Do you know the basic memory structures associated with any Oracle Database. (W.r.t 10g / 11g / 12c)?

The basic memory structures associated with Oracle Database include:

- **System Global Area (SGA)**

  The SGA is a group of shared memory structures, known as *SGA components*, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

- **Program Global Area (PGA)**

  A PGA is a memory region that contains data and control information for a server process. It is nonshared memory created by Oracle Database when a server process is started. Access to the PGA is exclusive to the server process. There is one PGA for each server process. Background processes also allocate their own PGAs. The total PGA memory allocated for all background and server processes attached to an Oracle Database instance is referred to as the **total instance PGA memory**, and the collection of all individual PGAs is referred to as the **total instance PGA**, or just **instance PGA**.

Do you know what read-consistency mechanism in Oracle 12c is?

When DML operations are performed in a transaction, the changes are visible only to the session performing the DML operations. The changes are visible to other users in the database only when a COMMIT is issued (or a DDL statement causes an implicit commit).

All data changes made in a transaction are temporary until the transaction is committed. Oracle Database 12*c* has a read-consistency mechanism to ensure that each user sees the data as it existed at the last commit.

When DML operations are performed on existing rows (through UPDATE, DELETE, or MERGE operations), the affected rows are locked by Oracle; therefore, no other user can perform a DML operation on those rows. The rows updated or deleted by a transaction can be queried by another session.

When changes are committed, they are made permanent to the database. All locks on the affected rows are released, and all savepoints are removed. The previous state of the data is lost (the undo segments may be overwritten). All users can view the changed data. When changes are rolled back, data changes are undone and the previous state of data is restored. All locks on the affected rows are released.

Oracle uses *read consistency* to make sure you do not see the changes made to data after your query is started. Also, Oracle uses a locking mechanism to make sure that two different user sessions can't modify data in the same row at the same time.

How many levels of consistency exist in Oracle?

2 levels: statement-level consistency and transaction-level consistency.
Oracle always uses statement-level consistency, which ensures that the data visible to a statement does not change during the life of that statement. Transactions can consist of one or more statements. When used, transaction-level consistency will ensure that the data visible to all statements in a transaction does not change for the life of the transaction.

How does Oracle implement consistency?

Oracle implements consistency internally through the use of ***system change numbers (SCNs)***. An *SCN* is a time-oriented, database-internal key. The SCN only increases, never decreases, and represents a point in time for comparison purposes.

The banking example will help clarify.
Matt starts running a total-balance report against the checking account table at 10:00 a.m.; this report takes five minutes. During those five minutes, the data he is reporting on changes when Sara transfers $5,000 from her checking account to her brokerage account. When Matt's session gets to Sara's checking-account record, it will need to reconstruct what the record looked like at 10:00 a.m. Matt's session will examine the *undo segment* that Sara used during her account-transfer transaction and will re-create the image of what the checking-account table looked like at 10:00 a.m.

Next, at 10:05 a.m., Matt runs a total balance report on the cash in the brokerage account table. If he is using transaction-level consistency, his session will re-create what the brokerage account table looked like at 10:00 a.m. (and exclude Sara's transfer). If Matt's session is using the default statement-level consistency, his session will report on what the brokerage account table looked like at 10:05 a.m. (and include Sara's transfer).

Oracle never uses locks for reading operations, because reading operations will never block writing operations. Instead, the *undo segments* (also known as *rollback segments*) are used to re-create the image needed. Undo segments are released for reuse when the transaction writing to them commits or if undo_management is set to auto and the undo_retention period is exceeded, so sometimes a consistent image cannot be re-created. When this happens, Oracle raises a "snapshot too old" exception. Using this example, if Matt's transaction can't locate Sara's transaction in the rollback segments because it was overwritten, Matt's transaction will not be able to re-create the 10:00 a.m. image of the table and will fail.

So, in the above example, Oracle internally assigns Matt's first statement the current SCN when it starts reading the checking-account table. This starting SCN is compared to each data block's SCN. If the data-block SCN is higher (newer), the rollback segments are examined to find the older version of the data.

How many types of types of locks are you aware of? Where and why are they used?

Locks are implemented by Oracle Database 12*c* to prevent destructive interaction between concurrent transactions. Locks are acquired automatically by Oracle when a DML statement is executed; no user intervention or action is needed. Oracle uses the lowest level of restrictiveness when locking data for DML statements—only the rows affected by the DML operation are locked.

