1. **BillPay:**

|  |  |  |
| --- | --- | --- |
| **BillId** | **Status** | **Amount** |
| 1 | In | 20 |
| 2 | In | 200 |
| 3 | Out | 100 |
| 4 | In | 250 |
| 5 | Out | 110 |

1. Considering there is no status column. Then for the output as:

|  |  |  |
| --- | --- | --- |
| **BillId** | **Amount** | **CummAmount** |
| 1 | 20 | 20 |
| 2 | 200 | 220 |
| 3 | 100 | 320 |
| 4 | 250 | 570 |
| 5 | 110 | 680 |

**Ans.**

SELECT [BillId], [Status], [Amount], (Select SUM ([Amount])

From [BillPay]

Where [BillId] <= bp.[BillId]) as CummAmount

FROM [BillPay] bp

1. With condition, like when In then add, when Out then subtract. For the output as:

|  |  |  |  |
| --- | --- | --- | --- |
| **BillId** | **Status** | **Amount** | **CummAmount** |
| 1 | In | 20 | 20 |
| 2 | In | 200 | 220 |
| 3 | Out | 100 | 120 |
| 4 | In | 250 | 370 |
| 5 | Out | 110 | 260 |

**Ans.**

SELECT [BillId], [Status], [Amount], (Select SUM (Case [Status] When 'IN' Then Amount Else -Amount end)

From [BillPay]

Where BillId<=bp.BillId) as CummAmount

FROM [BillPay] bp

1. **SellProduct**

|  |  |  |
| --- | --- | --- |
| **Customer** | **Product** | **QTY** |
| Kate | VEG | 2 |
| Kate | SODA | 6 |
| Fred | MILK | 3 |
| Fred | VEG | 24 |
| Kate | VEG | 3 |

1. For the output as:

|  |  |  |
| --- | --- | --- |
| **Customer** | **Product** | **Quantity** |
| Fred | MILK | 3 |
| Fred | VEG | 24 |
| Kate | SODA | 6 |
| Kate | VEG | 5 |

**Ans.**

SELECT Customer, Product, SUM (QTY)'Quantity'

From SellProduct

GROUP BY Customer, Product

Order BY Customer

1. For the output as:

|  |  |
| --- | --- |
| **Customer** | **No\_of\_Duplications** |
| Fred | 2 |
| Kate | 3 |

**Ans.**

Select Customer, Count(\*)[No\_of\_Duplications] From SellProduct

Group by Customer

Having count(Customer) > 1

1. **StudentMarks**

|  |  |  |
| --- | --- | --- |
| **SubjectId** | **StudentId** | **Marks** |
| 1 | 1 | 70 |
| 1 | 2 | 80 |
| 2 | 1 | 85 |
| 2 | 2 | 70 |
| 3 | 1 | 85 |
| 3 | 2 | 80 |
| 3 | 3 | 90 |

1. For the out put as:

|  |  |
| --- | --- |
| **SubjectId** | **Marks** |
| 1 | 80 |
| 2 | 85 |
| 3 | 90 |

**Ans.**

Select [SubjectId] ,Max([Marks]) as Marks

From [StudentMarks]

Group By [SubjectId]

1. For the output as:

|  |  |  |
| --- | --- | --- |
| **SubjectId** | **StudentId** | **Marks** |
| 1 | 2 | 80 |
| 2 | 1 | 85 |
| 3 | 3 | 90 |

**Ans.**

Select B.[SubjectId] , A.[StudentId], B.marks as Marks

From (Select [SubjectId], Max ([Marks]) as marks

From [StudentMarks]

Group By [SubjectId]) B

JOIN [StudentMarks] A ON B.SubjectId = A.SubjectId AND B.marks=A.Marks

Order By A. [SubjectId]

1. **Population Info**

|  |  |  |
| --- | --- | --- |
| **Name** | **District** | **Gender** |
| Mr. Zahid | Dhaka | Male |
| Mr. Tarun | Dhaka | Male |
| Mr. RR | Pabna | Male |
| Mr. ZZ | Pabna | Female |
| Mr. YY | Pabna | Female |
| Mr. PT | Dhaka | Female |
| Mr. KK | Khulna | Male |

1. For the output as :

|  |  |  |
| --- | --- | --- |
| District | Male | Female |
| Dhaka | 2 | 1 |
| Khulna | 1 | 1 |
| Pabna | 1 | 2 |

**Ans.**

Select [District], SUM(male)as Male, SUM(female) as Female

From(

Select [District],

CASE [Gender] WHEN 'Male' Then 1 ELSE 0 END as male,

CASE [Gender] WHEN 'Female' Then 1 ELSE 0 END as female

From [PopulationInfo])P

Group By [District]

**Normalization: The process of organizing data to minimize redundancy is called normalization. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them**. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

<http://support.microsoft.com/kb/283878>

<http://en.wikipedia.org/wiki/Database_normalization>

First Normal Form (1NF):

* Eliminate repeating groups in individual tables.
* Create a separate table for each set of related data.
* Identify each set of related data with a primary key.

