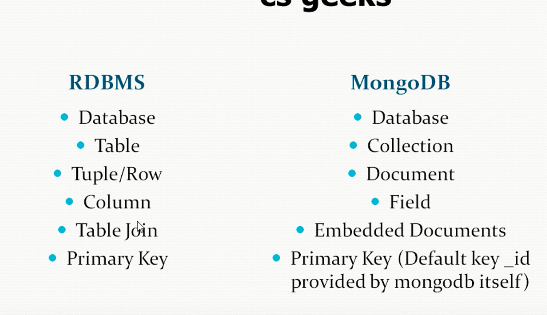
Mongo is a **disk based** data store, that is most efficient when it's working set fits within physical RAM (like all software). Being a disk based data means there are no intrinsic limits on the size of a Mongo database, however configuration options, available disk space, and other concerns may mean that databases sizes over a certain limit may become impractical or inefficient.

Both Redis and Mongo can be clustered for high availability, backup and to increase the overall size of the datastore.

# MongoDB: Complex Data

# Redis: Key-Value

https://stackoverflow.com/questions/5400163/when-to-redis-when-to-mongodb



Query

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Two tables highest slary for each department

101 EE

102 ME.

103 CS

104 IT

105 CI

id ename deptid salary

1 A 101 10007

2 B 101 10006

3 C 101 10005

4 D 101 10004

5 E 102 2001

6 F 102 1001

7 G 102 1007

8 H 103 3300

9 I 103 4407

10 J 103 1001

11 K 104 1004

12 L 104 1002

13 M 104 1004

14 N 105 1099

SELECT id, Name,max(Salary),d.deptid AS DEPT

From employee1 e

inner JOIN

department d

ON e.deptid=d.deptid

GROUP BY e.deptid having max(Salary) >=5000;

**Order by column but null value should come first**

SELECT \* FROM employee order BY deptid is null desc,deptid desc;

**Selct only Odd Records**

SELECT \* FROM employee where mod(id,2) = 0;;

SELECT SUBSTRING("SQL Tutorial", 5, 3) AS ExtractString;

Return a substring of a string before a specified number of delimiter occurs:

**SELECT SUBSTRING\_INDEX("www.w3schools.com", ".", 2);**

**mysql doesn't support full join**

**Duplicate Rows**

Select Student\_id, Student\_name, Department , COUNT(\*)

from tblstudent group by Student\_ID,Student\_name,Department

having COUNT(\*)>1

Select Student\_id, Student\_name, Department,

ROW\_NUMBER() OVER(PARTITION BY Student\_id, Student\_name ORDER BY Student\_id) AS DuplicateRowCount

FROM tblstudent

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Salary**

SELECT TOP 1 salary ,Name FROM (SELECT DISTINCT TOP 4 salary, Name FROM employee ORDER BY Salary DESC) a ORDER BY salary

**Without using top n max**

SELECT \* FROM empy Emp1

WHERE (4) = (SELECT COUNT(DISTINCT(Emp2.Salary))FROM empy Emp2

WHERE Emp2.Salary > Emp1.Salary)

By department

SELECT salary,Dept

FROM employees Emp1

WHERE (2) IN (SELECT COUNT(DISTINCT(Emp2.salary))FROM employees Emp2

WHERE Emp2.salary >= Emp1.salary GROUP BY Dept) ;

SELECT Salary,name

FROM

(

SELECT Salary,name,DENSE\_RANK() OVER(ORDER BY Salary DESC) as Rno from Emp\_Salary

) as tbl

WHERE Rno=3

By department

SELECT salary,Dept

FROM employees Emp1

WHERE (2) IN (SELECT COUNT(DISTINCT(Emp2.salary))FROM employees Emp2

WHERE Emp2.salary >= Emp1.salary GROUP BY Dept) ;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Delete duplicate records/rows by creating identity column.**

**Delete duplicate rows if primary key is present**

Id is primary key

DELETE t1 FROM tabletest t1, tabletest t2

WHERE t1.id < t2.id

AND t1.name = t2.name

AND t1.email = t2.email;

Create table Test1 (a int not null, b int not null, c int not null, id int not null identity)

GO

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (4,5,6)

ALTER TABLE dbo.Employees ADD ID INT IDENTITY(1,1)

Now write this query to delete duplicate rows.

DELETE FROM MyTable

LEFT OUTER JOIN (

SELECT MIN(RowId) as RowId, Col1, Col2, Col3

FROM MyTable

GROUP BY Col1, Col2, Col3

) as KeepRows ON

MyTable.RowId = KeepRows.RowId

WHERE

KeepRows.RowId IS NULL

Delete duplicate records using Row\_Number()

**If you do not want to make any changes in table design or don't want to create identity column on table then you can remove duplicate records using Row\_Number in sql server 2005 onwards.**

for this write below mentioned code and execute.

Delete duplicate rows using Common Table Expression(CTE)

With CTE\_Duplicates as

(select empid,name , row\_number() over(partition by empid,name order by empid,name ) rownumber

from EmpDup )

delete from CTE\_Duplicates where rownumber!=1

WITH Dup\_Students\_CTE (Student\_ID , Student\_name, Department, DuplicateRowCount)

AS

(

SELECT Student\_ID, Student\_name, Department,

ROW\_NUMBER() OVER(PARTITION BY Student\_ID, Student\_name ORDER BY Student\_ID) AS DuplicateRowCount

FROM tblstudent

)

DELETE FROM Dup\_Students\_CTE

WHERE DuplicateRowCount > 1

GO

**transfer data from one table to another**

Create table with data from another table

Select \* INTO test675 From Test

**Data Migration**

## Natural Join

Natural join is a type of equi join which occurs implicitly by comparing all the same names columns in both tables. The join result have only one column for each pair of equally named columns.

1. In Natural join, you can't see what columns from both the tables will be used in the join. In Natural join, you might not get the desired result what you are expecting.
2. Natural join clause is not supported by SQL Server, it is supported by Oracle and MySQL.

**Group By and having**

1. WHERE clause is used to impose condition on SELECT statement as well as single row function and is used before GROUP BY clause where as HAVING clause is used to impose condition on GROUP Function and is used after GROUP BY clause in the query[

2. Having can used only with the select statement .it is typically used with group by clause or aggregate function. When it is used without group by it work like where clause

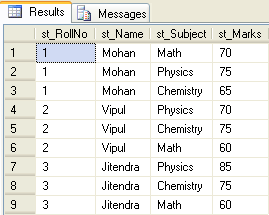
|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in  the second  position |
| WHERE CustomerName LIKE 'a\_%\_%' | Finds any values that starts with "a" and are  at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a"  and ends with "o" |

**Introduction**

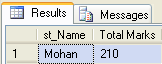
We always get confused between WHERE and **Having** **clause** and make mistakes. Here in this article, I will try to highlight all the major **difference**s between WHERE and **HAVING**, and things you should be aware of, when using either WHERE or **HAVING**.