Locks are held for the duration of the transaction. A commit or rollback will release all the locks. **There are two types of locks: explicit and implicit**. The locks acquired by Oracle automatically when DML operations are performed are called *implicit locks*. There is no implicit lock for SELECT statements.

If the user locks data manually, it is called *explicit locking*. The **LOCK TABLE** statement and **SELECT…FOR UPDATE** statements are used for explicitly locking the data.

The SELECT…FOR UPDATE statement is used to lock specific rows, preventing other sessions from changing or deleting those locked rows. When the rows are locked, other sessions can select these rows, but they cannot change or lock these rows. The syntax for this statement is identical to a SELECT statement, except you append the keywords FOR UPDATE to the statement.

The locks acquired for a SELECT FOR UPDATE will not be released until the transaction ends with a COMMIT or ROLLBACK, even if no data changes.

```
SELECT product_id, warehouse_id, quantity_on_hand
FROM oe.inventories
WHERE quantity_on_hand < 5
FOR UPDATE;
```

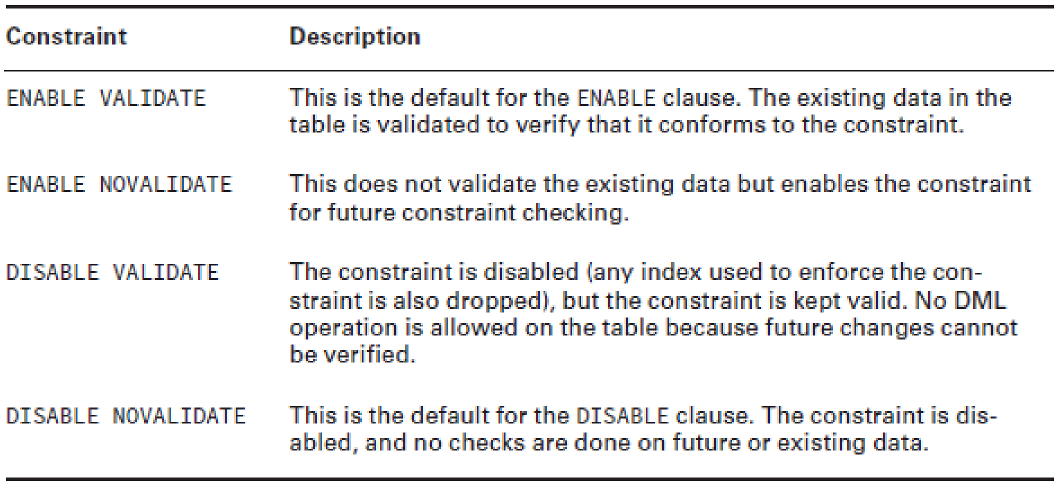
The LOCK statement is used to lock an entire table, preventing other sessions from performing most or all DML on it. Locking can be in either shared or exclusive mode. Shared mode prevents other sessions from acquiring an exclusive lock but allows other sessions to acquire a shared lock. Exclusive mode prevents other sessions from acquiring either a shared lock or an exclusive lock. The following is an example of using the LOCK statement:

```
LOCK TABLE inventories IN EXCLUSIVE MODE;
```

What do you understand by this statement:

ALTER TABLE wh01 MODIFY CONSTRAINT pk_wh01 DISABLE NOVALIDATE;

ENABLE and DISABLE affect only future data that will be added or modified in the table. In contrast, the VALIDATE and NOVALIDATE keywords in the ALTER TABLE statement act on the existing data. Therefore, a constraint can have four states,



Constraint	Description
`ENABLE VALIDATE`	This is the default for the `ENABLE` clause. The existing data in the table is validated to verify that it conforms to the constraint.
`ENABLE NOVALIDATE`	This does not validate the existing data but enables the constraint for future constraint checking.
`DISABLE VALIDATE`	The constraint is disabled (any index used to enforce the constraint is also dropped), but the constraint is kept valid. No DML operation is allowed on the table because future changes cannot be verified.
`DISABLE NOVALIDATE`	This is the default for the `DISABLE` clause. The constraint is disabled, and no checks are done on future or existing data.

By default Oracle checks whether the data conforms to the constraint when the statement is executed. Can you change this behaviour?

By default, Oracle checks whether the data conforms to the constraint when the statement is executed. Oracle allows you to change this behavior if the constraint is created using the DEFERRABLE clause (NOT DEFERRABLE is the default). It specifies that the transaction can set the constraint-checking behavior.

INITIALLY IMMEDIATE specifies that the constraint should be checked for conformance at the end of each SQL statement (this is the default). INITIALLY DEFERRED specifies that the constraint should be checked for conformance at the end of the transaction.

