DATABASES SDE SHEET (RIDDHI DUTTA)

Disclaimer : Use it only for your interviews. Study the subject well otherwise. Credits : My Notes , Geeks for Geeks , TutorialsPoint, Javatpoint.

Connect with me on Linkedin.
Connect with me on Instagram.
Subscribe to my Youtube Channel
For more such technical content.

Basics OF DBMS , RELATIONSHIPS , KEYS.

Q1. What is DBMS?

Collection of interrelated data. Helps in efficient retrieval , eg insertion and deletion of data.

Q2. Why DBMS over File Processing System?

Redundancy of Data
Inconsistency of Data
Data Mapping and Access(Independent files for independent tables. Very difficult to join them)
Unauthorised access or security(To access only a particular students data in a students file)
No concurrent access
No backup and recovery

Q3 Explain levels of Data Abstraction.

Physical Level - The lowest level of abstraction that describes how the data is actually stored. This describes low level complex data structure in detail. Logical Level - The next higher level of abstraction that describes what data are stored in the database and what relationships exist among those data. View Level - The highest level of abstraction that describes only part of the entire database.

Q4. What are Instances and Schemas?

Overall design of a database is a **schema**.
The collection of information stored in the database at a particular moment is called the **instance** of the database.

Q5. Explain Data Models.

Data models define how the logical structure of a database is modeled.

Entity-Relationship Model

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on −

- Entities and their *attributes.*
- Relationships among entities.

An entity is a thing or object in a real world which is distinguishable from all other objects.
An entity is described by set of the set of **attributes.**

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an n ary relation.

Q6. Explain Database Constraints for SQL.

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL.

- NOT NULL Constraint − Ensures that a column cannot have NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX − Used to create and retrieve data from the database very quickly.

Also there are domain constraints , which specifies the type and range of values of each attributes.

Q7. Explain Database Languages.

DML -> Data Manipulation Language (Insert , update , delete , select)

DML are of two types i.e Procedural DML(Require a user to specify **how** to get the needed data) and Declarative DML(Require a user to specify **what** data are needed.)

- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

DQL -> Data Query Language

DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

- **SELECT** – is used to retrieve data from the a database.

DDL -> Data Definiton Language

Specifies a database schema by a set of definitions. A part of the DDL specifies various constraints in a database.

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary. •
  RENAME –is used to rename an object existing in the database.

DCL -> Data Control Language

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

TCL -> Transaction Control Language

TCL commands deals with the transaction within the database.

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs. •

SAVEPOINT–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

Q8. **Explain role of an Database Administrator.**

DBA is the person who has the central control of both the data and programs of the DBMS. The functions of DBA are

Schema Definition

Storage Structure and Access Method

Schema and Physical Organisation Modification

Granting of the organisation or data.

Routine Maintenance.

Q9. Explain types of attributes.

An attribute is a property or characteristic of an entity. An entity may contain any number of attributes.

1. Simple Attribute - It is indivisible. Eg phone number. Cannot be divided into more attributes.
2. Composite Attribute -> It can be divided into several composite or simple attributes
3. Single Valued Attribute -> An attribute that can have only one value for an entity. Eg name
4. Multi Valued Attribute -> An attribute that can have more than one value for an entity. Eg phone number
5. Derived Attribute -> The value of the attribute can be derived from 1 or more attributes of the entity. For eg age which can be calculated from date_of_birth attribute.

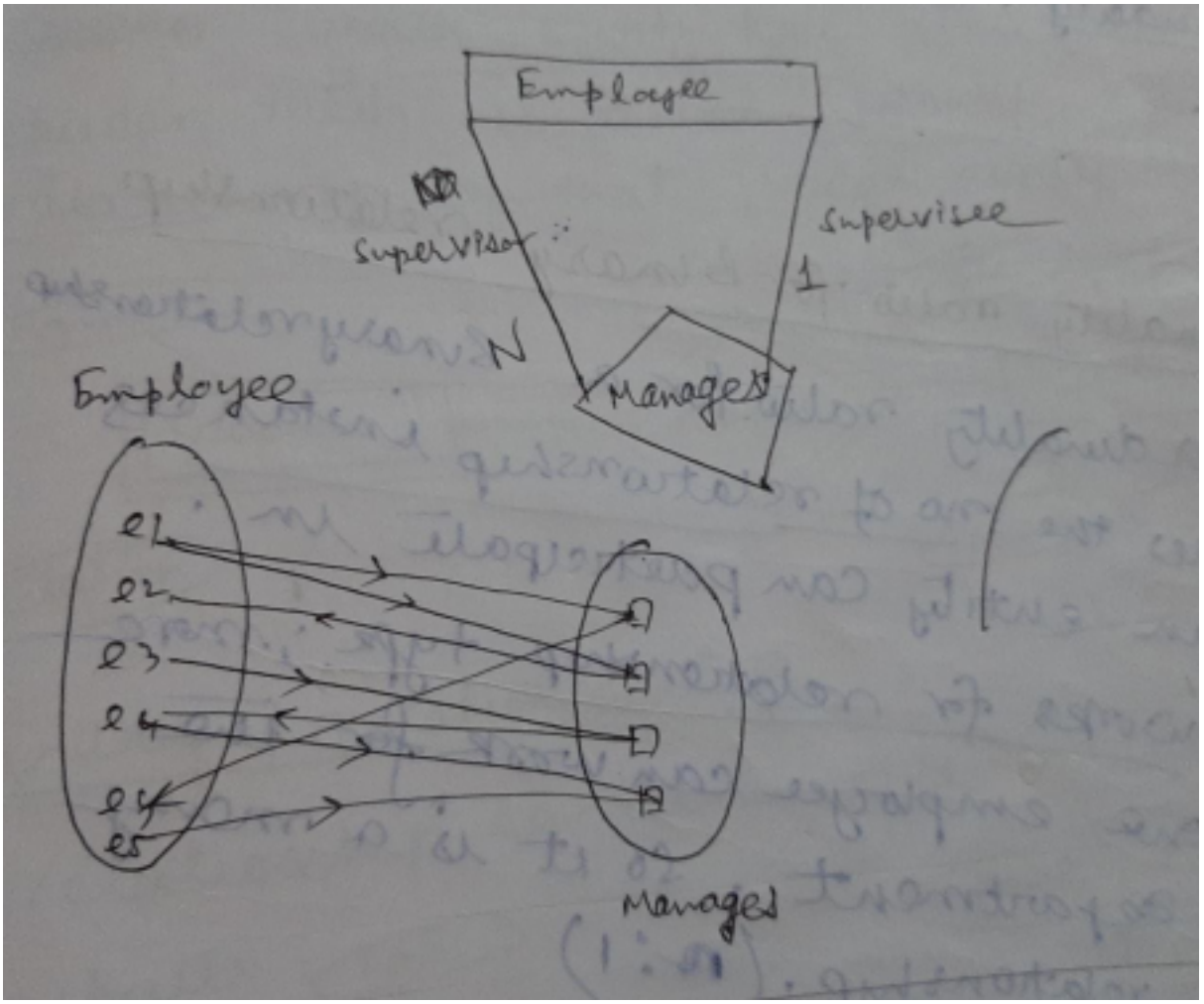
Null Attribute -> Null is a special attribute which means not known or not available.