So we can see that the difference between the having and where clause in sql is that the where clause can not be used with aggregates, but the having clause can. One way to think of it is that the having clause is an additional filter to the where clause.

------------------------------------------------------------------------------

A HAVING clause is like a WHERE clause, **but applies only to groups as a whole, The WHERE clause is applied first to the individual rows in the tables. Only the rows that meet the conditions in the WHERE clause are grouped.** The HAVING clause is then applied to the rows in the result set. Only the **whereas the WHERE clause applies to individual rows. A query can contain both a WHERE clause and a HAVING clause.** groups that meet the HAVING conditions appear in the query output. You can apply a HAVING clause only to columns that also appear in the GROUP BY clause or in an aggregate function.   
Most of ------------------------------------------------------------------------------  
the time you will get the same result with Where or **Having**. The below given two SQL command produces the same result set That is, both count the number of records found for the states of California and Los Angles.   
  
SELECT state, COUNT (\*)  
FROM Test  
WHERE state IN ('CA', 'LA')  
GROUP BY state  
ORDER BY state  
  
SELECT state, COUNT(\*)  
FROM Test  
GROUP BY state  
**HAVING** state IN ('CA', 'LA')  
ORDER BY state 

1. **SELECT st\_Name, SUM(st\_Marks) AS 'Total Marks'**
2. **FROM StudentMarks**
3. **where st\_Name='Mohan'**
4. **GROUP BY st\_Name;**



**Background**

(Optional) So, where is the **difference**, which is better? I'll let you answer those questions in a minute.  
  
The main reason for using WHERE **clause** is to select rows that are to be included in the query. For example, assume table Test.Suppose I want the names, account numbers, and balance due of all customers from California and Los Angles. Since STATE is one of the fields in the record format, I can use WHERE to select those customers.

**Using the code**

SELECT cusnum, lstnam, init  
FROM Test  
WHERE state IN ('CA', 'LA')  
  
CUSNUM LSTNAM INIT BALDUE  
====== ============ ==== ========  
938472 John G K 37.00  
938485 Mark J A 3987.50   
593029 Lily E D 25.00  
  
  
Suppose I want the total amount due from customers by state. In that case, I would need to use the GROUP BY **clause** to build an aggregate query.   
  
SELECT state,SUM(baldue)  
FROM Test  
GROUP by state  
ORDER BY state  
  
State Sum(Baldue)  
===== ===========  
CA 250.00  
CO 58.75   
GA 3987.50   
MN 510.00  
NY 589.50  
TX 62.00   
VT 439.00   
WY .00

**Points of Interest**

Suppose I want the same information, but I don't care about states where nobody owes me any money. Since the total owed by state is an aggregate figure, i.e., the figure is generated from a group of records, you must use **HAVING** to select the proper data.   
  
SELECT state, SUM(balancdue)  
FROM Test  
GROUP by state  
**HAVING** SUM(baldue) > 0  
ORDER BY state  
  
State Sum(Baldue)  
===== ===========  
CA 250.00  
CO 58.75   
GA 3987.50   
MN 510.00  
NY 589.50  
TX 62.00   
VT 439.00   
 **Here's the rule. If a condition refers to an aggregate function, put that condition in the HAVING clause. Otherwise, use the WHERE clause.**  
  
**Here's another rule: You can't use HAVING unless you use GROUP BY.**  
  
Now, go back to the first example, where WHERE and **HAVING** produce the same result set. What's the**difference**? The first query uses the WHERE **clause** to restrict the number of rows that the computer has to sum up. But the second query sums up all the rows in the table, then uses **HAVING** to discard the sums it calculated for all states except Texas and Georgia. The first query is obviously the better one, because there is no need to make the computer calculate sums and then throw them away.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Having clause without Group by function**

create table empy(empid int , marks int, dept varchar(40))

insert into  empy(empid  , marks , dept ) values (1, 2 ,'A')

insert into  empy(empid  , marks , dept ) values (2, 3 ,'B')

insert into  empy(empid  , marks , dept ) values (3, 4 ,'C')

insert into  empy(empid  , marks , dept ) values (4, 5 ,'B')

insert into  empy(empid  , marks , dept ) values (10, 2 ,'C')

insert into  empy(empid  , marks , dept ) values (1, 2 ,A)

insert into  empy(empid  , marks , dept ) values (1, 2 ,A)

**What is the difference between a "where" clause and a "having" clause?** - **"Where" is a kind of restiriction statement. You use where** clause to restrict all the data from DB.Where clause is using before result retrieving. But Having clause is using after retrieving the data.Having clause is a kind of filtering command.

**What structure can you implement for the database to speed up table reads?**- Follow the rules of DB tuning we have to: 1] properly use indexes ( different types of indexes) 2] properly locate different DB objects across different tablespaces, files and so on.3] create a special space (tablespace) to locate some of the data with special datatype ( for example CLOB, LOB and …

**What is a "constraint"?** - A **constraint allows you to apply simple referential integrity checks to a table. There** are four primary types of constraints that are currently supported by SQL Server: PRIMARY/UNIQUE - enforces uniqueness of a particular table column. DEFAULT - specifies a default value for a column in case an insert operation does not provide one. FOREIGN KEY - validates that every value in a column exists in a column of another table. CHECK - checks that every value stored in a column is in some specified list. Each type of constraint performs a specific type of action. Default is not a constraint. NOT NULL is one more constraint which does not allow values in the specific column to be null. And also it the only constraint which is not a table level constraint

**Why can a "group by" or "order by" clause be expensive to process**? - Processing of "group by" or "order by" clause often requires creation of Temporary tables to process the results of the query. Which depending of the result set can be very expensive.

**What is "index covering" of a query**? - Index covering means that "Data can be found only using indexes, without touching the tables"

**What is a SQL view**? - An output of a query can be stored as a view. View acts like small table which meets our criterion. View is a precomplied SQL query which is used to select data from one or more tables. A view is like a table but it doesn’t physically take any space. View is a good way to present data in a particular format if you use that query quite often. View can also be used to restrict users from accessing the tables directly.

IN & BETWEEN

An easier method of using compound conditions uses IN or BETWEEN. For example, if you wanted to list all

Managers and staff:

SELECT EMPLOYEEIDNO

FROM EMPLOYEESTATISTICSTABLE

WHERE POSITION IN ('Manager', 'Staff');

or to list those making greater than or equal to $30,000, but less than or equal to $50,000, use:

SELECT EMPLOYEEIDNO

FROM EMPLOYEESTATISTICSTABLE

WHERE SALARY BETWEEN 30000 AND 50000;

One special use of GROUP BY is to associate an aggregate function (especially COUNT; counting the number

of rows in each group) with groups of rows. First, assume that the Antiques table has the Price column, and

each row has a value for that column. We want to see the price of the most expensive item bought by each

Owner. We have to tell SQL to group each owner's purchases, and tell us the maximum purchase price:

SELECT BUYERID, MAX(PRICE)

FROM ANTIQUES

GROUP BY BUYERID;

Now, say we only want to see the maximum purchase price if the purchase is over $1000, so we use the

HAVING clause:

SELECT BUYERID, MAX(PRICE)

FROM ANTIQUES

GROUP BY BUYERID

HAVING PRICE > 1000;

**Queries**

# SQL Subquery

Sub query or Inner query or Nested query is a query in a query. A subquery is usually added in the WHERE Clause of the sql statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value.