The DEFERRABLE status of a constraint cannot be changed using ALTER TABLE MODIFY CONSTRAINT; you must drop and re-create the constraint. You can change the INITIALLY {DEFERRED|IMMEDIATE} clause using ALTER TABLE.

If the constraint is DEFERRABLE, you can set the behavior by using the SET CONSTRAINTS command or by using the ALTER SESSION SET CONSTRAINT command. You can enable or disable deferred constraint checking by listing all the constraints or by specifying the ALL keyword. The SET CONSTRAINTS command is used to set the constraint-checking behavior for the current transaction, and the ALTER SESSION command is used to set the constraintchecking
behavior for the current session.

As an example, let's create a primary key constraint on the CUSTOMER table and a foreign key constraint on the ORDERS table as DEFERRABLE. Although the constraints are created as DEFERRABLE, they are not deferred because of the INITIALLY IMMEDIATE clause.

```
ALTER TABLE customer ADD CONSTRAINT pk_cust_id
PRIMARY KEY (cust_id) DEFERRABLE
INITIALLY IMMEDIATE;
```

```
ALTER TABLE orders ADD CONSTRAINT fk_cust_id
FOREIGN KEY (cust_id)
REFERENCES customer (cust_id)
ON DELETE CASCADE DEFERRABLE;
```

If you try to add a row to the ORDERS table with a CUST_ID value that is not available in the CUSTOMER table, Oracle returns an error immediately, even though you plan to add the CUSTOMER row soon. Since the constraints are verified for conformance as each SQL statement is executed, you must insert the row in the CUSTOMER table first and then add it to the ORDERS table. Because the constraints are defined as DEFERRABLE, you can change this behavior by using this command:

```
SET CONSTRAINTS ALL DEFERRED;
```

Now you can insert rows to these tables in any order. Oracle checks the constraint conformance only at commit time.

If you want deferred constraint checking as the default, create or modify the constraint by using INITIALLY DEFERRED, as in this example:

```
ALTER TABLE customer MODIFY CONSTRAINT pk_cust_id
INITIALLY DEFERRED;
```

(One of our students, you got selected at BNP Paribas (an MNC bank) was posed the below Question in the Technical Interview)

You have been provided the following information to create tables and constraints for an application developed in your company to maintain geographic information:

■■ The COUNTRY table stores the country name and country code. The country code uniquely identifies each country. The country name must be present.

■■ The STATE table stores the state code, name, and its capital. The country code in this table refers to a valid entry in the COUNTRY table. The state name must be present. The state code and country code together uniquely identify each state.

■■ The CITY table stores the city code, name, and population. The city code uniquely identifies each city. The state and country where the city belongs are also stored in the table, which refers to the STATE table. The city name must be present.

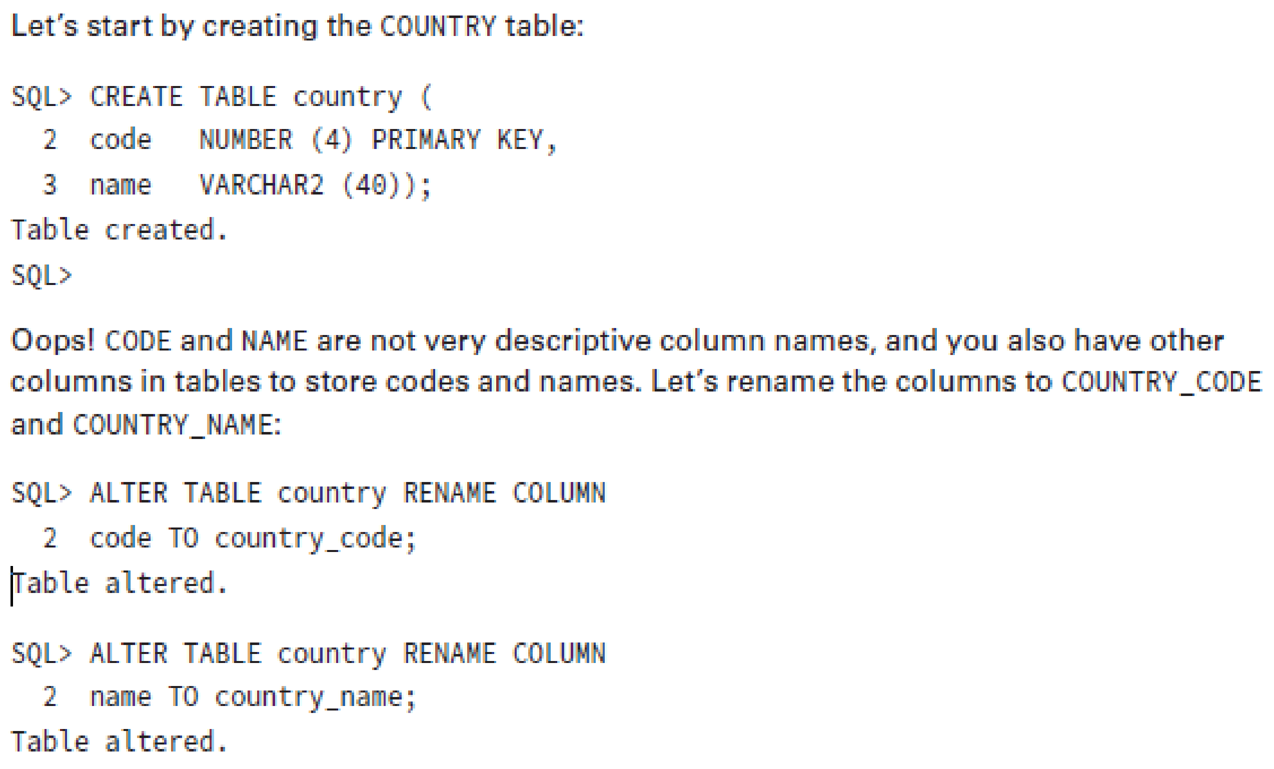
■■ Each table should have a column identifying the created-on timestamp, with the system date as the default.