Q10. What is a RelationShip Degree ?

It is no of participating entity types.

Q11. What is a Recursive Relationship ?

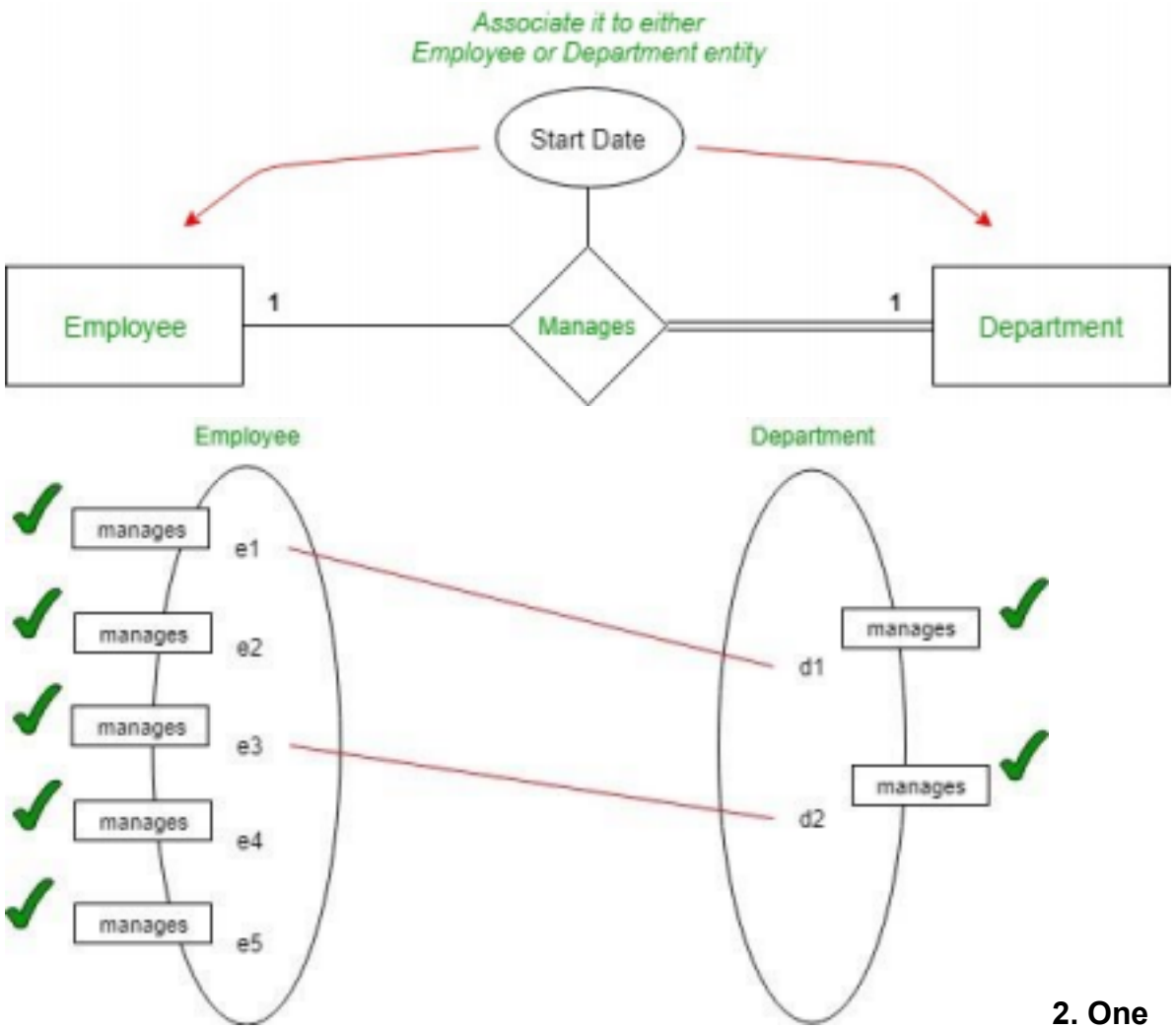
A relationship that associates an entity of a particular entity set with another entity of the same entity set.



Q12. Explain Cardinality Ratio for Binary Relationship.

