SQL Concept Questions

Q1. DBMS? RDBMS?

A **Database Management System** (**DBMS**) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

RDBMS stands for **Relational Database Management System**. RDBMS stores the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables. Example: SQL Server

Q2. Database?

Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways.

Q3. Tables? Records? Fields?

A **table** is a set of data that are organized in a model with Columns and Rows. Columns can be categorized as vertical, and Rows are horizontal. A table has a specified number of columns called **fields** but can have any number of rows which is called **record**.

Q4. SQL?

SQL stands for **Structured Query Language**, and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.Standard SQL Commands are Select.

Q5. Key? Primary/Unique/Foreign Key?

A Primary Key is used to uniquely identify each row in a table and does not allow null values.

A Unique Key identifies each row in the table uniquely. it allows null values.

There can be many unique constraints defined per table, but only one Primary key constraint defined per table.

A Foreign Key is one or more columns whose values are based on the primary key values from another table. For example:

```
CREATE TABLE Students (
    ID INT NOT NULL
    Name VARCHAR(255)
    LibraryID INT
    PRIMARY KEY (ID)
    FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
)
```

Q6. Join? the types of Join?

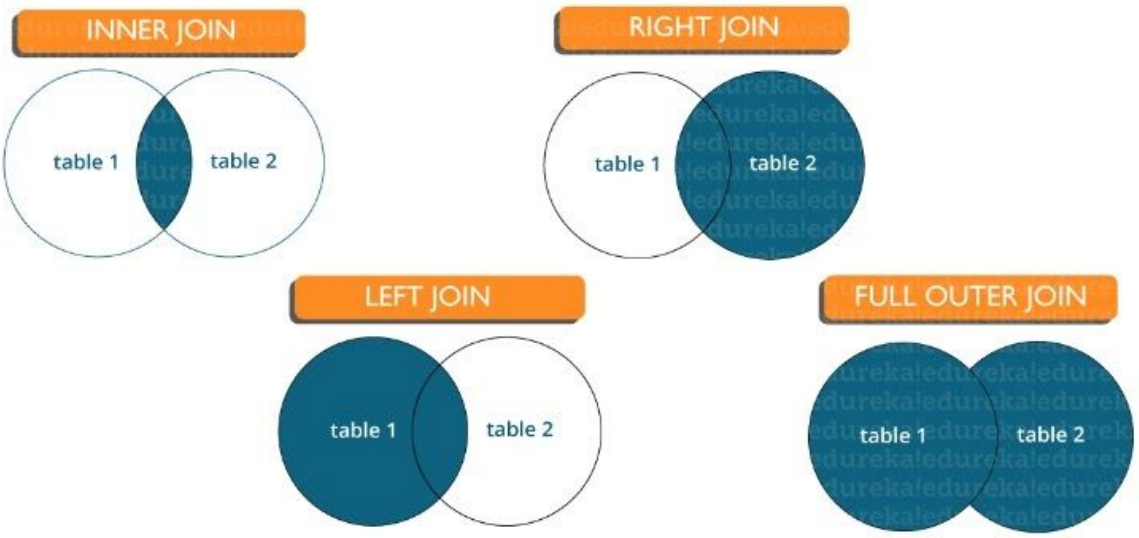
Join is a keyword used to query data from more tables based on the relationship between the fields of the tables. There are various types of join which can be used to retrieve data and it depends on the relationship between tables.

Inner Join (Join): return rows when there is at least one match of rows between the tables.

Right Join: return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

Left Join: return rows which are common between the tables and all rows of the Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

Full Join (full outer join): return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.



Q7. Self-Join?

A **Self-join** is a case of regular join where a table is joined to itself based on some relation between its own column(s). **Self-join** uses INNER JOIN or LEFT JOIN and a table alias is used to assign different names to the table within the query.

Q8. Cross-Join?

Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

SELECT stu.name, sub.subject

FROM students AS stu CROSS JOIN subjects AS sub;

Q9. Index?