■■ The user should not be able to delete from the COUNTRY table if there are records in the STATE table for that country.

■■ The records in the CITY table should be automatically removed when their corresponding state is removed from the STATE table.

■■ All foreign and primary key constraints should be provided with meaningful names.

// The problem is solved from a student (Fresher) point of View , *Not a database Expert :*



Let's start by creating the COUNTRY table:

```
SQL> CREATE TABLE country (
  2  code   NUMBER (4) PRIMARY KEY,
  3  name   VARCHAR2 (40));
Table created.
SQL>
```

Oops! CODE and NAME are not very descriptive column names, and you also have other columns in tables to store codes and names. Let's rename the columns to COUNTRY_CODE and COUNTRY_NAME:

```
SQL> ALTER TABLE country RENAME COLUMN
  2  code TO country_code;
Table altered.

SQL> ALTER TABLE country RENAME COLUMN
  2  name TO country_name;
Table altered.
```

```
SQL>
```

You also forgot to provide a name for the primary key constraint. Because the table was created with a system-generated name, you have to find the name first to rename the constraint:

```
SQL> SELECT constraint_name, constraint_type
  2  FROM user_constraints
  3  WHERE table_name = 'COUNTRY';

CONSTRAINT_NAME                C
------------------------------ -
SYS_C0010893                   P


SQL> ALTER TABLE country RENAME CONSTRAINT SYS_C0010893 TO pk_country;
Table altered.

SQL>
```