For example a table contains telephone numbers of user. But one user may have more than one numbers.

**Second Normal Form (2NF):**

- The database must meet all the requirements of the 1NF.

- If a table has a composite key, all attributes must be related to the whole key

**Third Normal Form (3NF):**

- The database must meet all the requirements of the second normal form.

- Any field which is dependent not only on the primary key but also on another field is moved out to a separate table.

- We mainly use 3NF

**Boyce-Codd normal form (or BCNF), 4NF, 5 NF, 6NF**

**De- Normalization:** is the process of attempting to optimise the read performance of a database by adding redundant data or by grouping data.

**From which Normal form we do de-normalization:** 3NF

**Data Manipulation Language (DML):** is used to retrieve, store, modify, delete, insert and update data in database. Examples: SELECT, UPDATE, INSERT statements

**Data Definition Language (DDL):** is used to create and modify the structure of database objects in database. Examples: CREATE, ALTER, and DROP statements

**Data Control Language (DCL):** is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it. Examples: GRANT, REVOKE

**Transactional Control Language** (TCL): is used to manage different transactions occurring within a database. Examples: COMMIT, ROLLBACK statements

**Unique Key:** is one that uniquely identifies each row in a table.

**Primary Key:** The attribute or combination of attributes that uniquely identifies each row. Primary Key can be Zero, but if you set Identity on the column it normally will start at 1 rather than Zero.

**Difference between Unique Key and Primary Key:**

- Unique key allow null but primary key does not.

- One table may have more than one unique key but only one primary key

**Foreign Key:** an attribute or combination of attributes in a table whose value matches a primary key in another table.

**Composite Key:** A primary key that consists of two or more attributes is known as composite key.

**Candidate Key and Alternate Key:** A candidate key is one that can uniquely identify each row of a table. Generally a candidate key becomes the primary key of the table. If the table has more than one candidate key, one of them will become the primary key, and the rest are called alternate keys.

**Stored Procedure:** A stored procedure is a set of SQL statements compiled into a single execution plan.

Benefits:

* Reuse
* Encapsulation of business rules (single execution of multiple sql statement)
* Precompiled

**Precompiled of SP means:** Sql Server would go through a Five Step Process:  
  
1) Parse  
2) Resolve  
3) Optimize  
4) Compile and  
5) Execute   
  
These prepares a query that is to be executed. This would create an **execution plan** that Sql Server will use to retrieve the data. **The execution plan will be stored in the Buffer Cache, that is a reserved memory space for Sql Server to hold the execution plan information**, With stored procedures, the subsequent execution of the same procedure will by-pass the Five steps process and go straight to the Execution plan. **This is called as Pre-Compiled plan.**

**Function:** A function is a SQL name block which is similar to a procedure.

**Difference b2n Stored Procedure and Function:**

- Functions must return a value (using the RETURN keyword), but for stored procedures it is not mandatory.

- When return scalar value function can be used as inline in SQL Statements but stored procedures cannot.

**View:** A view can be thought of as either a virtual table or a stored query. The data accessible through a view is not stored in the database as a distinct object. What is stored in the database is a SELECT statement. The result set of the SELECT statement forms the virtual table returned by the view. A user can use this virtual table by referencing the view name in Transact-SQL statements the same way a table is referenced

- A view can reference another view.

- Any modifications, including UPDATE, INSERT, and DELETE statements, must reference columns from only one base table.

- The columns that are being modified in the view must reference the underlying data in the table columns directly. They cannot be derived in any other way, such as through: aggregate function (AVG, COUNT, SUM, MIN, MAX, etc), Union, etc.

**Trigger:** A trigger is a special kind of stored procedure that **automatically** executes when an event occurs in the database server. Triggers fire when any valid event (Insert, Update, or Delete) is fired.

**Scalar-valued Function:** Returns a single value of a scalar data type.

**Inline Table-valued Function:** Returns a row set of the SQL Server table data type using SELECT Statement. Unlike a view, functions can accept parameters.

**Multi-statement Table-valued Function:** Explicitly defines the structure of the table to return. Define column names and data types in the RETURNS clause.

**Indexes: A database index** is very much like the index in a book: the book index has an alphabetized list of topics with page numbers to the location of the data. A database index has an ordered list of values (made up of one or more table columns), with pointers to the row in which the value and its corresponding data reside.

