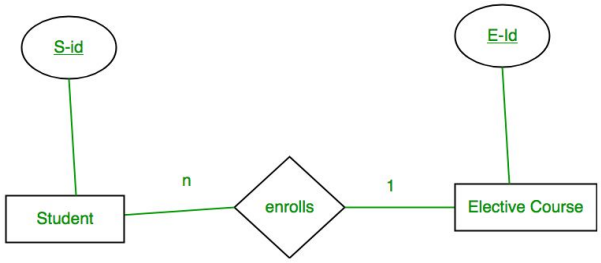
DBMS Interview Question

1. Define data, database, data warehouse, database system, DBMS
    a. Data: any fact that can be recorded
    b. Database: collection of related data
    c. Data warehouse: a type of db with large stored records
    d. Database system: database+DBMS software = database system
    e. DBMS: stores data in such a way that it becomes easier to retrieve, manipulate and produce information
2. Uses of DBMS
    a. Consistency
    b. Better manage data
    c. Back up and recovery
3. What is Data Independence?
    - Data independence specifies that "the application is independent of the storage structure and access strategy of data." It makes you able to modify the schema definition at one level without altering the schema definition in the next higher level.
4. Different models in DBMS. or What are the different levels of abstraction in the DBMS? Or 3 levels of data abstraction?
   Defines how logical structure of db is modeled
    a. High level or ER model = small model for layman
    b. Representational or Relational model = we use tables / blueprint
    c. Low level or physical = actual building
5. ER model?
   Customers tell requirements (based on real-world entities and their relationships)
    - Entity = a thing / an instance
    - Attributes = characteristics
    - Relationships = association
    - Entity type -> entity(instance) person(name,age,add. ….) -> (26, raju, kjd)
    - Entity type = (schema or heading) intension
    - Entity = extension

- Entity set = all entities of that entity type
- State of db = at any time number of elements

6. Explain Entity, Entity Type, and Entity Set in DBMS?
    a. entity= an object, place or a thing e.g. person, book etc.
    b. Entity type = collection of entities which have same attributes
    STUDENT table contains rows in which each row is an entity holding attributes like name, age , and id of the students, hence STUDENT is an Entity Type which holds the entities having same attributes.
    c. Entity set = a collection of the entities of the same type. Eg: A collection of the employees of a firm.
7. Different types of attributes
    a. Simple vs composite
    b. Single vs multivalued
    c. Stored vs derived
    d. complex
8. Different relationships
    - Association among entities
    - Degree in a relationship = number of entity sets participating
    - Cardinality = maximum number of relationships an entity can participate in (1:N)



every student can enroll only in one elective course but for an elective course there can be more than one student.

- Participation or existence = minimum number of relationships an entity can participate in (min-max representation)

9. Relationships
    - One to one
    - One to many

- Many to many
- Recursive relationship [between two same entities , degree = 2(binary)]

10. Attribute to relationships
    - Put to n side
    - Weak entity = total participation
    - Strong entity = might or might not
11. Relational DB and terms
    - Relational db (tables)
    - RDBMS
    - Relation, tuple, attributes, domain {set of values}, relation schema, degree of relation(no. Of attributes), relation state
    - r(R) subset of D1 x D2 x D3 x D4 ……
    - Relation is a set, duplicates are not allowed
    - NULL
12. Relational constraints

    Restrictions on data to be inserted. Conditions that hold for it to be a valid relation.

    a. Domain constraint = every attribute should lie in defined DOMAIN
    b. Key constraint = uniqueness + no NULL
    c. Entity constraint = no NULL (primary key != NULL value)
    d. Referential integrity constraint = if a relation refers to a same relation, then that key element must exist
    e. Integrity constraints = entity integrity + referential integrity
13. Keys
    a. Super key = a set of attributes that collectively identifies an entity in an entity set
    b. Candidate key = a minimal super key
    c. Primary key = one of candidate keys chosen by db designer to uniquely identify the entity set
    d. Foreign key = a key attribute of a relation that can be referred to in other relation. Used to link two or more tables together
14. Anomalies

a. Insertion anomaly
    - Attribute is not present
b. Deletion anomaly
    -
c. Updation anomaly

15. Actions upon constraint violations
    a. Reject
    b. Cascade
    c. Set NULL or some other value
16. Count number of super keys given candidate keys
17. Conversion of ER to relational
    a. Convert entity to relation
    b. Convert weak entities into relations
        - Partial key
    c. Convert relationships into relation
    d. Deal with multivalued
        - New table
    e. Deal with n-ary relationship
18. What is normalization, functional dependency, types of FD, closure set of attributes, lossy and lossless decomposition, FD preservation.
    a. Normalization -
        - A method to remove all the anomalies and bring the db to a consistent state.
        - To remove redundancy
    b. Functional dependency
        - This is basically a constraint which is useful in describing the relationship among the different attributes in a relation.
        - A -> B
            - If t1(A) = t2(A), then t1(B)=t2(B)
        - If two tuples have same values for attributes A1,A2,A3….., then these two tuples must have same values for attributes B1,B2,B3….