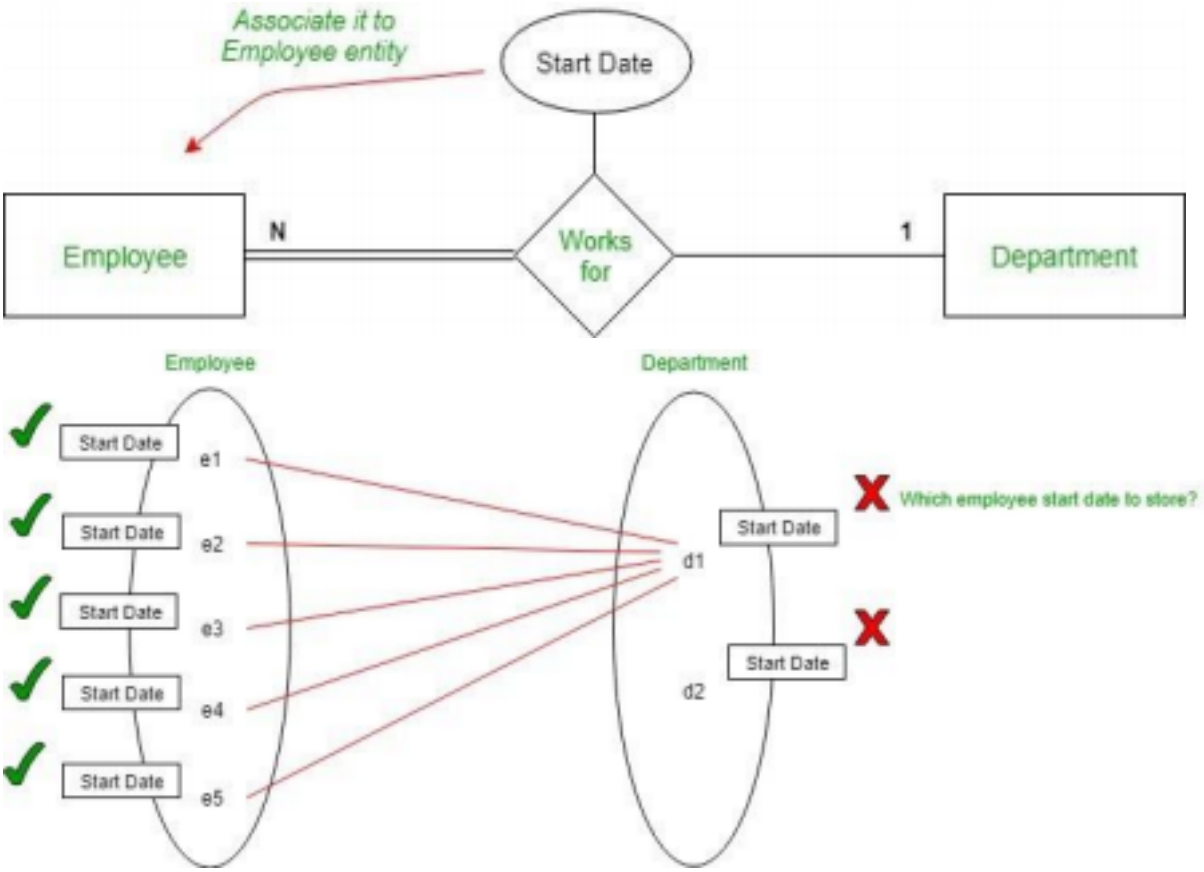
1. One to one relationship:

In an organisation an employee manages a department and each department is managed by some employee. So, there is a total participation of em>Department entity and there is *one to one* relationship between the given entities. Now, if we want to store the *Start_Date* from which the employee started managing the department then we may think that we can give the *Start_Date* attribute to the relationship *manages*. But, in this case we may avoid it by associating the *Start_Date* attribute to either *Employee* or *Department* entity.



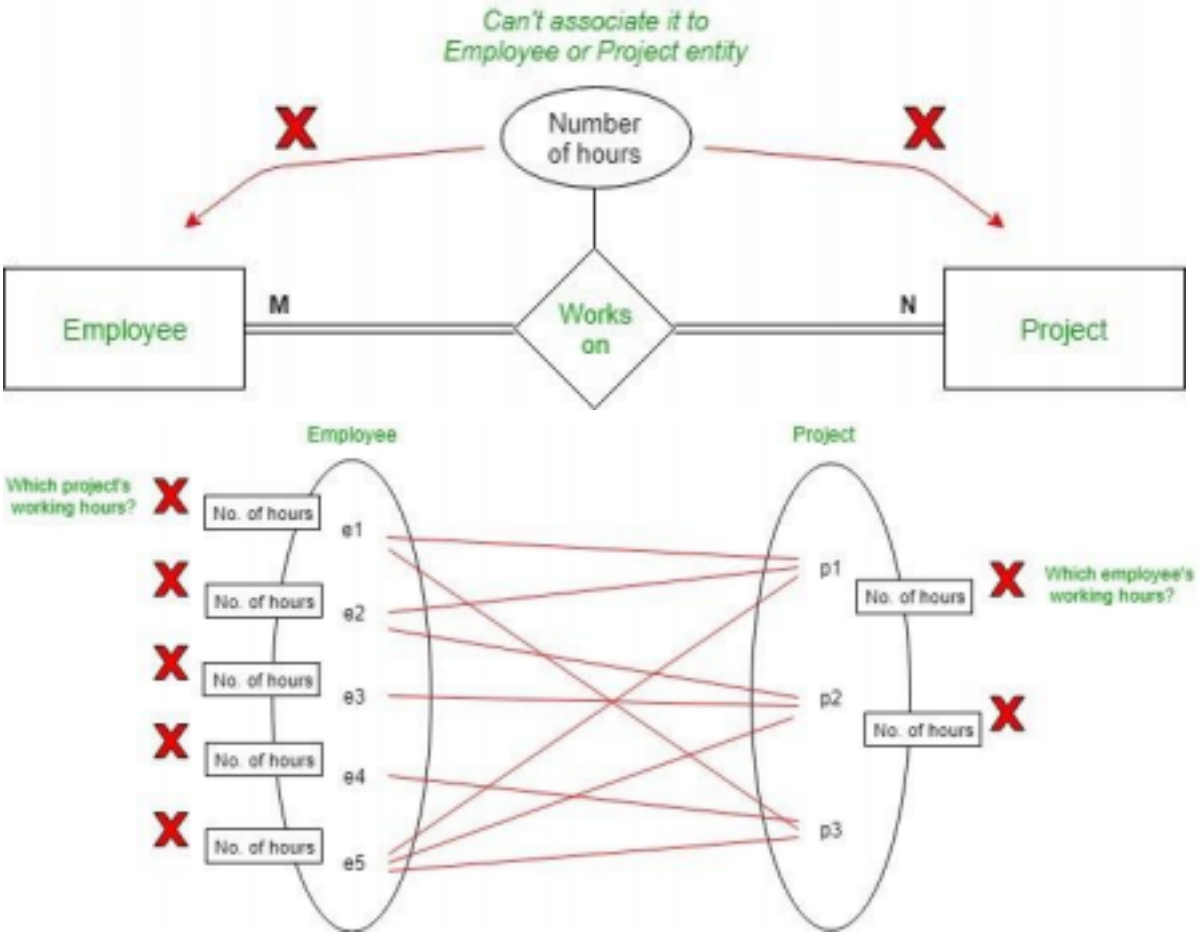
2. One to many relationship:

In an organisation many employees can work for a department but each employee can work for only a single department. So, there is a *one to many* relationship between the entities. Now if we want to store the *Start_Date* when employee started working for the department, then instead of assigning it to the relationship we should assign it to the *Employee* entity. Assigning it to the *employee* entity makes sense as each employee can work for only single department but on the other hand one department can have many employees working under it and hence, it wouldn't make sense if we assign Start_Date attribute to Department.