A database **index** is a data structure that provides quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure. **Unique and Non-Unique Index:**

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index.

Non-unique indexes, on the other hand, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

Clustered and Non-Clustered Index:

Clustered indexes are indexes whose order of the rows in the database correspond to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas multiple non-clustered indexes can exist in the table.

The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the database in the same order as the corresponding keys appear in the clustered index.

Clustering index can improve the performance of most query operations because they provide a linear-access path to data stored in the database.

Q10. The difference between Clustered and Non-clustered index?

The differences can be broken down into **three small factors** - Clustered index modifies the way records are stored in a database based on the indexed column. Non-clustered index creates a separate entity within the table which references the original table; Clustered index is used for easy and speedy retrieval of data from the database, whereas, fetching records from the non-clustered index is relatively slower; In SQL, a table can have a single clustered index whereas it can have multiple non-clustered indexes.

Q11. Common clauses used with SELECT query?

WHERE clause in SQL is used to filter records that are necessary, based on specific conditions.

ORDER BY clause in SQL is used to sort the records based on some field(s) in ascending (ASC) or descending order (DESC).

```
SELECT *   FROM myDB.students    WHERE graduation_year = 2019
ORDER BY studentID DESC;
```

GROUP BY clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarized results from the database.

HAVING clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since WHERE clause cannot filter aggregated records.

```
SELECT COUNT(studentId), country           FROM myDB.students
WHERE country != "INDIA"
GROUP BY country
HAVING COUNT(studentID) > 5;
```

OR

```
SELECT Columns | *                  FROM   Table_Name
[WHERE  Search_Condition]
[GROUP BY Group_By_Expression]
[HAVING Search_Condition]
[ORDER BY Order_By_Expression [ASC|DESC]]
```

Q12. What are UNION, MINUS and INTERSECT commands?

The **UNION** operator combines and returns the result-set retrieved by two or more SELECT statements.

The **MINUS** operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.

The **INTERSECT** clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

Certain conditions need to be met before executing either of the above statements in SQL. Each SELECT statement within the clause must have the same number of columns; The columns must also have similar data types; The columns in each SELECT statement should necessarily have the same order.

Q13. UNION, UNION ALL?

Union and union all are used to merge rows from two or more tables.

Union set operator removes duplicate records. Whereas union all does not.

Union operators sorts the data in ascending order. union all does not.

Union all is faster than union operators.

Q14. Cursor? How to use a Cursor?

A database cursor is a control structure that allows for traversal of records in a database. Cursors, in addition, facilitates processing after traversal, such as retrieval, addition and deletion of database records. They can be viewed as a pointer to one row in a set of rows. Working with SQL Cursor:

DECLARE a cursor after any variable declaration. The cursor declaration must always be associated with a SELECT Statement.

Open cursor to initialize the result set. The OPEN statement must be called before fetching rows from the result set.

FETCH statement to retrieve and move to the next row in the result set.

Call the CLOSE statement to deactivate the cursor.

Finally use the **DEALLOCATE** statement to delete the cursor definition and release the associated resources. Please see below example:

```
DECLARE @name VARCHAR(50)       /* Declare All Required Variables */
DECLARE db_cursor CURSOR FOR   /* Declare Cursor Name*/
SELECT name FROM myDB.students WHERE parent_name IN ('Sara', 'Ansh')
OPEN db_cursor      /* Open cursor and Fetch data into @name */
FETCH next  FROM db_cursor INTO @name
CLOSE db_cursor    /* Close the cursor and deallocate the resources */
DEALLOCATE db_cursor
```

Q15. View? modification to view?

A view in SQL is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

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: An aggregate function (AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR and VARP); A computation; the column cannot be computed from an expression using other columns; Columns formed using set operators (UNION, UNION ALL, CROSSJOIN, EXCEPT, and INTERSECT) amount to a computation and are also not updatable; The columns that are being modified cannot be affected by GROUP BY, HAVING, or

DISTINCT clauses; TOP cannot be used anywhere in the select_statement of the view when WITH CHECK OPTION is also specified.