- Main use
    - Will be helpful in splitting/decomposition
    - For reducing redundancy

c. Types of FD
    - Trivial = A -> B where B is subset of A
    - Non trivial = A -> B where B is not subset of A and A intersect B is not NULL
    - Completely non trivial =A -> B where B is not subset of A and A intersect B is NULL

d. Closure set of attributes
   Find candidate key (max 2^n-1)
    - Equivalent FDs = FD2 ⊃ FD1 and FD1 ⊃ FD2

e. Lossy
    - We get extra info

f. Lossless
    - No extra anything

g. FD preservation
    - All FDs are recovered

19. What is denormalization?
    a. Denormalization is the process of boosting up database performance and adding of redundant data which helps to get rid of complex data. Denormalization is a part of database optimization technique. This process is used to avoid the use of complex and costly joins. Denormalization doesn't refer to the thought of not to normalize instead of that denormalization takes place after normalization. In this process, firstly the redundancy of the data will be removed using normalization process than through denormalization process we will add redundant data as per the requirement so that we can easily avoid the costly joins.

20. Normal forms
    a. 1NF
        - All the attributes in a relation must have atomic domains

- No multivalued or no composite

b. 2NF
   - We do not allow partial dependency
   - Prime attributes = part of primary key
     Non-prime = not part of primary key
   - It states that every non-prime attribute should be fully dependent on prime key i.e. if X->A holds then there shouldn't be any properset Y of X for which Y->A also holds true.
   - Both lossless and FD preserving
   - How to solve? Closure of LHS
   - X -> A (partial dependency)
     partial key non key

c. 3NF
   - Eliminate redundancy caused by transitive dependency
   - Must be 2NF
   - Lossless and FD preserved
   - X -> A (transitive)
     Non key non key
   - 3NF
     X -> A
     super attribute or prime attribute

d. BCNF
   - Boyce codd NF
   - X -> A
     Must be super attribute

21. Explain ACID properties
    a. Atomicity
       - all or nothing rule.' Which implies all are considered as one unit, and they either run to completion or not executed at all.
    b. Consistency

- This property refers to the uniformity of the data. Consistency implies that the database is consistent before and after the transaction.

c. Integration

- This property states that the number of the transaction can be executed concurrently without leading to the inconsistency of the database state.

d. Durability

-This property ensures that once the transaction is committed it will be stored in the non-volatile memory and system crash can also not affect it anymore.

22. Relational algebra?
    - What we want and how we want it.
23. Basic operations and Relational algebra expression?
    - Basic operations
        - Projection (π)
            - Vertical partitioning
            - Not commutative
            - Duplicates not allowed
            - Cardinality = 1(min) , all attributes(max)
        - Selection (σ)
            - Horizontal partitioning
            - Commutative A.B = B.A
            - Cardinality = 0(min) , |R|(max)
        - Union (U)
        - Set Difference (-)
        - Rename (ρ)
        - Cross Product (X)
    - Relational algebra expressions
        - Join (⋈)
        - Division (/)
        - Intersection (∩) = (A-(A-B))

24. Set operations and properties
    - Set operations
        - Binary ∩,U,- = tables must be union compatible i.e. same degree and same domain
        - Unary π,σ,ρ
    - Properties

.	commutative	associative
U	yes	yes
∩	yes	yes
-	yes	yes

    - Cartesian product
        - R x S
        - |R| = m d(R) = k
        - |S| = n d(S) = l
        - |RxS| = mn d(RxS) = k+l

25. Joins.
    - Join = σ,x
    - Cardinality = 0(min), mn(max)
    - Meaningful extension of cartesian product

Cartesian product	mn(min)	mn(max)
INNER JOIN	0	mn
LEFT JOIN	m	mn
RIGHT JOIN	n	mn
FULL JOIN	max(m,n)	mn

    - Theta join
        - If comparison is between 2 attributes <A=B> not <A>10>
    - Natural join
        - When you join two tables which have common attributes with same name
        - You may use rename operation

26. Complete set?
    - σ,x,U,-,π

27. Extended Relational Algebra operation
    - Arithmetic operations
    - Aggregation operations = sum, average, MAX,MIN, COUNT, COUNT_DISTINCT
28. Relational Algebra vs Relational calculus
    - RA is procedural language(what and how i.e. order)
    - RA has same power as safe RC
    - Safe RC = RC - unsafe RC = RA
29. Relational calculus
    - RC is declarative language(what we want) or non-procedural
    - Tuple RC =
        - we will examine a tuple as a whole
        - There can be Unsafe operations that lead to infinite loop
        - {t| P(t)}
        -
    - Domain RC = we work on columns
30. Relationally complete language?
    - If a language is able to express everything that RA expresses
    - RA = DRC = TRC
    - All the languages will try to be relationally complete else we will not be able to apply some of the operations on data
31. Free and bounded variables
    - {t1.A1 , t2.A2| P1(t1)AND P2(t2)}
    - Variables that are not present in LHS, they will be at RHS. called bounded variables.
    - Wherever we use qualifiers = bounded variables
    - We do not use qualifiers = free variables
32. What are the different types of languages that are available in the DBMS?
    a. DDL: DDL is Data Definition Language which is used to define the database and schema structure by using some set of SQL Queries like CREATE, ALTER, TRUNCATE, DROP and RENAME.

b. DCL: DCL is Data Control Language which is used to control the access of the users inside the database by using some set of SQL Queries like GRANT and REVOKE.
c. DML: DML is Data Manipulation Language which is used to do some manipulations in the database like Insertion, Deletion, etc. by using some set of SQL Queries like SELECT, INSERT, DELETE and UPDATE.