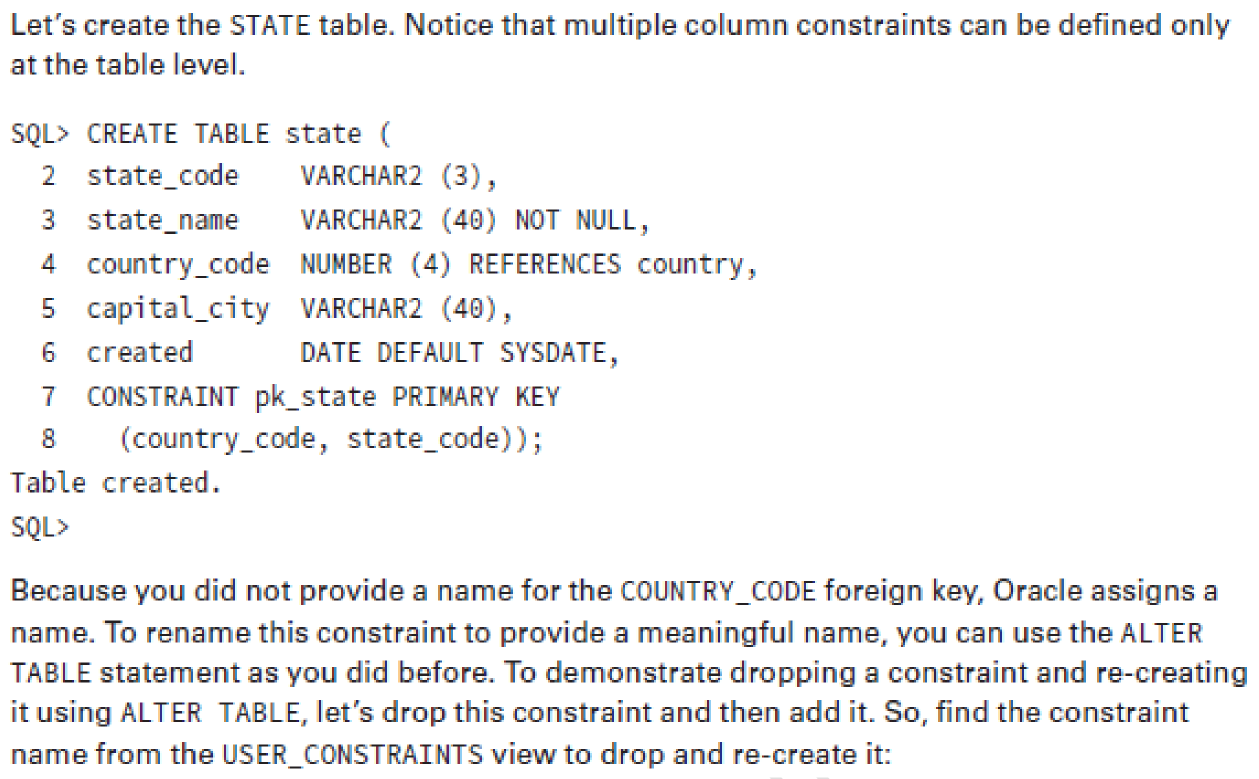
Oops, again! The table should include a column to store the created-on date, and the country name cannot be `NULL`.

Before you continue, realize that if you have a good logical and physical design before you start creating tables, you will not have any of these problems. This is not the typical or recommended approach to creating tables for the application. The objective here is to demonstrate the various options available.

```
SQL> ALTER TABLE country MODIFY country_name NOT NULL
  2  ADD created  DATE DEFAULT SYSDATE;
Table altered.
SQL>
```

Review the table created:

```
SQL> DESCRIBE country
 Name                  Null?    Type
 ------------------ -------- ------------
 COUNTRY_CODE          NOT NULL NUMBER(4)
 COUNTRY_NAME          NOT NULL VARCHAR2(40)
 CREATED                        DATE
SQL>
```


Let's create the STATE table. Notice that multiple column constraints can be defined only at the table level.

```
SQL> CREATE TABLE state (
  2  state_code    VARCHAR2 (3),
  3  state_name    VARCHAR2 (40) NOT NULL,
  4  country_code  NUMBER (4) REFERENCES country,
  5  capital_city  VARCHAR2 (40),
  6  created       DATE DEFAULT SYSDATE,
  7  CONSTRAINT pk_state PRIMARY KEY
  8    (country_code, state_code));
Table created.
SQL>
```

Because you did not provide a name for the COUNTRY_CODE foreign key, Oracle assigns a name. To rename this constraint to provide a meaningful name, you can use the ALTER TABLE statement as you did before. To demonstrate dropping a constraint and re-creating it using ALTER TABLE, let's drop this constraint and then add it. So, find the constraint name from the USER_CONSTRAINTS view to drop and re-create it:

```
SQL> SELECT constraint_name, constraint_type  FROM user_constraints WHERE table_name = 'STATE';

CONSTRAINT_NAME   Type
------------------------------ -
SYS_C002811       C
PK_STATE          P
SYS_C002813       R

SQL> ALTER TABLE state DROP CONSTRAINT SYS_C002813;
```
Table altered.

```
SQL> ALTER TABLE state ADD CONSTRAINT fk_state FOREIGN KEY (country_code) REFERENCES country;
```
Table altered.

Now you'll create the CITY table. Notice the foreign key constraint is created with the ON DELETE CASCADE clause:

```
SQL> CREATE TABLE city (
2 city_code VARCHAR2 (6),
3 city_name VARCHAR2 (40) NOT NULL,
4 country_code NUMBER (4) NOT NULL,
5 state_code VARCHAR2 (3) NOT NULL,
6 population NUMBER (15),
7 created DATE DEFAULT SYSDATE,
8 constraint pk_city PRIMARY KEY (city_code),
9 constraint fk_city FOREIGN KEY
10 (country_code, state_code)
11 REFERENCES state ON DELETE CASCADE);
Table created.
```