**Deadlock: A deadlock is a situation where in two transactions wait for each other to give up their respective locks.** Transaction A attempts to update table 1 and subsequently read/update data from table 2, whereas transaction B attempts to update table 2 and subsequently read/update data from table 1. In such situations, transaction A holds locks that transaction B needs to complete its task and vice versa; neither transaction can complete until the other transaction releases locks.

In this case SQL Server selects one (which one need less rollback) of the processes as a deadlock victim and rolls back the statements.

WAITFOR DELAY '00:00:00.05'

ERROR\_NUMBER() = 1205

ROLLBACK TRANSACTION

Lock()

TableA

|  |  |
| --- | --- |
| **Fld1** | **Fld2** |
| 1 | A |
| 2 | B |
| 3 | C |
| 4 | D |

TableB

|  |  |
| --- | --- |
| **Fld1** | **Fld2** |
| 4 | E |
| 5 | F |
| 3 | C |
| 7 | H |

**INNER JOIN:** A JOIN that displays only rows that have a match in both the Joined tables is known as inner JOIN.

Select \* From TableA A

INNER JOIN TableB B ON A.Fld1 = B.Fld1 // only JOIN can be used also

|  |  |  |  |
| --- | --- | --- | --- |
| **Fld1** | **Fld2** | **Fld1** | **Fld2** |
| 3 | C | 3 | C |
| 4 | D | 4 | E |

**LEFT OUTER JOIN:** This JOIN returns all the rows from the left table in conjunction with the matching rows from the right table.

Select \* From TableA A

LEFT JOIN TableB B ON A.Fld1 = B.Fld1 // LEFT OUTER JOIN

|  |  |  |  |
| --- | --- | --- | --- |
| **Fld1** | **Fld2** | **Fld1** | **Fld2** |
| 1 | A | NULL | NULL |
| 2 | B | NULL | NULL |
| 3 | C | 3 | C |
| 4 | D | 4 | E |

**RIGHT OUTER JOIN:** This join returns all the rows from the right table in conjunction with the matching rows from the left table.

Select \* From TableA A

RIGHT JOIN TableB B ON A.Fld1 = B.Fld1 // RIGHT OUTER JOIN

|  |  |  |  |
| --- | --- | --- | --- |
| **Fld1** | **Fld2** | **Fld1** | **Fld2** |
| 4 | D | 4 | E |
| NULL | NULL | 5 | F |
| 3 | C | 3 | C |
| NULL | NULL | 7 | H |

**FULL OUTER JOIN:** This join combines left outer join and right outer join

Select \* From TableA A

FULL JOIN TableB B ON A.Fld1 = B.Fld1 // FULL OUTER JOIN

|  |  |  |  |
| --- | --- | --- | --- |
| **Fld1** | **Fld2** | **Fld1** | **Fld2** |
| 1 | A | NULL | NULL |
| 2 | B | NUL | NULL |
| 3 | C | 3 | C |
| 4 | D | 4 | E |
| NULL | NULL | 5 | F |
| NULL | NULL | 7 | H |

**CROSS JOIN:** Cross multiplication. **Here Returns 16 Rows**

Select \* From TableA A

CROSS JOIN TableB

OR

Select \* From TableA, TableB

**UNION:** Select only distinct values. It eliminates duplicate rows.

Select \* From TableA

UNION

Select \* From TableB

|  |  |
| --- | --- |
| **Fld1** | **Fld2** |
| 1 | A |
| 2 | B |
| 3 | C |
| 4 | D |
| 4 | E |
| 5 | F |
| 7 | H |

**UNION ALL:** Select all values. It does not eliminate duplicate rows.

Select \* From TableA

UNION ALL

Select \* From TableB

|  |  |
| --- | --- |
| **Fld1** | **Fld2** |
| 1 | A |
| 2 | B |
| **3** | **C** |
| 4 | D |
| 4 | E |
| 5 | F |
| **3** | **C** |
| 7 | H |

**IN Vs EXISTS:** WHEN YOU USE 'IN', WHILE CHECKING FOR WHERE CONDITION SQL SERVER ENGINE DOES WHOLE TABLE SCAN. IF YOU USE 'EXISTS' AS SOON AS ENGINE FINDS THE REQUIRED ROW IT WILL STOP EXECUTING QUERY AND GOING FURTHER SCANNING TABLE.

SO BASICALLY EXISTS IS FASTER AS COMPARED TO IN.

**HAVING Clause Vs WHERE Clause:**

- HAVING clause is used only with select statement WHERE clause can be used with SELECT, DELETE, UPDATE statements.

- HAVING clause cannot be used without GROUP BY Clause but WHERE Clause can.

- WHERE clause cannot contain aggregate functions but HAVING Clause can.