Q16. Normalization? Denormalization?

Normalization is the process of organizing the columns, tables of a database to minimize the redundancy of data. Normalization involves dividing large tables into smaller tables and defining relationships between them. The different types of Normalization Forms are:

First Normal Form: This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

Second Normal Form: Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

Third Normal Form: This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.

Fourth Normal Form: Meeting all the requirements of third normal form and it should not have multi-valued dependencies.

Denormalization is the process of optimizing the read performance of a database by adding redundant data or by grouping data. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in the query processor by an over-normalized structure.

Q17. relationship?

Database Relationship is defined as the connection between the tables in a database. There are various data basing relationships, and they are as follows:.

One to One Relationship.

One to one relationship is a simple reference between two tables. Consider Customer and Address tables as an example. A customer can have only one address and an address references only one customer.

One to Many Relationship.

One-to-many relationships can be implemented by splitting the data into two tables with a primary key and foreign key relationship. Here the row in one table is referenced by one or more rows in the other table. An example is the Employees and Departments table, where the row in the Departments table is referenced by one or more rows in the Employees table.

Many to Many Relationships.

Many-to-Many relationship is created between two tables by creating a junction table with the key from both the tables forming the composite primary key of the junction table. An example is Students, Subjects and Stud_Sub_junc tables. A student can opt for one or more subjects in a year. Similarly a subject can be opted by one or more students. So a junction table is created to implement the many-to-many relationship.

Self-Referencing Relationship.

a self-referencing relationship (also known as a recursive relationship) does not exist between a pair of tables. It is a relationship that exists between the records within a table, for example, a given record in the table is related to other records within the table. a self-referencing relationship can be one-to-one, one-to-many, or many-to-many.

Q18. Query? SubQuery? types of SubQuery?

A **DB query** is a code written in order to get the information back from the database.
A **SubQuery** is a query within another query. The outer query is called the **main query**, and the inner query is called a **subquery**. SubQuery is always executed first, and the result of subquery is passed on to the main query.
There are two types of subquery – **Correlated** and **Non-Correlated.**
A correlated subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM the list of the main query.
A Non-Correlated subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

Q19. Stored Procedure?

A stored procedure is a subroutine available to applications that access a relational database management system (RDBMS). Such procedures are stored in the

database data dictionary. The sole disadvantage of stored procedure is that it can be executed nowhere except in the database and occupies more memory in the database server. It also provides a sense of security and functionality as users who can't access the data directly can be granted access via stored procedures.

```
DELIMITER $$
CREATE PROCEDURE FetchAllStudents()
BEGIN
SELECT *  FROM myDB.students;
END $$
DELIMITER ;
```

Q20. Advantages and Disadvantages of Stored Procedure?

Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data.

Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

Q21. Recursive stored procedure?

A stored procedure which calls by itself until it reaches some boundary condition. This recursive function or procedure helps programmers to use the same set of code any number of times.

Q22. Trigger?

A DB trigger is a code or programs that automatically execute with response to some event on a table or view in a database. Mainly, trigger helps to maintain the integrity of the database.

Example: When a new student is added to the student database, new records should be created in the related tables like Exam, Score and Attendance tables.

Q23. User defined functions? types of user defined functions?

User defined functions are the functions written to use that logic whenever required. It is not necessary to write the same logic several times. Instead, function can be called or executed whenever needed. Three types of user defined functions:

Scalar Functions.

Inline Table valued functions.

Multi statement valued functions.

Scalar returns unit, variant defined the return clause. Other two types return tables as a return.

Q24. TRUNCATE? DELETE? DROP statements?

DELETE statement is used to delete rows from a table.

TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

DROP command is used to remove an object from the database. If you drop a table, all the rows in the table are deleted and the table structure is removed from the database.

Q25. DROP, TRUNCATE difference?