Subqueries are an alternate way of returning data from multiple tables.

Another common usage of subqueries involves the use of operators to allow a Where condition to include the

Select output of a subquery. First, list the buyers who purchased an expensive item (the Price of the item is $100

greater than the average price of all items purchased):

SELECT BUYERID

FROM ANTIQUES

WHERE PRICE >

(SELECT AVG(PRICE) + 100

FROM ANTIQUES);

The subquery calculates the average Price, plus $100, and using that figure, an OwnerID is printed for every

item costing over that figure. One could use DISTINCT BUYERID, to eliminate duplicates.

List the Last Names of those in the AntiqueOwners table, ONLY if they have bought an item:

SELECT OWNERLASTNAME

FROM ANTIQUEOWNERS

WHERE OWNERID IN

(SELECT DISTINCT BUYERID

FROM ANTIQUES);

The subquery returns a list of buyers, and the Last Name is printed for an Antique Owner if and only if the

Owner's ID appears in the subquery list (sometimes called a candidate list). Note: on some DBMS's, equals

can be used instead of IN, but for clarity's sake, since a set is returned from the subquery, IN is the better

choice.

For an Update example, we know that the gentleman who bought the bookcase has the wrong First Name in the

database...it should be John:

UPDATE ANTIQUEOWNERS

SET OWNERFIRSTNAME = 'John'

WHERE OWNERID =

(SELECT BUYERID

FROM ANTIQUES

WHERE ITEM = 'Bookcase');

First, the subquery finds the BuyerID for the person(s) who bought the Bookcase, then the outer query updates

his First Name.

 sp\_rename 'tblEmp' , 'tlEmployee'

GRANT = User or role has been explicitly given access.  
DENY = User or role has been explicitly denied access. This trumps any GRANT.  
REVOKE = This is the "undo." It will remove either a GRANT or DENY.  
  
 GRANT SELECT ON TABLE MyTable TO PUBLIC  
  
  
        test=> CREATE TABLE permtest (col INTEGER);   
        CREATE   
        test=> -- now only the owner can use permtest   
        test->   
        test=> GRANT SELECT ON permtest TO meyers;   
        CHANGE   
        test=> -- now user 'meyers' can do SELECTs on permtest   
        test=>   
        test=> GRANT ALL ON permtest TO PUBLIC;     
        CHANGE   
  
<http://sqldbpool.com/2008/05/08/sql-server-2005-interview-questions/>  
<http://aspalliance.com/1455_Frequently_Asked_Questions_in_SQL_Server_2005__Programmer_Perspective.2>  
  
**What does the ON DELETE CASCADE option do?**  
ON DELETE CASCADE

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the rows deleted from those tables.

**ON UPDATE CASCADE**  
Specifies that if an attempt is made to update a key value in a row, where the key value is referenced by foreign keys in existing rows in other tables, all of the foreign key values are also updated to the new value specified for the key. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the key values updated in those tables.  
  
**Varchar and nvarchar**  
  
the difference is that nvarchar is used to store unicode data, which is used to store multilingual data in your database tables. Other languages have an extended set of character codes that need to be saved and this datatype allows for this extension. If your database will not be storing multilingual data you should use the varchar datatype instead. The reason for this is that nvarchar takes twice as much space as varchar, this is because of the need to store the extended character codes for other languages  
**Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, and walkthroughs and inspection meetings.**

**Validation ensures that functionality, as defined in requirements, is the intended behaviour of the product; validation typically involves actual testing and takes place after verifications are completed.**  
  
  
<http://sqldbpool.files.wordpress.com/2010/12/sql-server-interview-questions.pdf>  
<http://sqldbpool.com/2008/05/08/sql-server-2005-interview-questions/>

Imp

<http://blog.sqlauthority.com/2012/08/17/sql-server-curious-case-of-disappearing-rows-on-update-cascade-and-on-delete-cascade-t-sql-example-part-2-of-2/>

**NVL**

select supplier\_id,  
NVL(supplier\_desc, supplier\_name)  
from suppliers;

This SQL statement would return the *supplier\_name* field if the *supplier\_desc* contained a null value. Otherwise, it would return the *supplier\_desc*.

**transaction**  
  
A transaction is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database).  
A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations:

* Decrement the savings account
* Increment the checking account
* Record the transaction in the transaction journal

[http://www.elated.com/articles/javascript-and-cookie](http://www.elated.com/articles/javascript-and-cookies/)

<http://blog.sqlauthority.com/2007/04/15/sql-server-interview-questions/>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Delete duplicate rows if primary key is present

Create table Test1 (a int not null, b int not null, c int not null, id int not null identity)

GO

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Test1 (A,B,C) VALUES (1,1,1)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (1,2,3)

INSERT INTO Tes1t (A,B,C) VALUES (4,5,6)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Why use No count on

Whenever we write any procedure and execute it a message appears in message window that shows no of rows affected with the statement written in the procedure and we become very happy to see that our procedure is working.

But do you know that this message creates an extra overhead on the network? Yes it does.  
By removing this extra overhead from the network, we can actually improve the performance of our database and our application.

How should we do it?  
**Solution**

When you create any procedure then first line of your procedure should be

SET NOCOUNT ON;

This one line of code turns off the message that SQL server sends back to front end after every T-SQL statement is executed. This is applied for all SELECT, INSERT, UPDATE and DELETE statements. As when stored procedures are executed there is no need to pass this information back to front end.

**Why use No lock?**

Nolock is not a performance tool. It controls the accuracy of the data read by that query. If you don't mind queries occasionally returning incorrect data, then by all means, use it.

I used to see my senior developers use WITH (NOLOCK) when querying in SQL Server and wonder why they use. Now i explored it and found that its useful to improve the performance in executing the query. However there is a disadvantage in using it. The disadvantage is that one may not be sure that they are getting the data which is currently being updated in the Table ie **Without lock protection, you cannot be guaranteed that the data isn’t changing during the time that the query is running**. I reffered [this](http://blogs.neudesic.com/blogs/phil_scott/archive/2005/12/05/11.aspx) link and found it prettu useful

**NOLOCK** is also known as "dirty reads". This option directs SQL Server not to issue shared locks and not to honor exclusive locks. So, if this option is specified, it is possible to read an uncommitted transaction. This results in higher concurrency and in lower consistency.

When you use the NOLOCK query hint you are telling the storage engine that you want to access the data no matter if the data is locked by another process or not.  This is why is can make it appear that the query is just running faster as you are no longer waiting for other processes to complete their writes, you are simply reading what ever is in the buffer pool or on disk at the time that you get to it.  This leads to a problem called dirty reads, meaning that you may not be getting the same values that you would get it you were to run the query again.  This isn’t necessarily a bad thing, just something to be aware of.  Usage of the NOLOCK hint may be just fine in your application, or it may be incredibly bad.