SQL

33. Difference between SQL and MySql
34. How do you communicate with an RDBMS?
    - You have to use Structured Query Language (SQL) to communicate with the RDBMS.
    - Using queries of SQL
35. What is the purpose of SQL?
    a. To interact with relational databases in the form of insertion, updating/modifying etc.
36. B
    a. SCHEMA
        - CREATE SCHEMA schema_name [AUTHORIZATION owner_name];
        - CREATE SCHEMA FINANCE AUTHORIZATION RAVI, GRANT SELECT ON FINANCE * to user 1;
    b. CREATE TABLE
        - CREATE TABLE <table name>(<d1> datatype (width), <d2> datatype <width>, ...);
        - CREATE TABLE customer (Name varchar(20), cust_id Number);
    c.
37. What is the concept of sub-query in terms of SQL?
    - Query inside a query

- Aka inner query which is found inside outer query

38. What is a CLAUSE in terms of SQL?
    - his is used with the SQL queries to fetch a specific data as per the requirements on the basis of the conditions that are put in the SQL. This is very helpful in picking the selective records from the complete set of the records.
    - Example: There is a query which has WHERE condition or the query with the HAVING clause.
39. Constraints

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);
```

a. NOT NULL - Ensures that a column cannot have a NULL value

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
```

b. UNIQUE - Ensures that all values in a column are different

```
CREATE TABLE Student
(
```

```
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

c. PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

d. FOREIGN KEY - Uniquely identifies a row/record in another table