3. Many to many relationship:

In an organisation an employee can work on many projects simultaneously and each project can have many employees working on it. Hence, it's a *many to many* relationship. So here assigning the *Number_of_Working_hours* to the employee will not work as the question will be that it will store which project's working hours because a single employee can work on multiple projects. Similar the case with the *project* entity. Hence, we are forced to assign the *Number_of_Working_hours* attribute to the relationship.



Conclusion: Give attributes to a relationship only in the case of **many to many** relationship.

Q13. What is a Weak Entity Type ?

Weak Entity types are those that do not have a key attribute of their own. They are identified by being related to specific entities from some other strong entity type. This strong entity type is called identifying or owner type of that relationship. Eg Guardian can be a weak entity type of a student in a student table.

Q14. Explain Types of Keys.

Super Key - A set of one or more attributes , that , taken collectively, allow us to uniquely identify a tuple in the relation.
Candidate Key - Candidate Key is a minimal superkey for which no proper subset is a superkey.
Primary Key - One of the candidate key arbitrarily chosen by the database designer as a principle means of identifying tuples within a relation.

Alternate Key - It is a candidate key which is not a primary key.

Foreign Key - Foreign Key of relation R1 is a set of attributes of R1 which is a primary key of another relation R2. It is said to be referring to the primary key of R2 so that for any tuple of R1, the value of the foreign key must also be present in some tuple in relation R2.

N.B - All candidate keys are superkeys but the reverse is not

true. SQL

Q1. What are the differences between SQL and PL/SQL?

SQL	**PL/SQL**
SQL is a query execution or commanding language	PL/SQL is a complete programming language
SQL is data oriented language	PL/SQL is a procedural language
SQL is very declarative in nature	PL/SQL has a procedural nature
It is used for manipulating data	It is used for creating applications
We can execute one statement at a time in SQL	We can execute block of statements in PL/SQL

SQL tells database, what to do? PL/SQL tells database how to do We can

embed SQL in PL/SQL We can not embed PL/SQL in SQL

Q2.Explain Operators of SQL.

SQL Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators

Description	Example

Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators

ALL The ALL operator is used to compare a value to all values in another value set.
AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
BETWEEN The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
IN The IN operator is used to compare a value to a list of literal values that have been specified.

LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL The NULL operator is used to compare a value with a NULL value.
UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Q3 . What is the difference between BETWEEN and IN operators in SQL?

The **BETWEEN** operator is used to fetch rows based on a range of values. For example,

```
SELECT * FROM Students
WHERE ROLL_NO BETWEEN 20 AND 30;
```

This query will select all those rows from the table Students where the value of the

field ROLL_NO lies between 20 and 30.

The **IN** operator is used to check for values contained in specific

sets. For example,

```
SELECT * FROM Students
WHERE ROLL_NO IN (20,21,23);
```

This query will select all those rows from the table Students where the value of the

field ROLL_NO is either 20 or 21 or 23.

Q4 . What is the difference between CHAR and VARCHAR2 datatype in SQL?

Both of these datatypes are used for characters but varchar2 is used for character strings of variable length whereas char is used for character strings of fixed length. For example, if we specify the type as char(5) then we will not be allowed to store string of any other length in this variable but if we specify the type of this variable as varchar2(5) then we will be allowed to store strings of variable length, we can store a string of length 3 or 4 or 2 in this variable.

Q5. What is the difference between primary key and unique constraints?

Primary key cannot have any NULL value while the unique Key can have only one null value.

Q6 . What is a view in SQL?

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.
The CREATE VIEW statement of SQL is used for creating Views.

Basic Syntax:

CREATE VIEW view_name AS
SELECT column1, column2.....

FROM table_name
WHERE condition;
view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows

Q7. Explain Foreign Key with Syntax.

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in Orders table is the primary key in

Customers table, i.e. it uniquely identifies each row in the Customers table.

Therefore, it is a Foreign Key in Orders table.

Syntax:

```
CREATE TABLE Orders
 (
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

Q8. Explain Different types of Join in an SQL.

An SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are: • **INNER JOIN**: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

- **LEFT JOIN**:This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN
- **RIGHT JOIN**:RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.
- **FULL JOIN**: FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result set will contain NULL values.

Q9 . What is the difference between having and where clause?

HAVING is used to specify a condition for a group or an aggregate function used in select statement. The WHERE clause selects before grouping. The HAVING clause selects rows after grouping. Unlike HAVING clause, the WHERE clause cannot contain aggregate functions.

Q10. Write an SQL query to print duplicate rows in a table.

Let us consider below table.

NAME SECTION

abc CS1

bcd CS2

abc CS1

In the above table, we can find duplicate row using below query.

```
SELECT name, section FROM tbl
GROUP BY name, section
HAVING COUNT(*) > 1
```

Q11. What is Identity?

Identity (or AutoNumber) is a column that automatically generates numeric values. A start and increment value can be set, but most DBA leave these at 1. A GUID column also generates numbers; the value of this cannot be controlled. Identity/GUID columns do not need to be indexed.

Q12 . What are the uses of view?

1. Views can represent a subset of the data contained in a table; consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
2. Views can join and simplify multiple tables into a single virtual table **3.** Views can act as aggregated tables, where the database engine aggregates data (sum, average etc.) and presents the calculated results as part of the data **4.** Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table **5.** Views take very little space to store; the database contains only the definition of a view, not a copy of all the data which it presents.

Q13 . What is a Trigger?

A Trigger is a code that associated with insert, update or delete operations. The code is executed automatically whenever the associated query is executed on a table. Triggers can be useful to maintain integrity in database.

Q14. What is a stored procedure?

A stored procedure is like a function that contains a set of operations compiled together. It contains a set of operations that are commonly used in an application to do some common database tasks.

Q15. What is the difference between Trigger and Stored Procedure?

Unlike Stored Procedures, Triggers cannot be called directly. They can only be associated with queries.

Q16. What is the difference between Group By and Distinct?

Group By sorts in ascending order whereas distinct does not , otherwise both work the same for columns.

Q17 . Write an SQL query to find second highest salary.