**What is a deadlock and what is a live lock? How will you go about resolving deadlocks?**

Deadlock is a situation when two processes, each having a lock on one piece of data, attempt to acquire a lock on the other's piece. Each process  would wait indefinitely for the other to release the lock, unless one of the user processes is terminated. SQL Server detects deadlocks and terminates one user's process.

A livelock is one, where a  request for an exclusive lock is repeatedly denied because a series of overlapping shared locks keeps interfering. SQL Server detects the situation after four denials and refuses further shared locks. A livelock also occurs when read transactions monopolize a table or page, forcing a write transaction to wait indefinitely.

Check out SET DEADLOCK\_PRIORITY and "Minimizing Deadlocks"  in SQL Server books online. Also check out the article Q169960 from Microsoft knowledge base.

create table Persons (P\_id int, LastName varchar(30), FirstName varchar(30), Address varchar(50), City Varchar(40)   )

into Persons(P\_id, LastName,FirstName, Address, Company ) values(1,'Sahu' , 'Ram', 'Fatehpur','MDSYnergy' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(2,'Goyal' , 'Vipin', 'Raibareily','Esoft' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(3,'Agarwal' , 'Ishant', 'barrelly','Esoft' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(4,'Gupta' , 'Gaurav', 'Kota','IBM' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(5,'Manan' , 'Brijesh', 'Lucknow','BEE' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(6,'Khan' , 'Afzal', 'Lucknow','BIRLA' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(7,'Thakur' , 'Deepak', 'Shakti','Schinder' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(8,'Agarwal' , 'Nitin', 'Gwalior','Amazon' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(9,'yadav' , 'Ranjit', 'Varanasi','Birla' )

insert into Persons(P\_id, LastName,FirstName, Address, Company ) values(10,'Gupta' , 'Aashish', 'Varanasi','MDSynergy' )

Alter table Persons Alter column P\_id int  Not null

SP\_RENAME 'Persons.Compny', 'Company', 'Column'

Alter table persons Add Primary Key(P\_id)

Alter table Persons Alter column P\_id varchar(30)

Alter table orders Add Foreign Key(P\_id) REFERENCES Persons(P\_id)

create table Orders (O\_id int,OrderNo varchar(30), P\_id varchar(30))

insert into Orders(o\_id, OrderNo,P\_id) values(2, '24325', '4' )

insert into Orders(o\_id, OrderNo,P\_id) values(4,'43534', '5' )

insert into Orders(o\_id, OrderNo,P\_id) values(5,'43434', '5' )

Add foreign key

Alter table Orders Add foreign key(P\_id) REFERENCES Persons(P\_id)

ALTER TABLE Persons

DROP CONSTRAINT PK\_\_Persons\_\_595B400

Sp\_heplconstraint s

SP\_columns Persons

**Duplicate Rows**

--------------------------------------------------------

|  |
| --- |
| CREATE TABLE Dup\_Students (ID INT, FirstName varchar(25), Department Char(2))  CREATE TABLE Dup\_Students (ID INT, FirstName varchar(25), Department Char(2))   INSERT INTO Dup\_Students VALUES(1, 'Jack', 'IT') INSERT INTO Dup\_Students VALUES(2, 'Alice', 'ME') INSERT INTO Dup\_Students VALUES(3, 'James', 'EE') INSERT INTO Dup\_Students VALUES(4, 'Nickle', 'CE') INSERT INTO Dup\_Students VALUES(5, 'George', 'IT') -- INSERT INTO Dup\_Students VALUES(1, 'Jack', 'IT') INSERT INTO Dup\_Students VALUES(2, 'Alice', 'ME') INSERT INTO Dup\_Students VALUES(3, 'James', 'EE') INSERT INTO Dup\_Students VALUES(4, 'Nickle', 'CE')  -- INSERT INTO Dup\_Students VALUES(1, 'Jack', 'IT') INSERT INTO Dup\_Students VALUES(2, 'Alice', 'ME')  Now you can view the number of duplicate records in each row.  SELECT ID, FirstName, Department, Count(\*) as DuplicateCount  From Dup\_Students group by ID,FirstName, Department  Before deleting you can view the records with row number for the duplicates. This result set is going to act as a temp table for the delete process.  SELECT ID, FirstName, Department,  ROW\_NUMBER() OVER(PARTITION BY ID, FirstName ORDER BY ID) AS DuplicateRowCount FROM Dup\_Students  We are going to delete all the duplicates using CTE (Common table expression) and ROW\_NUMBER(), which is a new in SQL server 2005.  WITH Dup\_Students\_CTE (ID, FirstName, Department, DuplicateRowCount) AS ( SELECT ID, FirstName, Department,  ROW\_NUMBER() OVER(PARTITION BY ID, FirstName ORDER BY ID) AS DuplicateRowCount FROM Dup\_Students ) DELETE FROM Dup\_Students\_CTE WHERE DuplicateRowCount > 1 GO |

-----------------------------------------------------------

|  |  |  |
| --- | --- | --- |
| **Re: Difference between primary key and unique key ?** | | |
|  |  |  |

First difference is that primary key doesn't accept null values whereas unique accepts one or multiple. second difference is that Clustered index is created on Primary key constraint and non clustered unique indexes is created on Unique key constraint.

**Composite key**, or **composite primary key**, refers to cases where more than one column is used to specify the **primary key** of a table

**Candidate Key(Primary Key)** is a Key which Maintains the Row

**Unique Key** .Can be defined based on the Entity

****

accepts null Values .So that the records can still be

entered submitting null values to this attribute.

**primary key:**- The attribute or combination of attributes

that uniquely identifies a row or record.

**Foreign Key:**- A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

**CREATE TABLE ORDERS   
(Order\_ID integer primary key,   
Order\_Date datetime,   
Customer\_SID integer references CUSTOMER(SID),   
Amount double);**

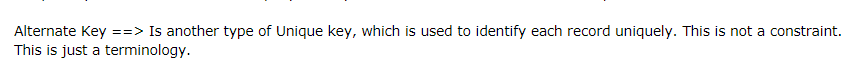
**Composite key**:- A primary key that consistsof two or more

attributes is known as composite key

555555RCCXVCXVCC

**candidate key:**- is a column in a table which has the

ability to become a primary key.