```
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)
```

)

e. CHECK - Ensures that all values in a column satisfies a specific condition

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

f. DEFAULT - Sets a default value for a column when no value is specified

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

g. INDEX - Used to create and retrieve data from the database very quickly

CREATE INDEX *index_name*

ON *table_name* (*column1*, *column2*, ...);

CREATE INDEX idx_pname

ON Persons (LastName, FirstName);

40. SQL Queries
    a. INSERT (2 ways)
       INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
       VALUES (*value1*, *value2*, *value3*, ...);
       INSERT INTO *table_name*
       VALUES (*value1*, *value2*, *value3*, ...);
        - Attribute values should be in order of attribute list
        - If you do not mention the value, it takes default value or NULL if default is not mentioned
    b. DELETE
       DELETE FROM *table_name* WHERE *condition*;
       DELETE FROM EMP WHERE emp_id=1;
        - Problem = integrity constraint
    c. DROP
        - Drop db
          DROP DATABASE *databasename*;
        - Drop table
          DROP TABLE *table_name*;
            - Problem = integrity constraint
    d. UPDATE
       UPDATE *table_name*
       SET *column1 = value1*, *column2 = value2*, ...
       WHERE *condition*;
        - Problem = integrity constraint VIOLATION
    e. b
41. SQL everything

a. SQL select
    - The SELECT statement is used to select data from a databse
    - SELECT *column1*, *column2, ...*

FROM *table_name*;

- SELECT * FROM *table_name*;

b. SQL select distinct
   - To return only distinct values
   - *SELECT DISTINCT column1, column2, ...*
     FROM table_name;
   - *SELECT DISTINCT Country FROM Customers;*
   - *SELECT COUNT(DISTINCT Country) FROM Customers; // to count distinct number of countries*

c. SQL where
   - Used to filter records
   - *SELECT col1, col2, ...*
     FROM table_name
     WHERE condition;
   - *SELECT * FROM Customers*
     WHERE Country='Mexico';

-

d. SQL And, Or, Not

- The WHERE clause can be combined with AND, OR, and NOT operators.
- *SELECT column1, column2, ...*
  FROM table_name
  WHERE condition1 AND condition2 AND condition3 ...;
- *SELECT column1, column2, ...*
  FROM table_name
  WHERE condition1 OR condition2 OR condition3 ...;
- *SELECT column1, column2, ...*
  FROM table_name
  WHERE NOT condition;

e. SQL Order by

- The ORDER BY is used to sort the result set in a certain order
- *SELECT column1, column2, ...*
  FROM table_name
  ORDER BY column1, column2, ... ASC|DESC;

- *SELECT * FROM Customers*
  ORDER BY Country DESC; // default is ASC
- *SELECT * FROM Customers*
  ORDER BY Country, CustomerName; //if some rows have same country then order them by CustomerName.
- *SELECT * FROM Customers*
  ORDER BY Country ASC, CustomerName DESC;

f. SQL Insert into

- To insert new records into a table
- *INSERT INTO table_name (column1, column2, column3, ...)*
  VALUES (value1, value2, value3, ...);
- *INSERT INTO table_name*
  VALUES (value1, value2, value3, ...);
- *If any column skipped then default or NULL value*

g. SQL NULL values

- A field with NULL value is a field with no value
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.
- *SELECT column_names*
  FROM table_name
  WHERE column_name IS NULL;
- *SELECT CustomerName, ContactName, Address*
  FROM Customers
  WHERE Address IS NULL;

h. SQL Update

- To modify the existing records in a table
- *UPDATE table_name*
  SET column1 = value1, column2 = value2, ...
  WHERE condition;
- *Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!*
- *UPDATE Customers*

SET ContactName='Juan';

i. SQL Delete

- Delete existing record in a table
- DELETE FROM *table_name* WHERE *condition*;
- If omit where clause = all records from table are deleted

j. SQL Limit

- Used to specify number of records to return.
- *SELECT column_name(s)*
  FROM table_name
  WHERE condition
  LIMIT number;
- *SELECT * FROM Customers*
  LIMIT 3;

k. SQL Min , Max

- Return smallest or largest value of the selected column
- *SELECT MIN(column_name)*
  FROM table_name
  WHERE condition;
- *SELECT MIN(Price) AS SmallestPrice*
  FROM Products;

l. SQL Count, Avg, Sum

- The COUNT() function returns the number of rows that matches a specified criteria.
- *SELECT COUNT(column_name)*
  FROM table_name
  WHERE condition;
- The AVG() function returns the average value of a numeric column.
- SELECT AVG(*column_name*)