```
SELECT name, MAX(salary) AS salary
 FROM employee
 WHERE salary < (SELECT MAX(salary)
 FROM employee);
```

Q18. Show the salaries of top 10 employees.

```
Select * from Employees order by salary desc limit 10. (MY SQL)
```

Q19. Show the names of employees who name starts with R.

Select name from Employee where name like 'R%'.

Q20. Show the name of employees whose second character in their name is I. (LIKE OPERATOR)

Select name from Employee where name like '_I%'.

NB - % replaces 0 or more character where _ replaces only one character

Q21. Show the salaries of John , Smith , Alice. (IN OPERATOR)

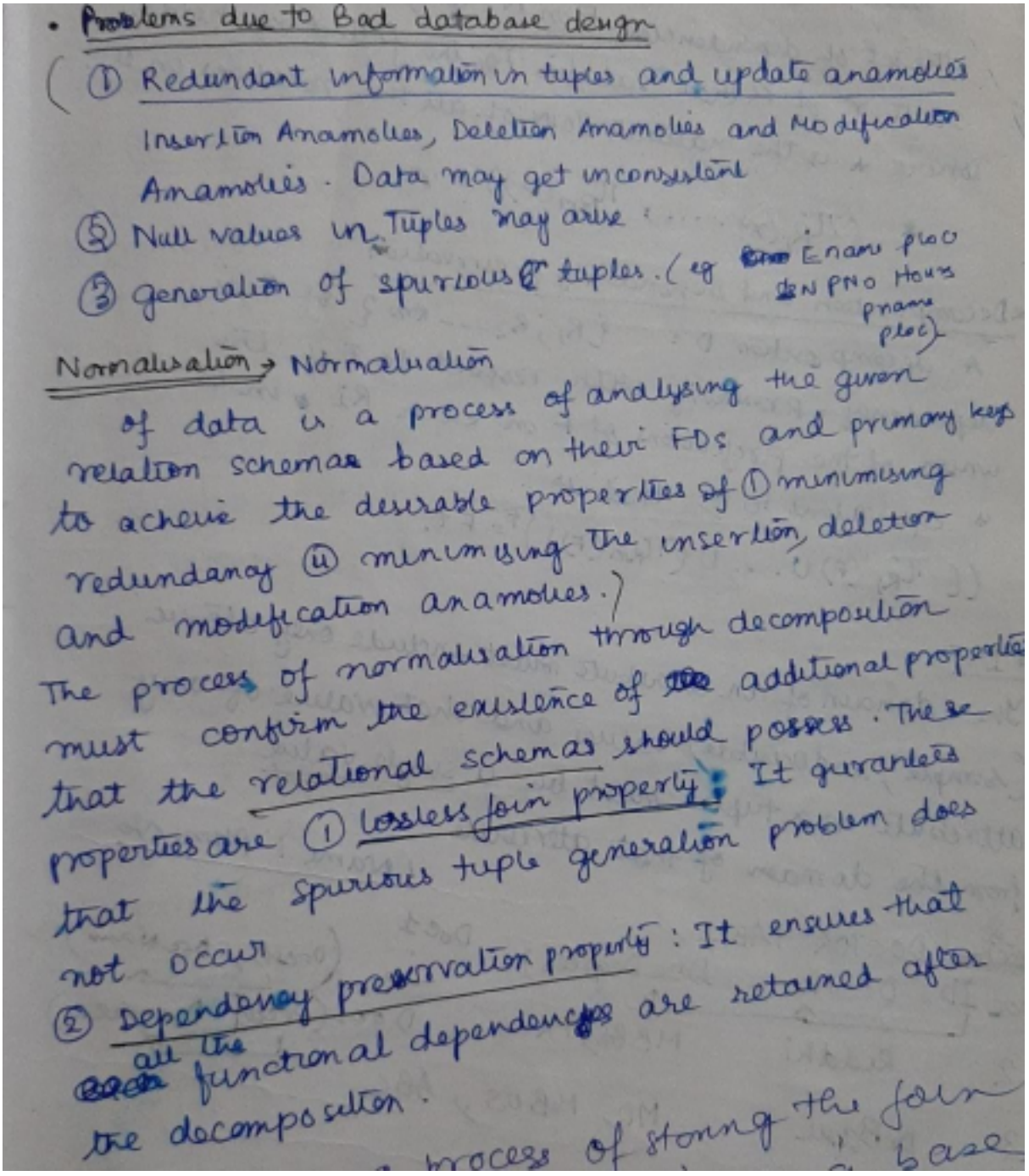
Select salary from Employee where name in ('John' , 'Smith' , 'Alice');

Q22. Show the name of Employees without having any phone number. (NULL VALUE CHECKING)

Select * from employees where phone_no is NULL.

NORMALISATION

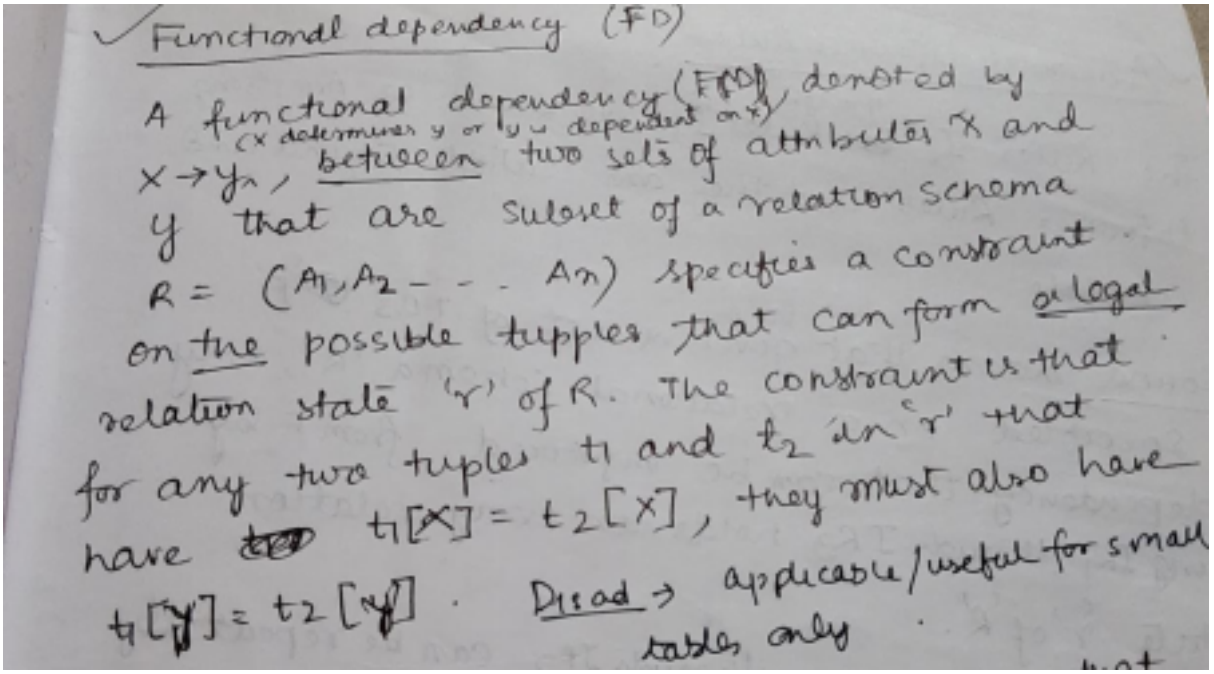
Q1. What is normalisation and why do we need it ?



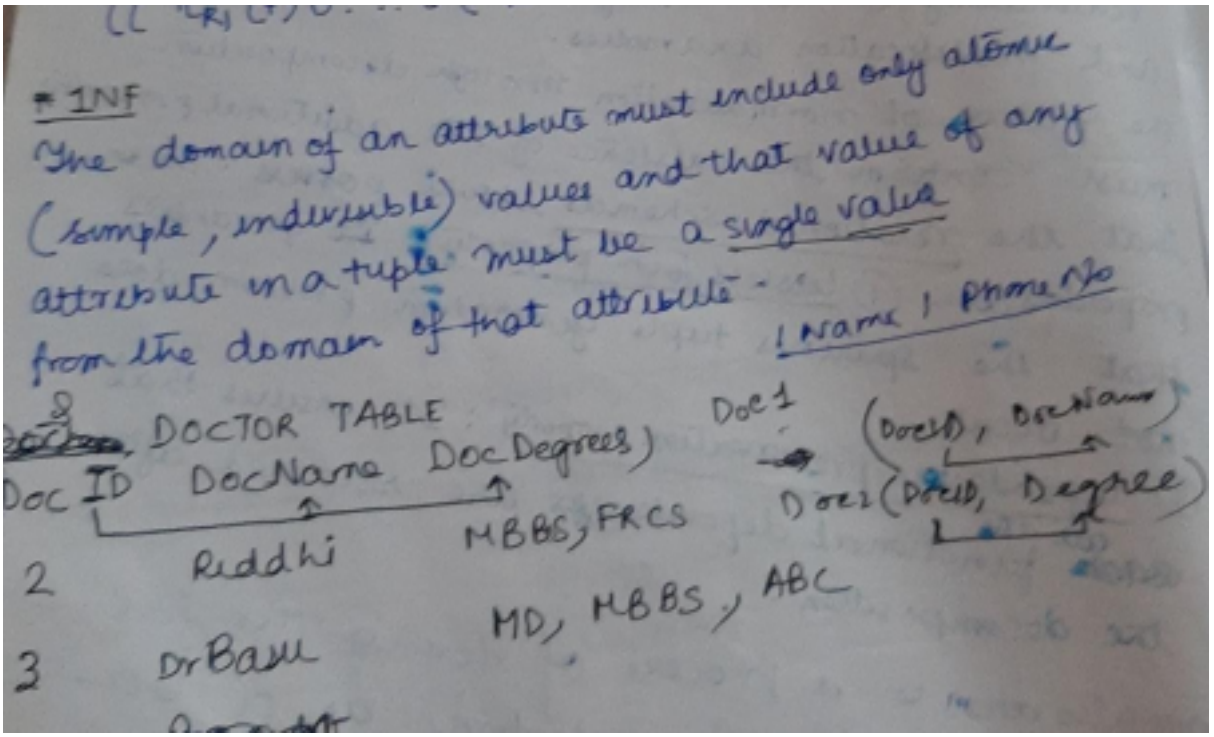
Q2. What is de-normalisation?

Process of storing the join of higher normal form relations as a base relation which is in a lower normal form. This is generally done for the sake of good performance.

Q3. What is Functional Dependency?



Q4. Explain 1NF.



Q5, Explain 2NF.

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:
TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology

47	English
83	Math
83	Computer

Q6. Explain 3NF.

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:
EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}
Non-prime attributes: In the given table, all attributes except EMP_ID are non prime.
Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.
That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.
EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

3.
EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich

462007	MP	Bhopal

Q7. Explain BCNF.

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India

264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

TRANSACTION

Q1. Explain ACID Properties.

Transaction is a unit of program execution that access and possibly updates various data items. To preserve the integrity of the database the system must ensure Atomicity , Consistency , Isolation and Durability.

Atomicity

The system must ensure that partially executed transactions / partially updated database should not be reflected in the database. Either all the operations of the transaction should be properly reflected in the database or none are.

It involves the following two operations.