TRUNCATE removes all the rows from the table, and it cannot be rolled back. **DROP** command removes a table from the database and operation cannot be rolled back.

Q26. DELETE, TRUNCATE difference?

Delete command is used to remove rows from the table, and WHERE clauses can be used for conditional sets of parameters. Commit and Rollback can be performed after delete statement. **Truncate** removes all rows from the table. **Truncate** operation cannot be rolled back. **Truncate** is a DDL statement. **Delete** is a DML statement. **Truncate** does not fire any delete triggers created on the table. Whereas the **Delete** does.

Q27. Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during creation of table or after creation using the ALTER TABLE command. The constraints are:

NOT NULL - Restricts NULL value from being inserted into a column.

CHECK - Verifies that all values in a field satisfy a condition.

DEFAULT - Automatically assigns a default value if no value is specified for the field.

UNIQUE - Ensures unique values to be inserted into the field.

INDEX - Indexes a field providing faster retrieval of records.
PRIMARY KEY - Uniquely identifies each record in a table.
FOREIGN KEY - Ensures referential integrity for a record in another table.

Q28. UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely.
Unlike the primary key, there can be multiple unique constraints defined per table. The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

Q29. Data Integrity?

Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

Q30. CLAUSE?

SQL clause is defined to limit the result set by providing condition to the query. This usually filters some rows from the whole set of records. Example – Query that has WHERE condition. Query that has HAVING condition.

Q31. Aggregate, Scalar functions?

An **aggregate** function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement. Following are the widely used SQL aggregate functions:

AVG() - Calculates the mean of a collection of values.
COUNT() - Counts the total number of records in a specific table or view.
MIN() - Calculates the minimum of a collection of values.
MAX() - Calculates the maximum of a collection of values.
SUM() - Calculates the sum of a collection of values.
FIRST() - Fetches the first element in a collection of values.
LAST() - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT function.

A **scalar** function returns a single value based on the input value. Following are the widely used SQL scalar functions:

LEN() - Calculates the total length of the given field (column).
UCASE() - Converts a collection of string values to uppercase characters.
LCASE() - Converts a collection of string values to lowercase characters.
MID() - Extracts substrings from a collection of string values in a table.
CONCAT() - Concatenates two or more strings.
RAND() - Generates a random collection of numbers of given length.
ROUND() - Calculates the round off integer value for a numeric field (or decimal point values).
NOW() - Returns the current date & time.
FORMAT() - Sets the format to display a collection of values.

Q32. Transaction? properties of the transaction?

A transaction is a logical unit of work performed against a database in which all steps must be performed or none. Properties of the transaction are known as ACID properties, such as:

Atomicity: Ensures the completeness of all transactions performed. Checks whether every transaction is completed successfully if not then transaction is aborted at the failure point and the previous transaction is rolled back to its initial state as changes undone

Consistency: Ensures that all changes made through successful transaction are reflected properly on database

Isolation: Ensures that all transactions are performed independently and changes made by one transaction are not reflected on other

Durability: Ensures that the changes made in the database with committed transactions persist as it is even after a system failure

Q33. Collation? all different types of collation sensitivity?

Collation is defined as a set of rules that determine how character data can be sorted and compared. This can be used to compare A and other language characters and also depends on the width of the characters. ASCII values can be used to compare these character data. Following are different types of collation sensitivity.

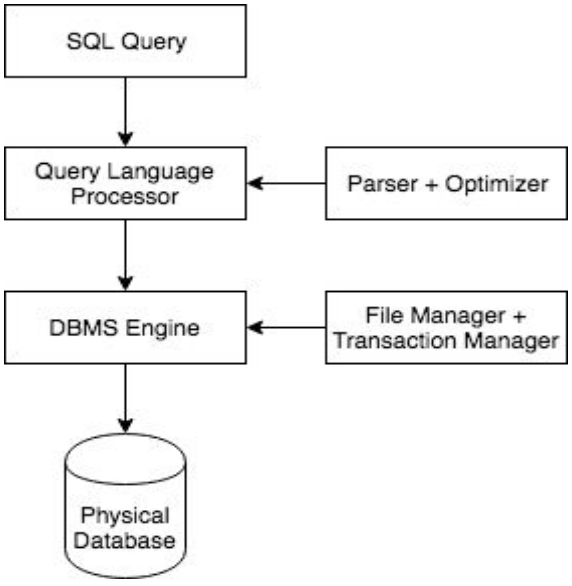
Case Sensitivity – A and a and B and b.