  FROM *table_name*
  WHERE *condition*;
- The SUM() function returns the total sum of a numeric column.
- *SELECT SUM(column_name)*
  FROM table_name

WHERE condition;

m. SQL Like

- LIKE operator is used in a WHERE clause to search a specific pattern in a column
- % = zero,one or multiple characters
- _ = single character
- *SELECT column1, column2, ...*
  FROM table_name
  WHERE columnN LIKE pattern;
- *SELECT * FROM Customers*
  WHERE CustomerName LIKE '_r%';

n. SQL Wildcard

- Used to substitute one or more characters in a string
- Used with SQL LIKE operator
- *Wildcard characters e.g. _ , %*
- *SELECT * FROM Customers*
  WHERE City LIKE 'ber%';

o. SQL In

- Allows you to specify multiple values in a WHERE clause
- IN operator is a shorthand for multiple OR condition
- *SELECT column_name(s)*
  FROM table_name
  WHERE column_name IN (value1, value2, ...);
- *SELECT column_name(s)*
  FROM table_name
  WHERE column_name IN (SELECT STATEMENT);
- *SELECT * FROM Customers*
  WHERE Country IN ('Germany', 'France', 'UK');
- *SELECT * FROM Customers*
  WHERE Country NOT IN ('Germany', 'France', 'UK');
- *SELECT * FROM Customers*
  WHERE Country IN (SELECT Country FROM Suppliers);

The above SQL statement selects all customers that are from the same countries as the suppliers:

p. SQL Between

- Operator selects values within a given range.
- Inclusive of begin and end values
- *SELECT column_name(s)*
  FROM table_name
  WHERE column_name BETWEEN value1 AND value2;
- *SELECT * FROM Products*
  WHERE Price NOT BETWEEN 10 AND 20;

q. SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- An alias only exists for the duration of the query.
- Alias column
  SELECT *column_name* AS *alias_name*
  FROM *table_name;*
- Alias table
  SELECT *column_name(s)*
  FROM *table_name* AS *alias_name;*
- *SELECT CustomerID AS ID, CustomerName AS Customer*
  FROM Customers;

r. SQL Join

- Clause used to combine rows from two or more tables, based on related column between them
- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

s. SQL Inner Join

- Selects records that have matching values in both tables
- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*

ON *table1.column_name = table2.column_name*;

t. SQL Left Join

- *SELECT Customers.CustomerName, Orders.OrderID*
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
  ORDER BY Customers.CustomerName;

u. SQL Right Join

- *SELECT Orders.OrderID, Employees.LastName, Employees.FirstName*
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
  ORDER BY Orders.OrderID;

v. SQL Full Join

- *SELECT Customers.CustomerName, Orders.OrderID*
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

w. SQL Self Join

- A regular join, just with itself
- *SELECT column_name(s)*
- *FROM table1 T1, table1 T2*
  WHERE condition;
- *SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City*
  FROM Customers A, Customers B
  WHERE A.CustomerID <> B.CustomerID
  AND A.City = B.City
  ORDER BY A.City;

x. SQL Union

- The UNION operator is used to combine the result-set of two or more SELECT statements.
  - Each SELECT statement within UNION must have the same number of columns

    - The columns must also have similar data types
    - The columns in each SELECT statement must also be in the same order
- SELECT *column_name(s)* FROM *table1*
  UNION
  SELECT *column_name(s)* FROM *table2*; //union selects only distinct values, use UNION ALL
- SELECT *column_name(s)* FROM *table1*
  UNION ALL
  SELECT *column_name(s)* FROM *table2*;
- SQL UNION with WHERE clause
  SELECT City, Country FROM Customers
  WHERE Country='Germany'
  UNION
  SELECT City, Country FROM Suppliers
  WHERE Country='Germany'
  ORDER BY City;

y. SQL Group BY

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.
- ORDERBY performs sorting while GROUPBY AGGREGATES Data
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  ORDER BY *column_name(s);*
- *SELECT COUNT(CustomerID), Country*
  FROM Customers
  GROUP BY Country;
- *SELECT COUNT(CustomerID), Country*
  FROM Customers

GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;

z. SQL Having

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  HAVING *condition*
  ORDER BY *column_name(s);*
- *SELECT COUNT(CustomerID), Country*
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5
  ORDER BY COUNT(CustomerID) DESC;

aa. SQL Exists

- Used to test for existence of any record in a subquerry
- The EXISTS operator returns true if the subquery returns one or more records.