—**Abort**: If a transaction aborts, changes made to database are not visible.

—**Commit**: If a transaction commits, changes made are visible.

Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

Referring to the example above,

The total amount before and after the transaction must be maintained.
Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let **X**= 500, **Y** = 500.

Consider two transactions **T** and **T".**

T	T"
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T"** starts. As a result , interleaving of operations takes place due to which **T"** reads correct value of **X** but incorrect value of **Y** and sum computed by

T": (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

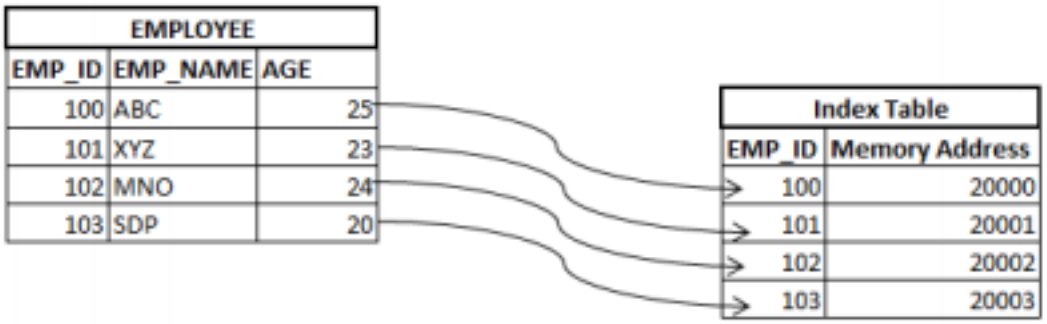
The **ACID** properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

INDEXING

Q1. What is an index?

Index is a data structure which is created on one or more columns of the table. In most of the cases, the indexes are structured as B tree. When index is created on a column of a table, it actually creates a lookup table with column and a pointer to the memory address where a row with this column is actually stored. Therefore when we query a table with column in WHERE clause, the parser first checks if this column is part of index, and if there is a index lookup table for it. If yes, then it checks for the memory address where the record is stored. It then directly jumps to that memory address and returns the result to the user. That is why index scan or if a

column has index, then it retrieves the data faster.



Q2. What kind of data structure is an index?

In most of the cases, B-Tree data structure is used to store the indexes. This is because of the time efficiency of B-Trees. In other words, B-trees are traversed, searched, inserted, deleted and updated in logarithmic time. In addition, B-Tree data are always sorted and stored. Hence it makes searching and inserting the data in known fraction of time. The data values stored in B-trees are balanced – all the values smaller than a particular node can be found at the left of the node and the values greater than the node value are found at the right of the node. Hence it is easy to search any value or record in B-tree indexes.

However, RDBMS actually determines which data structure needs to be used for index. In certain RDBM, we can tell which data structure needs to be used for the index.

Q3. How does a hash table index work?