**Foreign Key**

|  |
| --- |
| A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.  **CREATE TABLE ORDERS  (Order\_ID integer primary key,  Order\_Date datetime,  Customer\_SID integer references CUSTOMER(SID),  Amount double);**  **Joins**  **Inner**  SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons INNER JOIN Orders ON Persons.P\_Id=Orders.P\_Id ORDER BY Persons.LastName SQL LEFT JOIN Keyword The LEFT JOIN keyword returns all rows from the left table (table\_name1), even if there are no matches in the right table (table\_name2). SQL LEFT JOIN Syntax SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2 ON table\_name1.column\_name=table\_n SQL RIGHT JOIN Keyword The RIGHT JOIN keyword returns all the rows from the right table (table\_name2), even if there are no matches in the left table (table\_name1). SQL RIGHT JOIN Syntax SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name SQL FULL JOIN Keyword The FULL JOIN keyword return rows when there is a match in one of the tables. SQL FULL JOIN Syntax SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name  **Self Join with new inserting method**  CREATE TABLE emp ( empid int, mgrid int, empname char(10) )  INSERT emp SELECT 1,2,'Vyas' INSERT emp SELECT 2,3,'Mohan' INSERT emp SELECT 3,NULL,'Shobha' INSERT emp SELECT 4,2,'Shridhar' INSERT emp SELECT 5,2,'Sourabh'  SELECT t1.empname [Employee], t2.empname [Manager] FROM emp t1, emp t2 WHERE t1.mgrid = t2.empid  Here's an advanced query using a LEFT OUTER JOIN that even returns the employees without managers (super bosses)  SELECT t1.empname [Employee], COALESCE(t2.empname, 'No manager') [Manager] FROM emp t1 LEFT OUTER JOIN emp t2 ON t1.mgrid = t2.empid  13Share  Here's an advanced query using a LEFT OUTER JOIN that even returns the employees without managers (super bosses)  SELECT t1.empname [Employee], COALESCE(t2.empname, 'No manager') [Manager] FROM emp t1 LEFT OUTER JOIN emp t2 ON t1.mgrid = t2.empid  13Share  **Types of constraint  :**    SQL Server constraints allow you to enforce rules in your database. These rules may affect business logic, database integrity and/or table structures. Each one plays an important role in your database architecture. The six types of constraints supported by Microsoft SQL Server include:     * [**NOT NULL Constraint**](http://www.1keydata.com/sql/sql-not-null.html): Ensures that a column cannot have NULL value. * [**DEFAULT Constraint**](http://www.1keydata.com/sql/sql-default.html): Provides a default value for a column when none is specified. * [**UNIQUE Constraint**](http://www.1keydata.com/sql/sql-unique.html): Ensures that all values in a column are different. * [**CHECK Constraint**](http://www.1keydata.com/sql/sql-check.html): Makes sure that all values in a column satisfy certain criteria. * [**Primary Key Constraint**](http://www.1keydata.com/sql/sql-primary-key.html): Used to uniquely identify a row in the table. * [**Foreign Key Constraint**](http://www.1keydata.com/sql/sql-foreign-key.html):   **DQL: SELECT** **DML: DELETE, INSERT, UPDATE** **DDL: CREATE, DROP, TRUNCATE, ALTER** **TCL: COMMIT, ROLLBACK, SAVEPOINT** **DCL: GRANT, REVOKE**  **DML**  DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.  SELECT – Retrieves data from a table INSERT -  Inserts data into a table UPDATE – Updates existing data into a table DELETE – Deletes all records from a table  **DDL**  DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.  CREATE – Creates objects in the database ALTER – Alters objects of the database DROP – Deletes objects of the database TRUNCATE – Deletes all records from a table and resets table identity to initial value.  **DCL**  DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.  GRANT – Gives user’s access privileges to database REVOKE – Withdraws user’s access privileges to database given with the GRANT command  **TCL**  TCL is abbreviation of **Transactional Control Language**. It is used to manage different transactions occurring within a database.  COMMIT – Saves work done in transactions ROLLBACK – Restores database to original state since the last COMMIT command in transactions Query types The Most Recent Member query you created is called a Select query. Access lets you create five different types of query:  **Select queries.** Used to retrieve data from one or more tables and display the results in a datasheet, which you can save or modify. You can also use Select queries to group records and calculate sums, averages and so on.  **Parameter queries.** For creating on-the-fly queries which prompt the user for criteria at the time the query is run. For example, you can create a parameter query that answers the question: Which countries have a population greater than X and less than Y? Each time you run the query, it will prompt you for the values of X and Y. Thus you can use the same query repeatedly to discover different information.  **Crosstab queries.** Used to summarise data from one field and group it in tabular fashion according to two criteria.  **Action queries.** Queries that make changes to the records in a table. There are four type of action queries: *Delete queries* remove records from a table; *Update queries* make global changes to a group of records in a table; *Append queries* add records from one or more tables to the end or one or more tables; *Make-table queries* create a new table from all or part of the data in an existing table.  **SQL queries.** A query created using SQL, which is a highly advanced querying language. SQL queries give you enormous flexibility, but require a high degree of expertise to use effectively. |

**Difference between sql server 2005 and sql server 2008**

**Duplicate Rows**

Select Student\_id, Student\_name, Department , COUNT(\*)

from tblstudent group by Student\_ID,Student\_name,Department

having COUNT(\*)>1

**SQL SERVER 2005:**  
1.Both are combined as SSMS(Sql Server management Studio).  
2.XML datatype is introduced.  
3.We can create 2(pow(20))-1 databases.  
4.Exception Handling  
5.Varchar(Max) data type  
6.DDL Triggers  
7.DataBase Mirroring  
8.RowNumber function for paging  
9.Table fragmentation  
10.Full Text Search  
11.Bulk Copy Update  
12.Cant encrypt  
13.Can Compress tables and indexes.(Introduced in 2005 SP2)  
14.Datetime is used for both date and time.  
15.Varchar(max) and varbinary(max) is used.  
16.No table datatype is included.  
17.SSIS is started using.  
18.CMS is not available.  
19.PBM is not available.  
  
**SQL SERVER 2008:**  
1.Both are combined as SSMS(Sql Server management Studio).  
2.XML datatype is used.  
3.We can create 2(pow(20))-1 databases.  
4.Exception Handling  
5.Varchar(Max) data type  
6.DDL Triggers  
7.DataBase Mirroring  
8.RowNumber function for paging  
9.Table fragmentation  
10.Full Text Search  
11.Bulk Copy Update  
12.Can encrypt the entire database introduced in 2008.  
--check it(<http://technet.microsoft.com/en-us/library/cc278098>(SQL.100).aspx)  
(<http://www.sqlservercentral.com/articles/Administration/implementing_efs/870/>)   
(<http://www.kodyaz.com/articles/sql-server-2005-database-encryption-step-by-step.aspx>)  
(<http://www.sql-server-performance.com/articles/dev/encryption_2005_1_p1.aspx>)  
(<http://geekswithblogs.net/chrisfalter/archive/2008/05/08/encrypt-documents-with-sql-server.aspx>)  
13.Can compress tables and indexes.  
-<http://www.mssqltips.com/tip.asp?tip=1582>  
14.Date and time are seperately used for date and time datatype,geospatial and timestamp with internal timezone   
is used.  
15.Varchar(max) and varbinary(max) is used.  
16.Table datatype introduced.  
17.SSIS avails in this version.  
18.Central Management Server(CMS) is Introduced.  
-<http://msdn.microsoft.com/en-us/library/bb934126.aspx>  
-<http://www.sqlskills.com/BLOGS/KIMBERLY/post/SQL-Server-2008-Central-Management-Servers-have-you-seen-these.aspx>  
19.Policy based management(PBM) server is Introduced.  
-<http://www.mssqltips.com/tip.asp?tip=1492>  
-<http://msdn.microsoft.com/en-us/library/bb510667.aspx>

**What's the difference between DELETE TABLE and TRUNCATE TABLE commands?**

DELETE TABLE is a logged operation, so the deletion of each row gets logged in the transaction log, which makes it slow. TRUNCATE TABLE also deletes all the rows in a table, but it won't log the deletion of each row, instead it logs the deallocation of the data pages of the table, which makes it faster. Of course, TRUNCATE TABLE can be rolled back.

