DBMS_MID-II_SHORT ANSWERS

1) What is the difference between nested query and correlated sub query. 2 M

Ans:

Nested Query	Correlated Subquery
Nested Query is Independent of outer query	Correlated Subquery Depends on outer query
Nested Query is Executed once	Nested Query is Executed repeatedly (for each outer row)
Nested Query is faster than Correlated Subquery	Slower when compared to Nested Query
Used when the result can be calculated before the outer query.	Used when the result changes for each row in the outer query.

2) What is functional dependency? Write its types. 2 M

Ans:

a functional dependency (FD) shows the relationship between two sets of attributes in a table.

It means: One attribute depends on another attribute.

If A → B, it means A functionally determines B

1. Trivial Functional Dependency	When the dependent attribute is part of the determinant itself.	{RollNo, Name} → Name
2. Non-Trivial Functional Dependency	When the dependent attribute is **not part** of the determinant.	RollNo → Name
3. Transitive Functional Dependency	When one attribute depends on another **through a third attribute**.	RollNo → DeptId and DeptId → DeptName ⇒ RollNo → DeptName

3) What property of decomposition is guaranteed by both BCNF and 3NF? 2 M

Ans:

The property of **lossless join** is guaranteed by both **BCNF** and **3NF** decompositions.

When a relation is decomposed into smaller relations in BCNF or 3NF, the **lossless join property** ensures that **no information is lost**.

The original relation can be reconstructed by joining the decomposed relations.

4) Why we need Triggers in a database? 2 M

Ans:

Triggers are used in a database to **automatically execute actions** in response to specific events (like INSERT, UPDATE, or DELETE) on a table.

Triggers help maintain data integrity, follow rules, and record changes automatically without manual intervention.

5) What is the Having clause in SQL query? 2 M

Ans:

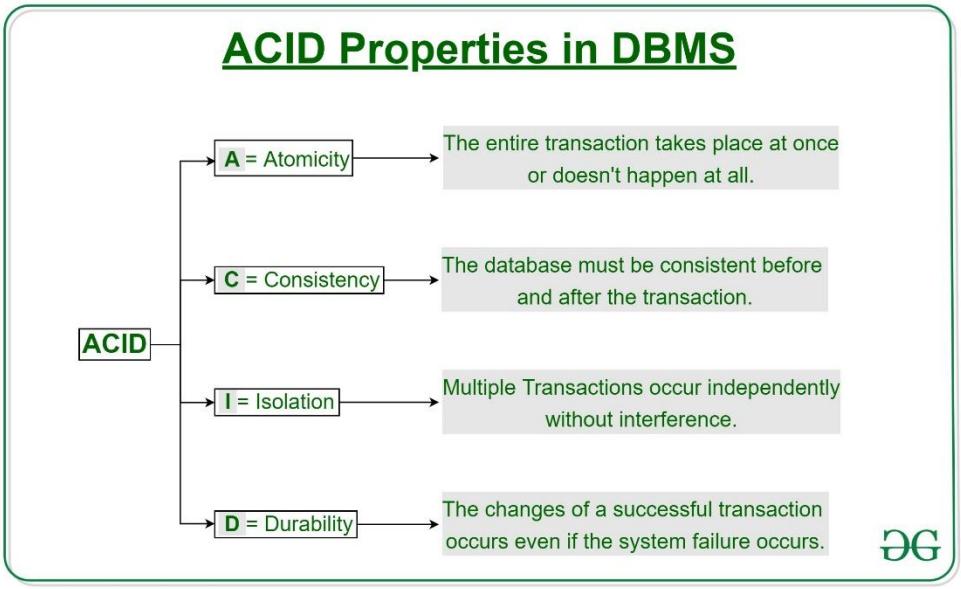
The **HAVING** clause in SQL is used to **filter the results of groups** created by the GROUP BY clause based on a **condition**.

```
SELECT dept, COUNT(*)
FROM employees
GROUP BY dept
HAVING COUNT(*) > 5;
```

The above query shows only departments having more than 5 employees

6) Define ACID properties of a transaction. 2 M

Ans:



7) What is the main purpose of finding the closure of a functional dependency? 2 M

Ans:

1) The main purpose of finding the closure of a functional dependency is to determine all attributes that can be functionally derived from a given set of attributes.
2) It helps in finding candidate keys, testing normalization, and checking equivalence of functional dependencies.