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE EXISTS
  (SELECT *column_name* FROM *table_name* WHERE *condition*);

bb. SQL Any, All

- The ANY and ALL operators are used with a WHERE or HAVING clause.
- The ANY operator returns true if any of the subquery values meet the condition.
- The ALL operator returns true if all of the subquery values meet the condition.
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name operator* ANY

(SELECT *column_name* FROM *table_name* WHERE *condition*);

42. SQL DB

a. SQL Create DB

- CREATE DATABASE *databasename*;

b. SQL Drop DB

- DROP DATABASE *databasename*;

c. SQL Create Table

- The CREATE TABLE statement is used to create a new table in a database.
- CREATE TABLE *table_name* (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ....
  );
- CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
  );

d. SQL Drop Table

- Used to drop an existing table in a db
- DROP TABLE *table_name*;

e. SQL Alter Table

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.
- ALTER TABLE *table_name*
  ADD *column_name datatype*;
- ALTER TABLE *table_name*

DROP COLUMN *column_name*;

- ALTER TABLE *table_name*
  MODIFY COLUMN *column_name datatype*;

f. SQL Constraints

- CREATE TABLE *table_name* (
- *column1 datatype constraint*,
- *column2 datatype constraint*,
- *column3 datatype constraint*,
- ....
-);

g. SQL Not Null

- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255) NOT NULL,
  Age int
  );

h. SQL Unique

- CREATE TABLE Persons (
  ID int NOT NULL UNIQUE,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int
  );

i. SQL Primary Key

- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (ID)
  );

j. SQL Foreign Key

- CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
  PRIMARY KEY (OrderID),
  FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
  );

k. SQL Check

- CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
  CHECK (Age>=18)
  );

l. SQL Default

- CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
  );

m. SQL Auto Increment

n. SQL Dates

o.

43. Difference between DROP andDELETE
    - DROP =
        - DDL_____used to drop/delete the existing table, database, index or view from the database.
        - cannot be rolled back
    - TRUNCATE

- DDL_____used to drop/delete the existing table, database, index or view from the database.
- TRUNCATE command is used to remove all rows (complete data) from a table. It is similar to the DELETE command with no WHERE clause.
- cannot be rolled back

- DELETE
  - DML_____also used to delete rows from the table
  - Can be rolled back

44. What is meant by trigger?
    a. Trigger is one of the very important codes or programs which get executed automatically in response to the events that occur in a table or a view. Eg: If a new record is inserted in an employee database then the data gets created automatically in the related tables like salary, department and roles tables.

45. b

FILE ORGANISATION

46. Why a separate DBMS and not OS?
    - Database -> files -> records -> fields
    - For file organisation, we could rely on OS but when we demand some file from OS, our intention is not just one line of file. We intend to see the file from beginning to the end. Therefore, the way we access the normal data and the way we access the db both are different.
    - In db we might not need to access whole table, we just be interested in 1 particular record. Thus if we use OS, it would just be waste of everything.
    - Therefore, the best thing id to let DBMS come out with its own software using which it can interact with OS and later whatever type of access it wants to do , it can optimize accordingly.
47. What is blocking factor?
    - Database will be divided into blocks.
    - Blocking factor = average number of records in a block

48. Strategies for storing files of record into block?
    a. Spanned strategy
        i. It allows partial part of record to be stored in a block.
        ii. Advantage = no wastage of memory
        iii. Disadvantage = number of block accesses to access a block increases.
        iv. This strategy is suitable for variable length records.
        v. Example = URL and content table.
    b. Unspanned strategy
        i. No record can be stored in more than 1 block.
        ii. Advantage = number of block accesses to access a block is less.
        iii. Disadvantage = memory wastage
        iv. This strategy is suitable for fixed length records.
        v. Example = employee table.