**What's the difference between a primary key and a unique key?**

Both primary key and unique enforce uniqueness of the column on which they are defined. But by default primary key creates a clustered index on the column, where are unique creates a nonclustered index by default. Another major difference is that, primary key doesn't allow NULLs, but unique key allows one NULL only.

|  |
| --- |
| **What is difference between a PROCEDURE & FUNCTION ?** |
| **Stored procedure** A stored procedure is a program (or procedure) which is physically stored within a database. They are usually written in a proprietary database language like PL/SQL for Oracle database or PL/PgSQL for PostgreSQL. The advantage of a stored procedure is that when it is run, in response to a user request, it is run directly by the database engine, which usually runs on a separate database server. As such, it has direct access to the data it needs to manipulate and only needs to send its results back to the user, doing away with the overhead of communicating large amounts of data back and forth.  **User-defined function** A user-defined function is a routine that encapsulates useful logic for use in other queries. While views are limited to a single SELECT statement, user-defined functions can have multiple SELECT statements and provide more powerful logic than is possible with views.  **User defined functions** have 3 main categories   1. **Scalar-valued function** - returns a scalar value such as an integer or a timestamp. Can be used as column name in queries 2. **Inline function** - can contain a single SELECT statement. 3. **Table-valued function** - can contain any number of statements that populate the table variable to be returned. They become handy when you need to return a set of rows, but you can't enclose the logic for getting this rowset in a single SELECT statement.   **Differences between Stored procedure and functions**  <http://www.c-sharpcorner.com/UploadFile/skumaar_mca/Diff-Proc-Func08062009071904AM/Diff-Proc-Func.aspx>  1.Stored procedure will be used for perform specific tasks  Normally functions will be used for computing value  2.Stored procedures may or may not return values  But function should return value  3.Stored procedure cannot be used in the select/where/having clause  But function can be called from select/where/having clause                            SELECT \* FROM fn\_EmployeeHistory (3) Ãƒ  its will return multi value.  4. Stored procedure can run independently. It can be executed using EXECUTE or EXEC command  But function cannot run independently  5.Temporary table (derived) cannot be created on function.  But it can be created in stored procedures  6.Stored procedure can call the user defined functions  But the function cannot call the stored procedures.  7. Stored procedures can have input and output parameters.  But the function can have only input parameters.  8.Stored procedures can have select and all DML operations.  But the function can do only select operation.  9. Function cannot have the transaction statements.->The transaction statement cannot be used in the function. Normally we won't do any DML operations in the function.  Stored procedure can use transaction statements.- The transaction statement can be used inside the stored procedures.  10.Stored procedures can use all the data types available in sql server.  But the function cannot use the ntext, image and timestamp data types as return type.  11.The table variable is one of the performances tuning mechanism. Because it takes minimum resources and it uses the memory location for store the data. (Recommended for minimum rows)  It can be created and do the operations. But it cannot be the return type.  But the function can create, update and delete the table variable. It can return table variable.  It can be created and can do all the DML operations and it can be the return type. That is called the multi valued table function.  12.Stored procedure allows getdate () or other non-deterministic functions can be allowed.  But the function won't allow the non-deterministic functions.  Stored procedure  **CREATE** PROCEDURE dbo.StoredProcedure1        (       @parameter1 datatype = default value,       @parameter2 datatype OUTPUT      )   \*/   **AS**      /\* SET NOCOUNT ON \*/      **RETURN**  **CREATE** **FUNCTION** dbo.Function1      (      /\*      @parameter1 datatype = default value,      @parameter2 datatype      \*/      )   RETURNS /\* datatype \*/   **AS**      BEGIN       /\* sql statement ... \*/      **RETURN** /\* value \*/      END  **What is database replicaion? What are the different types of replication you can set up in SQL Server?**  Replication is the process of copying/moving data between databases on the same or different servers. SQL Server supports the following types of replication scenarios:      \* Snapshot replication     \* Transactional replication (with immediate updating subscribers, with queued updating subscribers)     \* Merge replication   |  |  | | --- | --- | | |  | | --- | |  | |   **Explian different types of BACKUPs avaialabe in SQL Server? Given a particular scenario, how would you go about choosing a backup plan?**  Types of backups you can create in SQL Sever 7.0+ are Full database backup, differential database backup, transaction log backup, filegroup backup. Check out the BACKUP and RESTORE commands in SQL Server books online. Be prepared to write the commands in your interview. Books online also has information on detailed backup/restore architecture and when one should go for a particular kind of backup.  **As a part of your job, what are the DBCC commands that you commonly use for database maintenance?**  DBCC CHECKDB, DBCC CHECKTABLE, DBCC CHECKCATALOG, DBCC CHECKALLOC, DBCC SHOWCONTIG, DBCC SHRINKDATABASE, DBCC SHRINKFILE etc. But there are a whole load of DBCC commands which are very useful for DBAs. Check out SQL Server books online for more information |

**SQL injection**

SQL Injection is used to hack the websites by changing the backend SQL statements, using this technique the hacker can steal the data from database and also delete and modify it.

select \* from selfjoin

where EmpName='R';delete \* from CUSTOMER--'

**How to get all database tables in sql server**

SELECT \* FROM information\_schema.tables

**How to find everything in DB**

SELECT sobjects.name

FROM sysobjects sobjects

WHERE sobjects.xtype = 'U'

Here is a list of other object types you can search for as well:

* C: Check constraint
* D: Default constraint
* F: Foreign Key constraint
* L: Log
* P: Stored procedure
* PK: Primary Key constraint
* RF: Replication Filter stored procedure
* S: System table
* TR: Trigger
* U: User table
* UQ: Unique constraint
* V: View
* X: Extended stored procedure

Web services

**Web services are open standard ( XML, SOAP, HTTP etc.) based Web applications that interact with other web applications for the purpose of exchanging data**

XML is used to [tag](http://www.webopedia.com/TERM/T/tag.html) the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available.

**Advantages**

Web Services offer many benefits over other types of distributed computing architectures.