Accent Sensitivity.

Kana Sensitivity – Japanese Kana characters.

Width Sensitivity – Single byte character and double byte character.

Q34. SQL processing



Q35. DDL/DML/DCL/TCL

- **DDL - Data Definition Language**

DDL is the short name of Data Definition Language, which deals with database schemas and descriptions of how the data should reside in the database.

CREATE - create a new table, view for a table or other object in the database

ALTER - modifies an existing database object, such as a table

DROP - deletes an entire table, a view of a table or other objects in 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 is the short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in a database.

SELECT - retrieves records from one or more tables

INSERT - creates a record

UPDATE - modifies records

DELETE - deletes records

MERGE - UPSERT operation (insert or update)

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - interpretation of the data access path

LOCK TABLE - concurrency control

- **DCL - Data Control Language**

DCL is the short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

GRANT (Grant privilege(s) to user)

REVOKE (Remove granted privilege(s) from a user)

- **TCL - Transaction Control Language**

TCL is the short name of Transaction Control Language which deals with a transaction within a database.

COMMIT - commits a transaction

ROLLBACK - rollback a transaction in case of any error occurs

SAVEPOINT - to rollback the transaction making points within groups

SET TRANSACTION - specify characteristics of the transaction

Q36. Having, Where clause?

The Where clause filters rows before grouping. Having clause filters rows after grouping. You cannot use aggregate functions in Where clauses. In Having clause, you can use aggregate functions.

Q37. NULL, Zero, Blank?

A **NULL** value is not at all the same as that of zero or a blank space. NULL value represents a value which is **unavailable**, **unknown**, **assigned** or not applicable whereas a zero is a number and blank space is a character.

Q38. What is the main difference between 'BETWEEN' and 'IN' condition operators?

BETWEEN operator is used to display rows based on a range of values in a row whereas the IN condition operator is used to check for values contained in a specific set of values.
Example of BETWEEN:
SELECT * FROM Students where ROLL_NO BETWEEN 10 AND 50;
Example of IN:
SELECT * FROM students where ROLL_NO IN (8,15,25);

Q39. Alias?

ALIAS command in SQL is the name that can be given to any table or a column. This alias name can be referred in WHERE clause to identify a particular table or a column.

Q40. Pattern matching?

LIKE operator is used for pattern matching, and it can be used as -.
% - Matches zero or more characters.
_(Underscore) – Matching exactly one character.
Example -.

```
Select * from Student where studentname like 'a%'
Select * from Student where studentname like 'ami_'
```

Q41. Auto Increment

Autoincrement keyword allows the user to create a unique number to get generated whenever a new record is inserted into the table. This keyword is usually required whenever PRIMARY KEY in SQL is used.

AUTO INCREMENT keyword can be used in Oracle and **IDENTITY** keyword can be used in SQL SERVER.