In hash table index, index will be created on the columns based on the hash function used on the column. That means, hash function will be applied on the column on which index has to be created and that result will be the location of the record stored. Hence, here in this method all the records will be scattered in the memory.

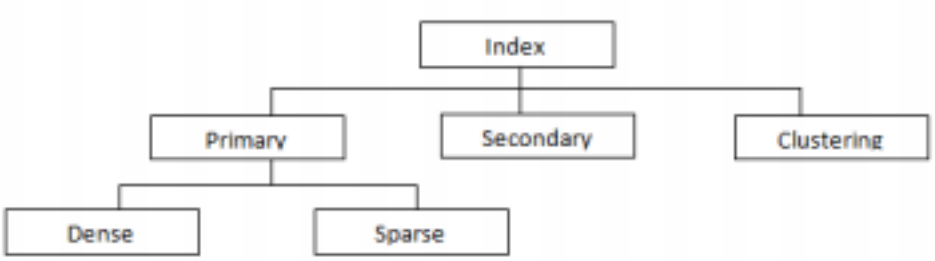
For example, consider that we have created hash index on PHONE_NUMBER column of the EMPLOYEES table. Let this hash function be any simple function to any complex function. Let us assume, in this case hash function be sum of all digits in the PHONE_NUMBER multiplied by 1000. Then if we need to search any particular phone number say, 546.897.231, then we will get it at the location 45000. Hence the pointer will go that location and retrieve required record details. If we need to see immediate next phone number details, then it will be at location 46000 which far away from previous phone number.

Q4. What are the disadvantages of hash index?

Hash indexes are created by creating a hash function on the column on which index needs to be created. Hence each column will have different

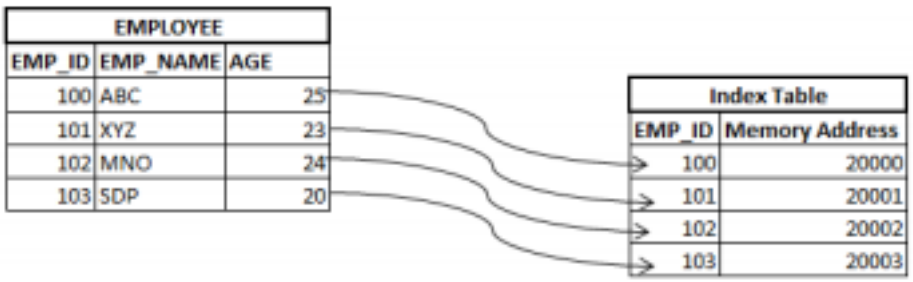
non-contiguous memory locations allocated to it. Therefore, if we need to search any contiguous records with conditions like ‘less than’ or ‘greater than’, then hash index will not be able to get all the records in one shot. It has to search for the records at various locations to get all the records. Hence it is not efficient for such searches. It is good only if we have to search key-value pairs. That means query with WHERE clause with ‘=’ condition will have better performance.

Q5. What are some other types of indexes?



Q6. What exactly is inside a database index?

When a index is created on a column or combination of columns in a table, another index lookup table is created with columns with which index is created and a pointer address to the memory location where the whole record of the table is stored. It will not have entire record information in the lookup table.



EMPLOYEE

EMP_ID	EMP_NAME	AGE
100	ABC	25
101	XYZ	23
102	MNO	24
103	SDP	20

Index Table

EMP_ID	Memory Address
100	20000
101	20001
102	20002
103	20003

Q7. How does a database know when to use an index?

When we run a query with condition ‘WHERE COLUMN_NAME = ‘XYZ’, then database will check if there is any index on this COLUMN_NAME. If there is index on this particular column, it will check for the selectivity of the column and decides if index has to be used or not. If the selectivity of the column is more than 0.33 then, index will be used to retrieve the data.

Q8. How to create an index in SQL?

The general syntax for creating index is :