49. Ways of storing files?
    a. Ordered file organisation
        - All records in a file are ordered on some search key. Key can even be non-contiguous but should be ordered.
        - Searching = binary
        - Advantage = searching a record is efficient
        - Disadvantage = insertion is expensive
        - You can only order records by one key
        - Eg. if u are ordering it by SSN, you cannot order it by ename also. Even though file is ordered, if you search based on ename, it would be unordered.
        - You get advantage only if you search by key.
    b. Unordered file organisation
        - All records in file are inserted wherever the place is available, usually at the end of file
        - Searching = linear
        - Advantage = inserting a record is efficient
        - Disadvantage = searching a record is inefficient compared to ordered file organisation

50. What is indexing and its need?
    - Why?
        - Say you want to read a topic from book, but you don't know the exact page number, but you know the order. What do you do is you randomly open the book and then compare and so on until you find the topic. This is called binary search.
    - Better way is to maintain an index
    - Advantage = index file will be smaller compared to the data file
    - Data file = key field + data
    - Index file = key field + pointer
    - Now index file is also a file therefore we can also divide it into blocks and then apply binary search.
    - Ordering can only be done based on 1 field and index files will also be ordered files.

Search key	Block pointer

    - Block pointer = pointer to block where key is available
    - Index is an ordered file
    - Searching = binary search
    - To access a record using index,

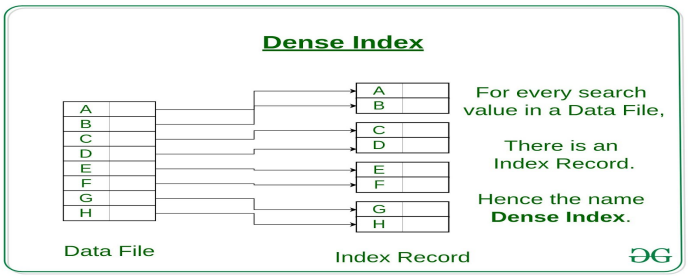
        Average number of block access = $\log(B_i) + 1$

        B_i = Index block access count

        1 = Data access
    - Index can be created on any field of relation (Primary key, non-key, candidate key). Even for unordered file.

51. Classification of indexing
    a. Dense Indexing
        - If an index entry is created for every search key value.

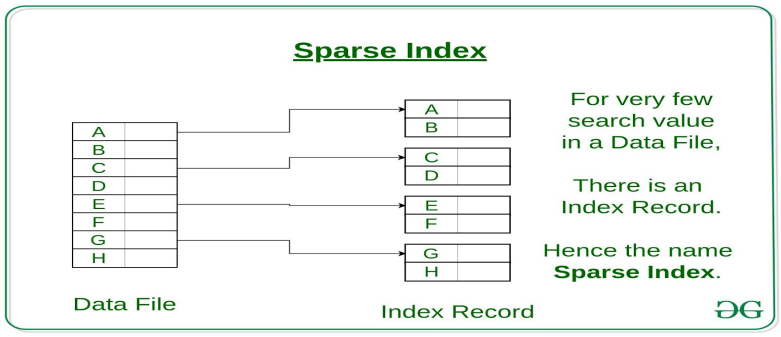
-



b. Sparse Indexing

- If an index entry is created for some records

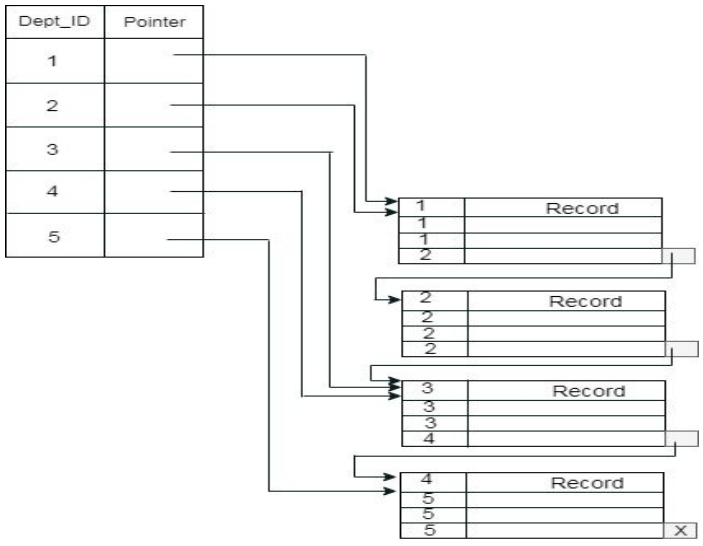
-

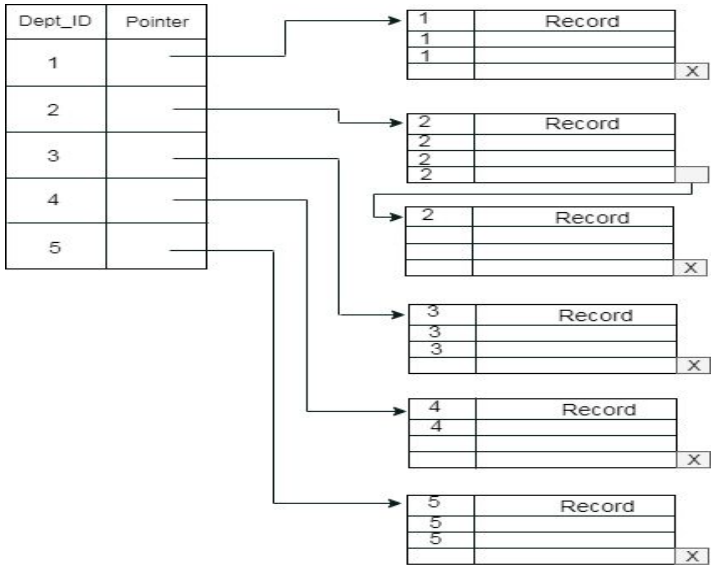


c. Both

- We would think both are complementary of each other. But they are not.
- For every distinct key, we are having an index. But only for some records.
- Sometimes the index is created on non-primary key columns which may not be unique for each record.

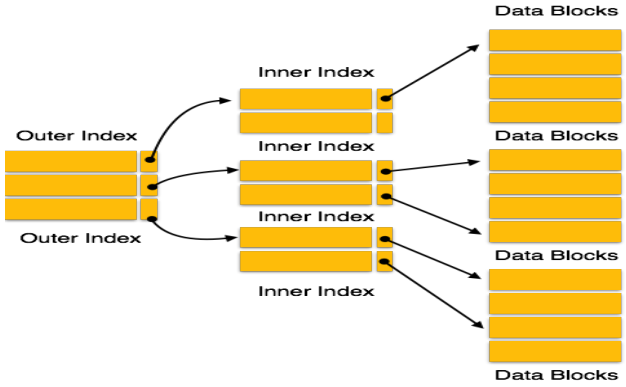
-





-

52. Types of Indexing
    a. Single level index
        - Only one index
        i. Primary index (Primary key + ordered)
        ii. Clustering index (Non key + ordered)
        iii. Secondary index (Non key/Candidate key + Unordered)
    b. Multilevel index
        - If index itself is too big, make its index too



-

        i. B trees
        ii. B^+ trees

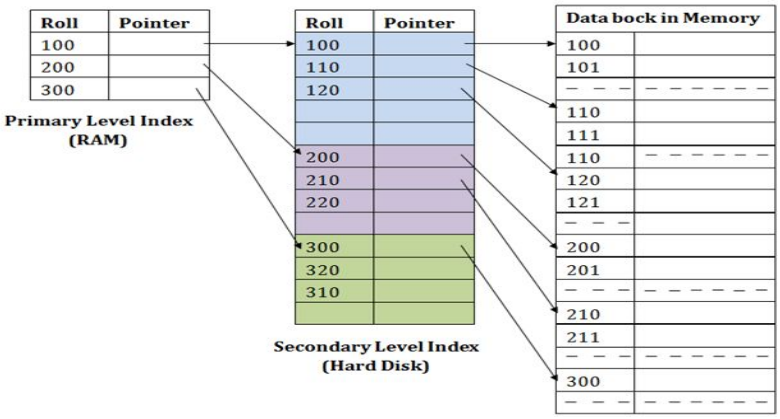
53. Primary indexing?
    a. A primary index is an ordered file whose records are of fixed length with two fields.
        - First field= same as Primary key of data file.

- Second field = a pointer to data where key is available

b. Primary index is defined on an ordered data file
c. The data file is ordered on a key field. The key field is generally the primary key of the relation.
d. Index created for first record of each block called block anchors.
e. Number of index entries = number of data blocks.
f. Average number of block access = log(B_i) + 1

   B_i = number of index blocks

   1 = Data access

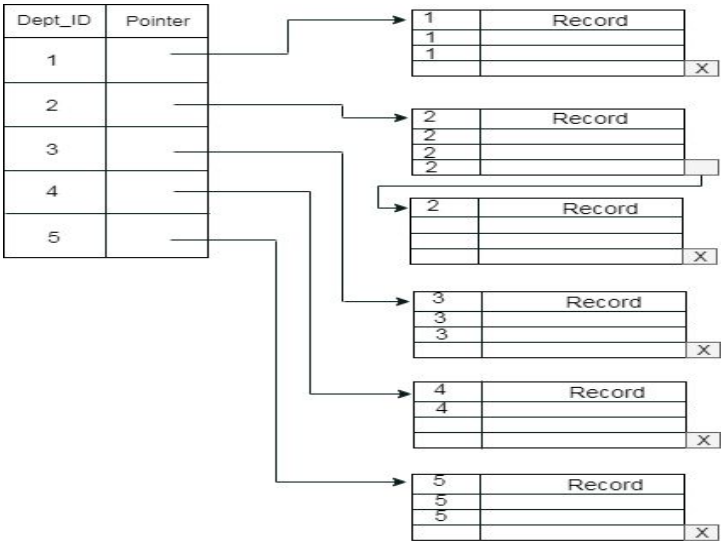


g.

h. Sparse indexing

54. Clustering index

a. Clustering index is defined on an ordered data file.
b. The data file is ordered on a non-key field which does not have distinct values for each record
c. Average number of block access >= log(B_i) + 1



d.

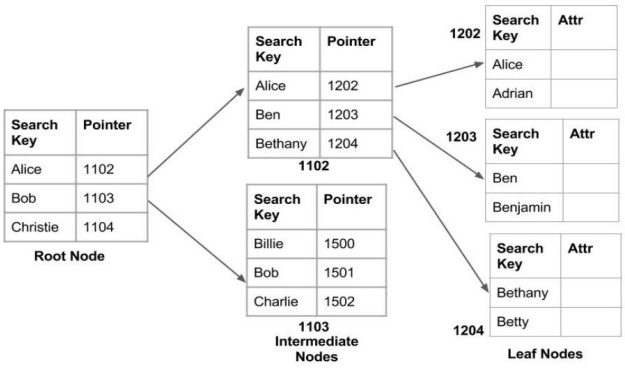
e. Both dense and sparse.

55. Secondary index

a. Secondary index provides a secondary means of accessing a file for which primary access already exist.

b. Index created for each record in a file

c. Number of index entries = number of records



d. **Non clustered index**

56. Multilevel indexing

a. Generalization of multilevel indexing

b. B and B+ trees are dynamic indexing

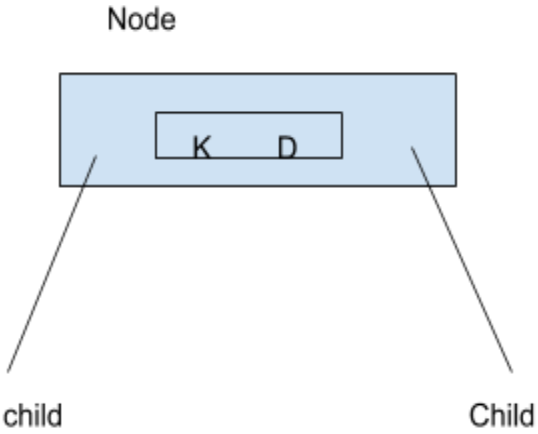
c. Node poniter

d.

57. B trees