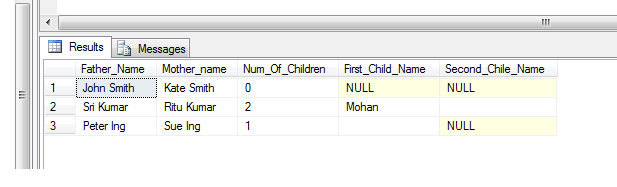
* Interoperability - This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their solutions. Services developed are likely, therefore, to have a longer life-span, offering better return on investment of the developed service. Web Services also let developers use their preferred programming languages. In addition, thanks to the use of standards-based communications methods, Web Services are virtually platform-independent.
* Usability - Web Services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web Services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.
* Reusability - Web Services provide not a component-based model of application development, but the closest thing possible to zero-coding deployment of such services. This makes it easy to reuse Web Service components as appropriate in other services. It also makes it easy to deploy legacy code as a Web Service.
* Deployability - Web Services are deployed over standard Internet technologies. This makes it possible to deploy Web Services even over the fire wall to servers running on the Internet on the other side of the globe. Also thanks to the use of proven community standards, underlying security (such as SSL) is already built-in.

DB

NULL is unknown value where as Emprty string is a value with zero length

NULL is the absence of value, and usually indicates something meaningful, such as unknown or not (yet) determined.

Re: Difference Between Null and Empty in Sql Server?   
Posted on: 05 Nov 2013

You may correlate NULL-EMPTY case by child birth scenario.  
  
NULL Case: Child is not born yet  
EMPTY Case: Child is born but we didn't give any name to him  
  
See how this data will look in a database table:  
  


null is the absence of any value (it is set to nothing), empty would be setting the value to an empty string or empty character ("" or '').

insert into **employee(emp\_no,emp\_fname,emp\_lname,dept\_no) values(29346,null,'james','d2')**10102 ann jones d3  
18316 john barrimore d1  
25348 matthew smith d3  
29346 james james d2  
29347 NULL james d2  
  
insert into **employee(emp\_no,emp\_lname,dept\_no) values(29346,'james','d2')**  
10102 ann jones d3  
18316 john barrimore d1  
25348 matthew smith d3  
29346 james james d2  
29347 NULL james d2

NULL is the absence of value, and usually indicates something meaningful, such as unknown or not (yet) determined. For example, if I start a project today, the StartDate is 2012-02-25. If I don't know how long the project is going to take, what should the EndDate be? I might have some idea what theProjectedEndDate may be, but I would set the EndDate to NULL, and update it when the project is complete.

'' is a zero-length (or "empty") string. It is not technically the absence of data, since it might actually be meaningful. For example, if I don't have a middle name, depending on your data model '' might make more sense than NULL since the latter implies unknown but '' can imply that it is known that I don't

Distinction between NULL and empty string may not be obvious because they both can mean 'no value' if you decide to. It depends entirely up to you but using NULL would be better mainly because it is a special case for databases which are designed to handle NULLs quickly and efficiently (much faster than strings). If you use it instead of an empty string your queries will be faster and more reliable.

**Can date column be null**

Yes Date and datetime can be null if we supply null in values and if we supply empty string it will take default date like 1900-01-01

**SELECT DECODE (Store\_Name,   
  'Los Angeles', 'LA',   
  'San Francisco', 'SF',   
  'San Diego', 'SD', 'Others') Area, Sales, Txn\_Date   
FROM Store\_Information;**

|  |  |  |
| --- | --- | --- |
| **Area** | **Sales** | **Txn\_Date** |
| **LA** | **1500** | **Jan-05-1999** |
| **SD** | **250** | **Jan-07-1999** |
| **SF** | **300** | **Jan-08-1999** |
| **Others** | **700** | **Jan-08-1999** |

Similar to the [**UNION**](http://www.1keydata.com/sql/sqlunion.html) command, **INTERSECT** also operates on two SQL statements. The difference is that, while **UNION** essentially acts as an **OR** operator

|  |  |  |
| --- | --- | --- |
| **Store\_Name** | **Sales** | **Txn\_Date** |
| Los Angeles | 1500 | Jan-05-1999 |
| San Diego | 250 | Jan-07-1999 |
| Los Angeles | 300 | Jan-08-1999 |
| Boston | 700 | Jan-08-1999 |

Table ***Internet\_Sales***

|  |  |
| --- | --- |
| **Txn\_Date** | **Sales** |
| Jan-07-1999 | 250 |
| Jan-10-1999 | 535 |
| Jan-11-1999 | 320 |
| Jan-12-1999 | 750 |

and we want to find out all the dates where there are both store sales and internet sales. To do so, we use the following SQL statement:

**SELECT Txn\_Date FROM Store\_Information  
INTERSECT  
SELECT Txn\_Date FROM Internet\_Sales;**

Result:

|  |
| --- |
| **Txn\_Date** |
| **Jan-07-1999** |

**CASE** is used to provide if-then-else type of logic to SQL. There are two formats: The first is a **Simple CASE**expression, where we compare an expression to static values. The second is a **Searched CASE** expression, where we compare an expression to one or more logical conditions.

**SELECT Store\_Name, Txn\_Date, CASE  
  WHEN Sales >= 1000 THEN 'Good Day'  
  WHEN Sales >= 500 THEN 'OK Day'  
  ELSE 'Bad Day'  
  END  
"Sales Status"  
FROM Store\_Information;**

Result:

|  |  |  |
| --- | --- | --- |
| **Store\_Name** | **Txn\_Date** | **Sales Status** |
| **Los Angeles** | **Jan-05-1999** | **Good Day** |
| **San Diego** | **Jan-07-1999** | **Bad Day** |
| **San Francisco** | **Jan-08-1999** | **Bad Day** |
| **Boston** | **Jan-08-1999** | **OK Day** |

**AUTO INCREMENT** interval value is controlled by the MySQL

**CREATE TABLE USER\_TABLE   
(Useridint NOT NULL AUTO\_INCREMENT,   
Last\_Namevarchar(50),   
First\_Namevarchar(50),   
PRIMARY KEY (Userid));**

Upon creation, there is no data in this table.

We insert the first value:

**INSERT INTO USER\_TABLE VALUES ('Perry', 'Jonathan');**

Now the table has the following values:

Table ***USER\_TABLE***

|  |  |  |
| --- | --- | --- |
| Userid | Last\_Name | First\_Name |
| 1 | Perry | Jonathan |

**IDENTITY**

**CREATE TABLE USER\_TABLE   
(Useridint PRIMARY KEY IDENTITY(2,1),   
Last\_Namenvarchar(50),   
First\_Namenvarchar(50));**

Upon creation, the table is empty.

We will insert the first value:

**INSERT INTO USER\_TABLE VALUES ('Washington', 'George');**

Now the table has the following values:

Table ***USER\_TABLE***

|  |  |  |
| --- | --- | --- |
| Userid | Last\_Name | First\_Name |
| 2 | Washington | George |

userid is 2 because we had spe

**NULL**

In SQL, **NULL** means that data does not exist. NULL does not equal to 0 or an empty string. Both 0 and empty string represent a value, while **NULL** has no value.