```
CREATE INDEX index_name
ON TABLE_NAME (COLUMN_NAME);
CREATE INDEX idx_phone
ON EMPLOYEES (PHONE NUMBER);
```

Q9. How to create a multi-column index in SQL? We can create index on combination of columns. The general syntax for creating multi-column index is :

```
CREATE INDEX index_name
ON TABLE_NAME (COLUMN_NAME1, COLUMN_NAME2,.. COLUMN_NAMEN);
CREATE INDEX idx_emp_name
ON EMPLOYEES (FIRST_NAME, LAST_NAME);
```

Q10. Explain Primary , Secondary and Clustered Indexes.

- Primary Index − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- Secondary Index − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- Clustering Index − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Q11. Explain Ordered Indexing.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

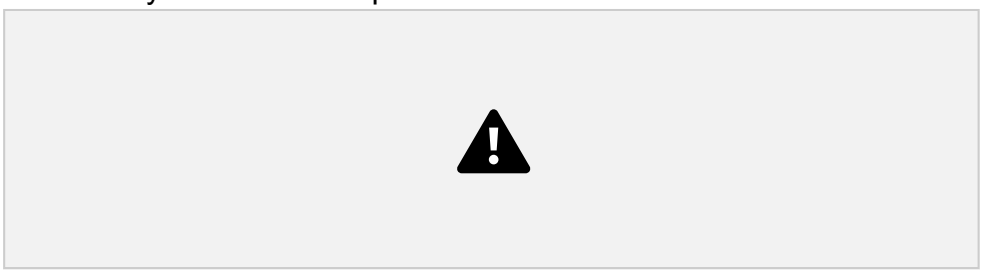
Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

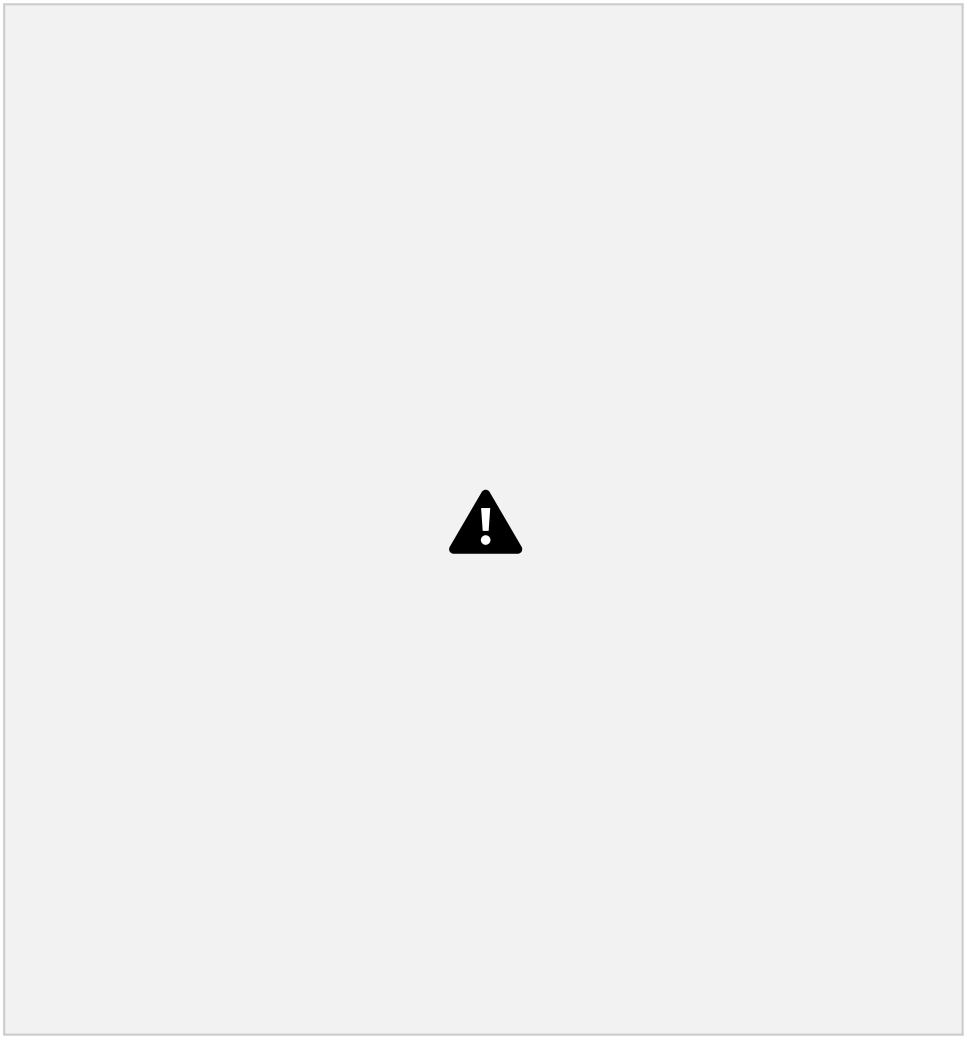
In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a

record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



Q12. Explain Multilevel Indexing

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



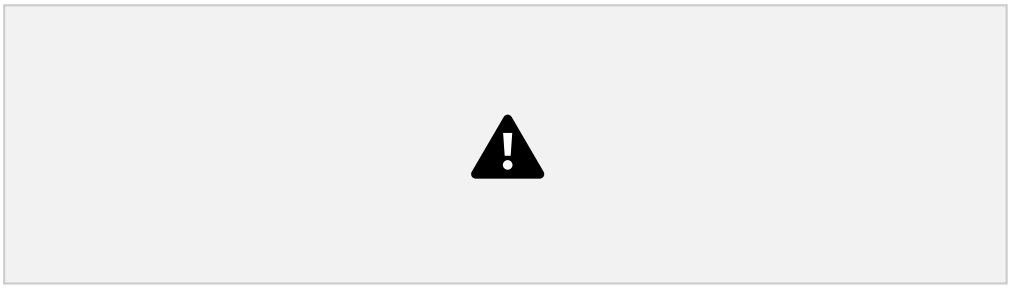
Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

Q13. What are B+ Trees?

A B_+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B_+ tree denote actual data pointers. B_+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B_+ tree can support random access as well as sequential access.

Structure of B+ Tree

Every leaf node is at equal distance from the root node. A B_+ tree is of the order n where n is fixed for every B_+ tree.



Internal nodes −

• Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node. • At most, an internal node can contain n pointers.

Leaf nodes −

• Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values. • At most, a leaf node can contain n record pointers and n key values. • Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.