SQL Query/Operation Questions

Query 1: 判断一张表的字段数据是否在另外一张表可以用left join 和where (不用in, not in)

```
select Name as Customers from Customers a left join orders b on a.Id = b.CustomerId
where b.Id is null
```

Query 2: 删除表中的重复数据（有ID字段）, 按照字段分组找出需要保留的ID

```
delete from emails where id not in (select min(id) from emails group by email, name)
```

Query 3: 返回表中第二高的薪资（没有则返回null）

```
select
case when max(salary) is not null then  max(salary) else null end  as SecondHighestSalary
from employee        where salary < (select max(salary) from employee)
```

Query 4: 返回表中第n高的薪资(for example 5)

```
select top 1 salary
from (select distinct top 5 salary from worker order by salary desc) a order by salary asc
```

Query 5: 返回表中第n低的薪资(for example 5)

```
select top 1 salary
from (select distinct top 5 salary from worker order by salary asc) a order by salary desc
```

Query 6: 返回表中第n高的薪资(for example 5), 不用top

```
select a.salary from worker a        where 5 = (
        select count(distinct b.salary) from salary b where a.salary <= b.salary)
```

Query 7: 返回表中第n低的薪资(for example 5), 不用top

```
select a.salary from worker a where 5 = (
        select count(distinct b.salary) from salary b where a.salary >= b.salary)
```

Query 8: 返回表中每个部门薪资最高的员工名字

```
with base as (
        select department, max(salary) from worker group by department
)
select * from worker w join base b
where w.department = b.department and w.salary = b.salary
```

Query 9: 返回表中每个部门薪资第二高的员工名字

```
with base as (
```

```
        SELECT DEPARTMENT , max(SALARY ) as SALARY
        FROM Worker group by DEPARTMENT
), secbase as (
        select sectable.department, max(sectable.salary) as salary
        from
        (select w.* from worker w join base b on w.DEPARTMENT  = b.department
        where w.SALARY  < b.salary )  sectable
        group by sectable.department
)
select * from worker w join secbase s on w.DEPARTMENT = s.department and
w.SALARY = s.salary
```

方法2：

```
select * from worker a
where 2 = (
        select count(distinct salary) from worker b
        where a.salary <= b.salary
        and a.department = b.department
)
order by a.department , a.SALARY desc

select b.Name as Department , a.Name as Employee , a.Salary
from Employee a , Department b
where 3 >= (SELECT count(distinct Salary) from Employee b WHERE a.Salary <=
b.Salary and a.DepartmentId = b.DepartmentId )
and a.DepartmentId = b.Id
order by b.Name ,  a.Salary desc
```

Query 10: 返回表中前50%的数据记录

```
with base as (
        SELECT *, row_number()over(order by getdate()) as rownum
        FROM Worker
)
select * from base where rownum <= (select count(rownum)/2 from base )
```

Query 11: 返回表中最后n条数据记录, for example 5

```
with base as (
        SELECT *, row_number()over(order by getdate()) as rownum
        FROM Worker
)
select * from base where rownum > (select max(rownum) - 5 from base )
```

Query 12: Write an SQL query to fetch duplicate records having matching data in some fields of a table.

```
select WORKER_TITLE, AFFECTED_FROM, count(*)
from title
group by WORKER_TITLE, AFFECTED_FROM
having count(*) > 1
```

Query 13: 连接字段函数concat()

```
select concat(FIRST_NAME, ' ', LAST_NAME) as worker_name, salary from worker
```

Query 14: 时间函数year(), month() JOINING_DATE = '2014-02-20 09:00:00'

```
select * from worker where year(JOINING_DATE) = 2014 and
month(JOINING_DATE) = 2
```

Query 15: SQL query转义字符(sql server) ESCAPE （通配符wildcard % _ ）

```
select * from worker where DEPARTMENT like '/_R' ESCAPE '/'
```

Query 16: 字符串函数replace(), len(), ltrim(), rtrim(), charindex(), substring(), upper(), lower()

```
select replace(FIRST_NAME, 'a', 'A') from worker
select len(FIRST_NAME) from worker
select ltrim(FIRST_NAME), rtrim(FIRST_NAME) from worker
select charindex('a', first_name COLLATE Latin1_General_CS_AS) from worker #区分大小写
select charindex('a', first_name) from worker     #不区分大小写
select substring(first_name, 1,3)  from worker    #index从1开始
select upper(first_name), lower(first_name) from worker #全部大写或小写
```