a. B trees are balanced trees

b. In B trees at every level we are going to have key and data pointer, and that data pointer actually pointing to block or record



c.

d. Root
   - A root of B tree can have children between 2 and P
     - P= order of tree
     - Order of tree = maximum number of children a node can have
   - Minimum number of keys = 2-1=1
   - Maximum number of keys = P-1

e. Internal nodes
   - An internal node of B tree can have children between $\left(\frac{P}{2}\right)$ and P
   - Minimum number of keys = $\left(\frac{P}{2}\right)$ - 1
   - Maximum number of keys = P-1
   - Internal nodes are arranged as
     - $< P_1 < K_1, Pr_1 > P_2 < K_2, Pr_2 > P_3 < K_3, Pr_3 > .... P_{p-1} < K_{p-1},$
   - P = pointer (block pointer) (pointing to nodes of tree (children))
   - K = key is what you are searching for
   - Pr = pointer to data or record that is indicated by key
   - Index node points to next index level as well as (key, pointer) pair directly points to data file

f. Leaf nodes
   - Minimum number of keys = $\left(\frac{P}{2}\right)$ - 1

- Maximum number of keys = P-1
- All leafs are at same level i.e. distance of every tree from root is same
-

g. Overflow in B-tree
   - If number of search key values in a B-tree node exceed P-1
   - Might happen when we try to insert

h. Underflow in B-tree
   - Means node may b having less than minimum

i. B-tree search
   - Searching a binary tree is similar to BST , however rather than moving left or right at each node we need to perform a "P-way" search to see which subtree to probe.
     a. Children to left will have keys less than parent key.
     b. Say you start searching key K on a level in a node.It has keys k1, k2, k3… now if,
        i. K < k1 -> move to next level on left child node.
        ii. K > k1 -> move to right on next key
        iii. Compare K < k2 (then move to left child node of k2)
     c. No. of comparison is very less compared to linear search
     d. T(n) = height of tree

j. B-tree search algorithm

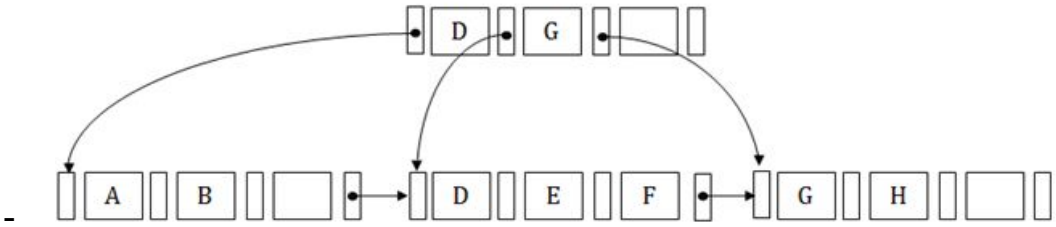
```
search_btree(tk, node, p) #tk=target_key
S=0                       # S tells where we are
while(S<P-1)                    # I want to check till
        If tk == node.key[S] then
                return node.ret[S];
        elseif tk<node.key[S] then
                break;
        else
                S++;
        end
```

end
if node.subtree[S] != null then
return search_btree(tk, node.subtree[S], p);
else return -1;

k. B-tree insertion
l. B-tree deletion
   - Borrow
   - merge

58. Difference between b-tree and b+tree
   - B-tree
      - in b-tree we are actually distributing the entire index records all over the tree which means at every ket the record pointer is also paired up with that i.e. along with ‘key’ , corresponding ‘record pointer’ will be there
   - B+tree
      - Having pointers is going to take alot of space and thus increase the depth of tree, thus we removed pointers and increased number of children

59. B+ trees
   a. The B+ tree is a balanced binary search tree. It follows a multi-level index format.
   b. In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
   c. Structure of B+ tree



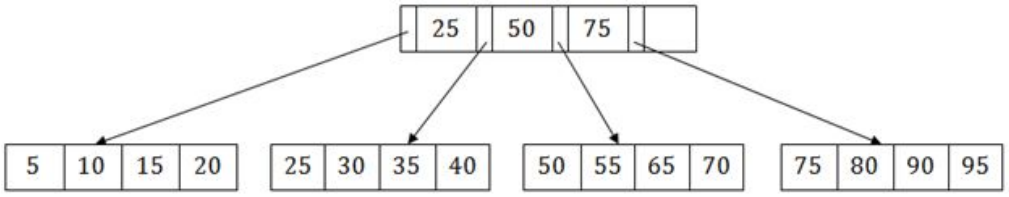
- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.
-

d. Internal node
   - An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
   - At most, an internal node of the tree contains n pointers.
   - An internal node of B tree can have children between $\left(\frac{P}{2}\right)$ and P
   - Minimum number of keys = $\left(\frac{P}{2}\right)$ - 1
   - Maximum number of keys = P-1
   -

e. Leaf node
   - The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
   - At most, a leaf node contains n record pointer and n key values.
   - Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.
   - (n, n+1) (keys, pointers)
   - $< \ < K_1, Pr_1 > \ < K_2, Pr_2 > \ < K_3, Pr_3 > .... \ < K_{p-1}, Pr_{p-1} >, \ P_{next} >$
   - Order of non-leaf > order of leaf because in leaf, pointer is going to consume some space
   - order(non-leaf) = max. no. of children i.e. node pointers
   - order(leaf) = max. no. of pairs

f. In b+trees we have 2 types of orders. In case both are using node pointers without using record pointers instead of record pointers, then same o(leaf)=o(non-leaf)

g. Searching a record in a B+ tree
   - Read from notes
   - Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.
   - So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.
   -



h. .

60.h