**VARCHAR Vs NVARCHAR:**

|  |  |  |
| --- | --- | --- |
| **Difference Criteria** | **VARCHAR** | **NVARCHAR** |
| Data Type | Non-Unicode Data | Unicode Data |
| Character size | 1 byte | 2 bytes |
| Maximum Length | 8000 bytes | 4000 bytes |

**When use NVARCHAR and when use VARCHAR:**

**Varchar vs Char:**

**char [ ( n ) ] :** Fixed-length, non-Unicode string data. n defines the string length and must be a value from 1 through 8,000. The storage size is n bytes. The ISO synonym for char is character.

**varchar [ ( n | max ) ]:** Variable-length, non-Unicode string data. n defines the string length and can be a value from 1 through 8,000. **max** indicates that the maximum storage size is 2^31-1 bytes (2 GB). The storage size is the actual length of the data entered + 2 bytes.

* Use **char** when the sizes of the column data entries are consistent.
* Use **varchar** when the sizes of the column data entries vary considerably.
* Use **varchar(max)** when the sizes of the column data entries vary considerably, and the size might exceed 8,000 bytes.

**SQL profiler:** SQL Profiler is a graphical tool that allows monitoring events in an instance of SQL Server You can capture and save data about each event to a file or SQL Server table to analyze later. For example, you can monitor a production environment to see which stored procedures is hampering performance by executing too slowly.

* Monitor the performance of an instance of SQL Server.
* Debug Transact-SQL statements and stored procedures.
* Identify slow-executing queries.

**Derived Table:** Sometimes querying data is not that simple and there may be the need to create temporary tables or views to predefine how the data should look prior to its final output.  Unfortunately there are problems with both of these approaches if you are trying to query data on the fly.

With the temporary tables approach you need to have multiple steps in your process, first to create the temporary table, then to populate the temporary table, then to select data from the temporary table and lastly cleanup of the temporary table.

With the view approach you need to predefine how this data will look, create the view and then use the view in your query. **Solution is derived table.** With SQL Server you have the ability to create derived tables on the fly

**Derived table is OG.**

SELECT   OG.OrderGroup,   
         COUNT(OG.OrderGroup) AS OrderGroupCount   
FROM     (SELECT   o.CustomerID,   
                   SUM(UnitPrice \* Quantity) AS TotalSales,   
                   CASE    
                     WHEN SUM(UnitPrice \* Quantity)    
                       BETWEEN 10001 AND 15000 THEN 'Medium'   
                     WHEN SUM(UnitPrice \* Quantity)    
                       BETWEEN 15001 AND 20000 THEN 'Large'   
                     WHEN SUM(UnitPrice \* Quantity) > 20000 THEN 'Very Large'   
                   END AS OrderGroup   
          FROM     [Order Details] AS od   
                   INNER JOIN Orders AS o   
                     ON od.OrderID = o.OrderID   
          GROUP BY o.CustomerID) AS OG   
GROUP BY OG.OrderGroup

**Common Table Expression (CTE):** A common table expression (CTE) can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement. A CTE is similar to a derived table in that it is not stored as an object and lasts only for the duration of the query. Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query.

A **CTE** can be used to:

* Create a recursive query. For more information
* Substitute for a view when the general use of a view is not required; that is, you do not have to store the definition in metadata.
* Reference the resulting table multiple times in the same statement.

**SQL Injection:** SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution.

* Can be solved by Parameterized SP
* Validate input from code

**How to write optimize code:**

* Try to avoid cursor
* Try to avoid unnecessary loop
* Try to avoid joining, specially outer join

**Cluster:** A Microsoft SQL Server Cluster is nothing more than a collection of two or more physical servers (generally called nodes) that work together and represent themselves as a single virtual server to a network.

It allows identical access to shared storage that provides the disk resources required to store the database files.

**Cursor:** A cursor is a set of rows together with a pointer that identifies a current row.   
  
In other word, Cursor is a database object used by applications to manipulate data in a set on a row-by-row basis, its like recordset in the ASP and visual basic.

**Master data: Master data**, which may include reference data, is information that is key to the operation of business and is focused primarily. This key business information may include data about customers, products, employees, materials, suppliers, etc. which often turns out to be non-transactional in nature.

Master data is often used by several functional groups and stored in different data systems across an organization and may or may not be referenced centrally Most software systems have lists of data that are shared and used by several of the applications that make up the system. For example, a typical ERP system as a minimum will have a customer Master, an Item Master, and an Account Master. This master data is often one of the key assets of a company. It's not unusual for a company to be acquired primarily for access to its Customer Master data.

For Roll Back:

* Transaction
* Commit Transaction
* RollBack Transaction

For id of latest inserted Data:

* SCOPE\_IDENTITY()