Query 17: 排序函数dense_rank() #序号连续, rank() #序号不连续

```
SELECT
   Score,
   DENSE_RANK() OVER (ORDER BY Score DESC) AS [Rank],
   rank() over (ORDER BY Score DESC) as [RACK2]
FROM Scores
```

Query 18: 创建function的语法：

```
CREATE FUNCTION getNthHighestSalary(@N INT) RETURNS INT AS
BEGIN
   RETURN (
      /* Write your T-SQL query statement below. */
      select distinct b.salary
      from (select salary, dense_rank() over (order by salary desc) as rank from
Employee) b
      where b.rank = @N
   );
END
```

Query 19: 创建存储过程：

```
if (exists (select * from sys.objects where name = 'getBookId'))
   drop proc getBookId
go
create proc getBookId(
   @bookAuth varchar(20),--输入参数,无默认值
   @bookId int output --输入/输出参数 无默认值
)
as
   select @bookId=book_id from books where book_auth=@bookAuth
declare @id int --声明一个变量用来接收执行存储过程后的返回值
exec getBookId '孔子',@id output
select @id as bookId;--as是给返回的列值起一个名字
```

Query 20: with … as 子句计算取消率

```
WITH total_number_of_requests AS (
    SELECT request_at, COUNT(Id) AS 'Requests'
    FROM(
      SELECT DISTINCT Id,Request_at
      FROM       Trips  AS A
      INNER JOIN  Users  AS B ON A.Client_Id = B.Users_Id
      INNER JOIN  Users  AS C ON A.Driver_Id = C.Users_Id
      WHERE B.Banned = 'No' AND C.Banned = 'No'
      AND   A.Request_at BETWEEN '2013-10-01' AND '2013-10-03'
   )Z
   GROUP BY Request_at
), total_number_of_cancels AS (
    SELECT  request_at, COUNT(Cancels) AS 'Cancels'
    FROM(
          SELECT  request_at,Id AS 'Cancels'
          FROM       Trips   AS A
          INNER JOIN  Users   AS B ON A.Client_Id = B.Users_Id
          INNER JOIN  Users   AS C ON A.Driver_Id = C.Users_Id
          WHERE B.Banned = 'No' AND C.Banned = 'No'
          AND   A.Request_at BETWEEN '2013-10-01' AND '2013-10-03'
          AND   A.Status IN ('cancelled_by_driver','cancelled_by_client')
   ) Z
   GROUP BY request_at
)
SELECT  A.request_at AS Day,
       ROUND(CAST(ISNULL(B.Cancels,0.0) AS FLOAT)/A.Requests,2) AS
'Cancellation Rate'
FROM      total_number_of_requests AS A
```

LEFT JOIN total_number_of_cancels AS B ON A.request_at=B.request_at

ROUND函数：按照指定的位数进行四舍五入。round(number, 2) #小数点后两位
CAST函数: 类型转换函数, CAST … AS FLOAT 转成浮点类

Query 21: 连续两天的数据进行比较， DATEADD()

```
select w.id
from weather w
join weather w1 on w.RecordDate = DATEADD(day, 1, w1.RecordDate)
and w.Temperature > w1.Temperature
```

Query 22: 数据大于某值并且ID或其他字段值连续三条或三条以上的数据查询

```
WITH base AS (
   SELECT id,visit_date, people, ROW_NUMBER()OVER(ORDER BY visit_date) AS
'R'
   FROM   stadium  WHERE people >= 100
), sequence AS (
   SELECT A.id       FROM       base AS A
   INNER JOIN  base AS B ON A.id =B.id-1 AND A.R=B.R-1
   INNER JOIN  base AS C ON A.id =C.id-2 AND A.R=C.R-2 )
SELECT A.id,visit_date, people    FROM       stadium  AS A  WHERE A.id IN
(SELECT id FROM sequence)              UNION
SELECT A.id,visit_date, people    FROM       stadium  AS A          WHERE A.id IN
(SELECT id+1 FROM sequence)  UNION
SELECT A.id,visit_date, people    FROM       stadium  AS A          WHERE A.id IN
(SELECT id+2 FROM sequence)
```