8) Write a SQL query on EXCEPT operation. 2 M

Ans:

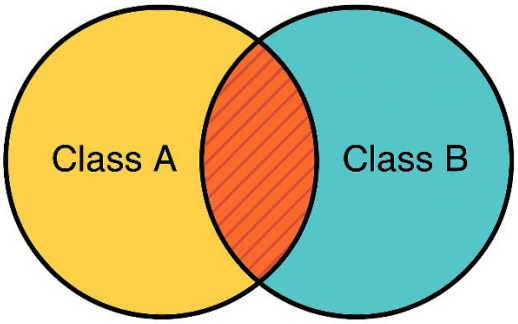
The equivalent keyword to EXCEPT is MINUS in Oracle. The **EXCEPT** (MINUS in Oracle) operation in SQL is a SET Operation that returns the rows from the **first query** that are **not present in the second query**.

```
SELECT student_id FROM ClassA

EXCEPT or MINUS

SELECT student_id FROM ClassB;
```

This query returns the **student IDs that are in ClassA but not in ClassB**.(Yellow Portion)



9) Write the differences between nested and correlated query. 2 M

Ans:

Nested Query	Correlated Subquery
Nested Query is Independent of outer query	Correlated Subquery Depends on outer query
Nested Query is Executed once	Nested Query is Executed repeatedly (for each outer row)
Nested Query is faster than Correlated Subquery	Slower when compared to Nested Query
Used when the result can be calculated before the outer query.	Used when the result changes for each row in the outer query.

10) What are Concurrency protocols. 2 M

Ans:

Concurrency protocols are rules or methods used in a database to **manage simultaneous transactions** so that data remains **consistent and accurate**.

Example:
Common concurrency control protocols include **Two-Phase Locking (2PL)** and **Timestamp Ordering**.

Two-Phase Locking (2PL): Ensures serializability by dividing a transaction into two phases — **growing phase** (acquires locks) and **shrinking phase** (releases locks).

Timestamp Ordering: Uses **timestamps** to order transactions so they execute in the same order as their timestamps, avoiding conflicts.

11) List out various data structures used for indexing purposes in database. 2 M

Ans:

B-Tree: Balanced tree structure that allows fast searching, insertion, and deletion.

B+ Tree: Extension of B-Tree where all data is stored at leaf nodes for efficient range queries.

Hash Index: Uses hash functions to quickly locate records based on key values.

Bitmap Index: Uses bit arrays to represent column values, useful for columns with few distinct values.

AVL Tree: A self-balancing binary search tree that keeps height differences minimal for faster lookups.

2-3-4 Tree: A balanced search tree where each node can have 2, 3, or 4 children, ensuring efficient search and insertion.

12) Is secondary index a dense index or sparse index? Justify your answer. 2 M

Ans:

A **secondary index** is always a **dense index**.

Justification:
In a secondary index, the data file is **not ordered** on the indexing attribute, so an **index entry must exist for every record** to locate it efficiently.

Hence, it is **dense** — containing one index entry for each record in the data file.

13) Is primary index a dense index or sparse index? Justify your answer. 2 M

Ans:

A **primary index** is usually a **sparse index**.

Justification:
In a primary index, the data file is **ordered on the primary key**, so it is enough to have **one index entry per block (or group) of records** rather than for every record. Hence, it is **sparse**, not dense.

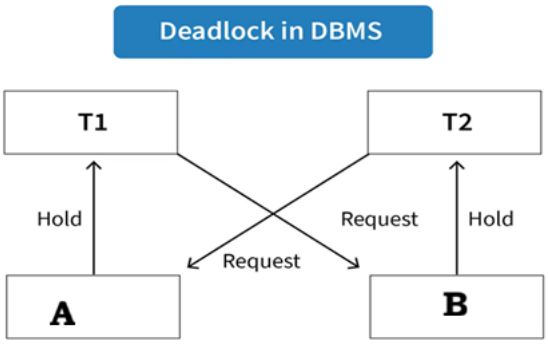
14) Define deadlock. 2 M

Ans:

A **deadlock** in a transaction occurs when **two or more transactions wait indefinitely** for each other to release locks on resources.

Example:

- **T1** locks **Item A** and waits for **Item B**.
- **T2** locks **Item B** and waits for **Item A**.
  Both transactions keep waiting — this situation is called a **deadlock**.



15) What is hash-based indexing in DBMS? 2 M

Ans:

Hash-based indexing uses a **hash function** to convert a search key into a **hash value (address)** where the corresponding record is stored.

It allows **fast data retrieval** for equality searches (e.g., = operator).

Example:
If a hash function is h(x) = x % 10, and we want to store a record with key **123**,
then h(123) = 123 % 10 = 3.
So, the record is stored in **bucket 3** of the hash table.

This method provides **direct access** to data using the key value.

*****