SQL Questions &Terms

Q. What is the difference between inner join, outer join, and full join?
Inner Join: Compares all the records of tables being joined and matches rows based on common fields and returns the matching rows.
Outer Join:

Left Outer Join: Returns all of the records of the left of the two tables along with the matching records in the second (right) table.
Right Outer Join: Returns all of the records of the right of the two tables along with the matching records in the first (left) table.

Full Join: A full join is an outer join that takes *all* data from both tables, matched where it can, as opposed to LEFT or RIGHT join that takes all the data from a table, *and* any matching records from the other table.

Example:

CUSTOMER table contains details of customers.
DISCOUNTDEAL table contains details of different discount deals.

You want to see all customers, and list the discount deal they have if they have one:

```
SELECT *
FROM CUSTOMER
LEFT JOIN DISCOUNTDEAL ON
CUSTOMER.DEALID=DISCOUNTDEAL.DEALID
```

You want to see all deals, and list the customers that have that deal if there are any:

```
SELECT *
FROM CUSTOMER
RIGHT JOIN DISCOUNTDEAL ON
CUSTOMER.DEALID=DISCOUNTDEAL.DEALID
```

(you could also do a LEFT JOIN from DISCOUNTDEAL to CUSTOMER to achieve the same thing, of course)

You want to see all deals (even it no customers are using them), and all customers (even if they don't have a deal)

```
SELECT *
FROM CUSTOMER
FULL OUTER JOIN DISCOUNTDEAL ON
CUSTOMER.DEALID=DISCOUNTDEAL.DEALID
```

Q. What is a cursor?
Cursor is current set of records. We have to write program unit 2, in which cursor execution takes place, and execute it manually.

Q. What is a trigger?
Trigger is stored in the database and gets fired automatically when any database operation takes place like insert, update, or delete. Trigger executes itself.

Q. What do transactions do?
Transactions help you make sure your data is consistent. They do this by allowing you to group command that are being sent to the database. If any command within the group fails, they are all "rolled back", meaning that the database will act like none of the commands ever happened. This could be useful if, for instance, you were updating a bank account transfer record. If a debit from one account went through, but the credit to another account failed (for any reason… power outage, bad syntax, whatever), your data consistency (and your balance sheet) would be hosed. If both commands were grouped into a transaction, then they would both be rolled back if one of them failed, maintaining data consistency.

Q. What makes a database “relational”?
In a **relational database** the data is arranged in relations which are nothing but conventional tables with some set of rules governing the arrangement of data within the table and among the tables. Any record or any row in a table is called an entity. And any value (or cell) in the row is called is called attribute.

Now the set of rules (12 rules) governing the relational database
are called Codd's rules (proposed by Dr. E F Codd in 1970.

To put things simple,
a) An entity must have a unique set of attributes, that is, every row in a table must be unique.
b) The referential integrity between two relations must be preserved. That is, if a record in a table A depends on a record in table B, then the database must detect and cancel any operation in table B that disturbs this integrity.

e) Domain integrity must be preserved. That is, if there is a restriction for an attribute set (column) that its value should be within a predefined range then this restriction must apply in every insert/modification.

f) Any value which is not supplied or unknown should be termed as null. A null value is not equal to empty string ("") or zero. A null is even
not equal to a null.

g) Access to the database should be easy, but the ability of the user to view or modify the data should depend on permissions. The part of the database for which the user has no kind of permission is invisible to the user.

Q. What is “**ACID**” compliance?
Any database transaction which is:

- A – Atomic
- C – Consistent
- I – Isolated
- D – Durable

is **ACID** compliant.

Q. Query to **extract** only **duplicate records** from table?
The trick is to use keywords GROUP and HAVING. The following query will **extract duplicate records** from a specific column of a particular table.

```
Select specificColumn
FROM particluarTable
GROUP BY specificColumn
HAVING COUNT(*) > 1
```

This will list all the records that are repeated in the column specified by “specificColumn” of a “particlarTable”.

Q. What are **Stored Procedures**?
Stored Procedures are precompiled SQL routines that are stored on the database server.

Benefits: Precompiled execution, Reduced client/server traffic, Efficient reuse of code and programming abstraction, and Enhanced security controls.

Characteristics:
-Accept data in the form of input parameters.
-Result is returned through use of recordset, output parameters, and a return code.

Example

Table: **Inventory**

ID	Product	Warehouse	Quantity
142	Green Beans	NY	100
214	Peas	FL	200
825	Corn	NY	140
512	Lima Benas	NY	180
491	Corn	FL	80
379	Watermelon	FL	85

Sample query:

```
SELECT Product, Quantity
FROM Inventory
WHERE Warehouse = 'FL'
```

Disadvantage: DB Server recompiles the query and executes it from scratch every time.

Stored Procedure:

```
CREATE PROCEDURE sp_GetInventory
@location varchar(10)
AS
SELECT Product, Quantity
FROM Inventory
WHERE Warehouse = @location
```

Now we can get the same results as before but using a much cleaner syntax and faster by running the following statement:

```
EXECUTE sp_GetInventory 'FL'

EXECUTE sp_GetInventory 'NY'
```