Query 23: 表中数据的值进行奇偶行互换方法：

```
select * from (
select
a.id , case when b.student is null then a.student else b.student end as student
from seat a
left join seat b on a.id = b.id - 1
where a.id % 2 <> 0
union
select
a.id , case when b.student is null then a.student else b.student end as student
from seat a
left join seat b on a.id = b.id + 1
where a.id % 2 = 0) A
order by id
```

Query 24: case when … then … when … then … end

```
update salary
```

```
set sex = (case when sex = 'm' then 'f' when sex = 'f' then 'm' end)
```

Query 25: 纵表转成横表的方法（ID对应的每个月份都有数据，转为横表显示）

```
select id, sum(case when month = 'Jan' then revenue else NULL end) as
Jan_Revenue ,
        sum(case when month = 'Feb' then revenue else NULL end) as Feb_Revenue
,
        sum(case when month = 'Mar' then revenue else NULL end) as Mar_Revenue
,
        sum(case when month = 'Apr' then revenue else NULL end) as Apr_Revenue
,
        sum(case when month = 'May' then revenue else NULL end) as
May_Revenue ,
        sum(case when month = 'Jun' then revenue else NULL end) as Jun_Revenue
,
        sum(case when month = 'Jul' then revenue else NULL end) as Jul_Revenue ,
        sum(case when month = 'Aug' then revenue else NULL end) as
Aug_Revenue ,
        sum(case when month = 'Sep' then revenue else NULL end) as
Sep_Revenue ,
        sum(case when month = 'Oct' then revenue else NULL end) as Oct_Revenue
,
        sum(case when month = 'Nov' then revenue else NULL end) as
Nov_Revenue ,
        sum(case when month = 'Dec' then revenue else NULL end) as
Dec_Revenue
from Department
group by id
```

Query 26: 创建空表

```
select * into dupblicate_a from worker where 1=2
```

Query 27: 存储过程例子：

```
IF EXISTS ( SELECT name FROM sysobjects WHERE type = 'P' AND uid =
user_id()
            AND name = 'mkoid' )
      DROP PROC mkoid
go

CREATE PROC mkoid (
        @oid  id_TY OUTPUT
)
AS BEGIN
        DECLARE @prefix  binary(3), @chkfix tinyint
```

```
        DECLARE    @seqnum     binary(4), @chksum tinyint

        SELECT     @oid = NULL  -- in case it breaks!

        SET transaction isolation level 1
        SELECT     @prefix = prefix, @chkfix = chksum
        FROMmkoid_prefix
        IF ( @@error != 0 ) RETURN 1

        WHILE @chksum IS NULL OR @chksum = 0
        BEGIN
              INSERT       mkoid_seqnum VALUES ()
              IF ( @@error != 0 ) RETURN 2

              SELECT  @seqnum = convert(binary(4), convert(int,@@identity))

              SELECT     @chksum = @chkfix ^ substring(@seqnum, 1, 1)
              SELECT     @chksum = @chksum ^ substring(@seqnum, 2, 1)
              SELECT     @chksum = @chksum ^ substring(@seqnum, 3, 1)
              SELECT     @chksum = @chksum ^ substring(@seqnum, 4, 1)
        END

        SELECT     @oid = @prefix + @seqnum + convert(binary(1), @chksum)

        RETURN     0
END
go
```

How to delete duplicate records in a table?

```
DELETE FROM <table name>
WHERE <primary key columns> NOT IN
 (SELECT MAX(rowid) FROM <table name>
Group by <primary key columns>
```

How can you create an empty table from an existing table?

```
Select * into studentcopy from student where 1=2
```

How to fetch the first 5 characters of the string?

```
Select SUBSTRING(StudentName,1,5) as studentname from student
Select LEFT(Studentname,5) as studentname from student
```