Any mathematical operations performed on **NULL** will result in **NULL**. For example,

**10 + NULL = NULL**

**Aggregate functions such as**[**SUM**](http://www.1keydata.com/sql/sql-sum.html)**,**[**COUNT**](http://www.1keydata.com/sql/sqlcount.html)**,**[**AVG**](http://www.1keydata.com/sql/sql-average.html)**,**[**MAX**](http://www.1keydata.com/sql/sql-max.html)**, and**[**MIN**](http://www.1keydata.com/sql/sql-min.html)**exclude NULL values. This is not likely to cause any issues for SUM, MAX, and MIN. However, this can lead to confusion with AVG and COUNT.**

**Table *Sales\_Data***

|  |  |
| --- | --- |
| **Store\_Name** | **Sales** |
| **Store A** | **300** |
| **Store B** | **200** |
| **Store C** | **100** |
| **Store D** | **NULL** |

**Below are the results for each aggregate function:**

**SELECT SUM (Sales), AVG (Sales), MAX (Sales), MIN (Sales), COUNT (Sales)  
FROM Sales\_Date;**

**Result:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SUM (Sales)** | **AVG (Sales)** | **MAX (Sales)** | **MIN (Sales)** | **COUNT (Sales)** |
| **600** | **200** | **300** | **100** | **3** |

The **ISNULL( )** function is available in both SQL Server and MySQL. However, their uses are different:

**SQL Server**

In SQL Server, the **ISNULL( )** function is used to replace **NULL** value with another value.

For example, if we have the following table,

Table ***Sales\_Data***

|  |  |
| --- | --- |
| **Store\_Name** | **Sales** |
| Store A | 300 |
| Store B | NULL |

The following SQL,

**SELECT SUM (ISNULL(Sales,100)) FROM Sales\_Data; in SQL server**

**Isull in microsoft**

**Same IFNULL used in** MySQL

Same NVL uses in Oracle

**Coalesce Function**

|  |
| --- |
| The **COALESCE** function in SQL returns the first non-NULL expression among its arguments. The syntax for**COALESCE** is as follows:Retunrs first no null value |

**COALESCE ("expression 1", "expressions 2", ...)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Business\_Phone** | **Cell\_Phone** | **Home\_Phone** |
| Jeff | 531-2531 | 622-7813 | 565-9901 |
| Laura | NULL | 772-5588 | 312-4088 |
| Peter | NULL | NULL | 594-7477 |

We can use the **COALESCE** function to achieve our goal:

**SELECT Name, COALESCE (Business\_Phone, Cell\_Phone, Home\_Phone) Contact\_Phone   
FROM Contact\_Info;**

Result:

|  |  |
| --- | --- |
| **Name** | **Contact\_Phone** |
| **Jeff** | **531-2531** |
| **Laura** | **772-5588** |
| **Peter** | **594-7477** |
| **NULLIF** | **NULLIF** function takes two arguments. If the two arguments are equal, then NULL is returned. Otherwise, the first argument is returned. The syntax for **NULLIF** is as follows:  Table ***Sales\_Data***   |  |  |  | | --- | --- | --- | | **Store\_Name** | **Actual** | **Goal** | | Store A | 50 | 50 | | Store B | 40 | 50 | | Store C | 25 | 30 | |

**SELECT Store\_Name, NULLIF (Actual, Goal) FROM Sales\_Data;**

Result:

|  |  |
| --- | --- |
| **Store\_Name** | **NULLIF (Actual, Goal)** |
| **Store A** | **NULL** |
| **Store B** | **40** |
| **Store C** | **25** |
|  |  |

**To Check multiple table existence**

SELECT \* FROM information\_schema.tables where TABLE\_NAME in ('Employee\_Test','Test4')

Or

IF object\_ID('dbo.Employee\_Test') is not null

PRINT 'ok'

ELSE

PRINT 'Not accounted for'

without order by max salary

ddl,dml,

trnucate vs delete

coumn null in a table

Profiler

**DELETE**   
1. DELETE is a DML Command.   
2. DELETE statement is executed using a row lock, each row in the table is locked for deletion.   
3. We can specify filters in where clause   
4. It deletes specified data if where condition exists.   
5. Delete activates a trigger because the operation are logged individually.   
6. Slower than truncate because, it keeps logs.   
7. Rollback is possible.   
  
**TRUNC ATE**   
1. TRUNCATE is a DDL command.   
2. TRUNCATE TABLE always locks the table and page but not each row.   
3. Cannot use Where Condition.   
4. It Removes all the data.   
5. TRUNCATE TABLE cannot activate a trigger because the operation does not log individual row deletions.   
6. Faster in performance wise, because it doesn't keep any logs.   
7. Rollback is not possible.   
  
DELETE and TRUNCATE both can be rolled back when used with TRANSACTION.

### DDL

**Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

### DML

**Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples:

* SELECT - retrieve data from the a database
* INSERT - insert data into a table
* UPDATE - updates existing data within a table
* DELETE - deletes all records from a table, the space for the records remain
* MERGE - UPSERT operation (insert or update)
* CALL - call a PL/SQL or Java subprogram
* EXPLAIN PLAN - explain access path to data
* LOCK TABLE - control concurrency

### DCL

**Data Control Language** (DCL) statements. Some examples:

* GRANT - gives user's access privileges to database
* REVOKE - withdraw access privileges given with the GRANT command

Table exists or not

**IF EXISTS (SELECT \***

**FROM sysobjects**

**WHERE xtype='U' AND name ='MakeMyTrip (India) Pvt\_ Ltd\_$Payment - Card')**

**SELECT 'tablename exists.'**

**ELSE**

**SELECT 'tablename does not exist.'**

Nth no salary without order by and max

select \* from empy E1 where 5 =

(select count(distinct E2.salary) from empy E2 where E1.salary <=E2.salary)

WITH Salaries AS

(

SELECT

Salary, NTILE(10) OVER(ORDER BY Salary DESC) AS 'NTile'

FROM

empy

)

SELECT

Salary

FROM

Salaries

WHERE

NTile = 5

SELECT \* FROM (

SELECT ROW\_NUMBER() OVER (ORDER BY SALARY DESC) AS rownumber,Salary

FROM Employee )

AS foo

WHERE rownumber = 5

**Insert 1000 records in table in ine go**

CREATE TABLE TEST( A INT, B varchar(23))

Declare @id int

select @id=1

while @id >=1 and @id <=1000

begin

insert into Test values(@id,'Ram'+convert(varchar(4),@id))

select @id=@id+1

end

Select \* from TEST

drop table Test

**Update 2 different column from different conditions**

update TestData   
SET Column1 = case when Column1 = 3 then 9 else Column1 end,    
Column2 = case when Column2 = 'E' then 'Z' else Column2 end

UPDATE

    A

SET

    A.Column1 = N.NewColumn1,

    A.Column2 = M.NewColumn2

FROM

   TestData A

   JOIN

   (VALUES

      ('1', '2'),

      ('2', '1')

   ) N (OldColumn1, NewColumn1) ON A.Column1 = OldColumn1

   JOIN

   (VALUES

      ('A', 'Z'),

      ('B', 'Y')

   ) M (OldColumn2, NewColumn2) ON A.Column2 = OldColumn2