Question Paper Code: 37783 S.E.(Information Technology Engineering)(SEM-III)

(Choice Base Credit Grading System)(R2016) DBMS Date of Exam: 2nd June 2018

Paper Solution

Q 1] a Explain the Role of DBA ?

Five main functions of a database administrator are:

- To create the scheme definition
- To define the storage structure and access methods
- To modify the scheme and/or physical organization when necessary
- To grant authorization for data access
- To specify integrity constraints

Q1] b List all the functional dependencies satisfied by the relation

Ans. When a column (entity) can be fully determined by another column, then it is functional dependent on that column.

Following are all functional dependencies which are possible from above relation.

$X \rightarrow Y$

$X \rightarrow Z$

$Y \rightarrow X$

$Y \rightarrow Z$

$Z \rightarrow X$

$Z \rightarrow Y$

$XY \rightarrow Z$

$YZ \rightarrow X$

$XZ \rightarrow Y$

Out of these functional dependencies, following are the functional dependencies satisfied by the relation.

$Y \rightarrow Z$

$X \rightarrow Z$

$XY \rightarrow Z$

Q 1] c What is the difference between Unique Key and Primary Key

Primary Key:

There can only be one primary key in a table

In some DBMS it cannot be NULL - e.g. MySQL adds NOT NULL

Primary Key is a unique key identifier of the record

Unique Key:

Can be more than one unique key in one table

Unique key can have NULL values

It can be a candidate key

Unique key can be NULL and may not be unique

Q 1 d Explain different types of attributes with examples?

Simple attribute – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

Composite attribute – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

Derived attribute – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

Single-value attribute – Single-value attributes contain single value. For example – Social_Security_Number.

Multi-value attribute – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Q 2 a Explain Static hashing technique with example

HASHING TECHNIQUES:

Hashing provides very fast access to records on certain search conditions. This organization is usually called a hash file.
The search condition must be an equality condition on a single field, called the hash field of the file. The hash field is also called as hash key.

The idea behind hashing is to provide a function 'h' called a **hash function (or) randomizing function,** that is applied to the hash field value of a record and yields the address of the disk block in which the record is stored.

Hashing is also used as an internal search within a program whenever a group of records is accessed or exclusively by using the value of one field.

Static Hashing

A bucket is a unit of storage containing one or more records (a bucket is typically a disk block).
The file blocks are divided into M equal-sized buckets, numbered bucket0, bucket1... bucketM-1.Typically, a bucket corresponds to one (or a fixed number of) disk block.

In a hash file organization we obtain the bucket of a record directly from its search-key value using a hash function, h (K).

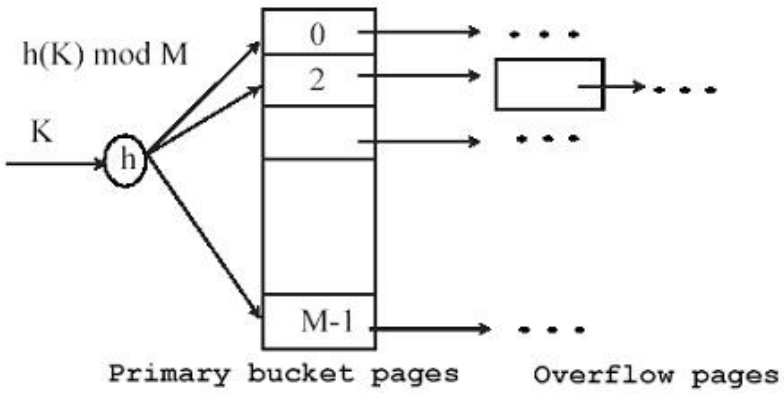
The record with hash key value K is stored in bucket, where i=h(K).

Hash function is used to locate records for access, insertion as well as deletion.

Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.

h(*K*) mod M = bucket to which data entry with key *k* belongs. (M = # of buckets)



Static External Hashing

One of the file fields is designated to be the hash key, K, of the file.
Collisions occur when a new record hashes to a bucket that is already full.

An overflow file is kept for storing such records. Overflow records that hash to each bucket can be linked together.

To reduce overflow records, a hash file is typically kept 70-80% full.

The hash function h should distribute the records uniformly among the buckets; otherwise, search time will be increased because many overflow records will exist.

Static Hashing (Contd.)

Hash function works on *search key* field of record *r.* Must distribute values over range 0 ... M-1.

H (*K*) = (a * *K* + b) usually works well.
a and b are constants;
lots known a but how to tune **h**.

Typical hash functions perform computation on the internal binary representation of the search-key.

For example, for a string search-key, the binary representations of all the characters in the string could be added and the sum modulo the number of buckets could be returned. .

Ideal hash function is **random**, so each bucket will have the same number of records assigned to it irrespective of the *actual distribution* of search-key values in the file.

Q 2 b Define Normalization ? Explain 1NF, 2NF and 3NF with examples ?

Normalization

Normalization :- Decomposing tables to eliminate data redundancy

To avoid
- Insertion / Deletion / Update Anamolies

1 NF : Every Attribute should have Atomic (Single) Value

Eg ① R →

E_id	E_Phone
ABC	123, 256

1NF ↓

E_id	E_Phone
ABC	123
ABC	256

2 Nf → Should be in 1 Nf
→ there must not be any partial dependency of any column on primary key

Eg ② R →

E_id	E_name	P_id	P_name	P_loc
A	B	C	D	E

A → B A determines B

C → DE C determines D & E

∴ R_1 | E_id | E_name |

R_2 | P_id | P_name | P_Loc |

3 Nf :- Should be in 1 Nf & 2 NF

→ Transitive functional dependency Should be Removed

Eg ③ R →

P_id	P_name	P_Loc
A	B	C

A → B & A → C

but B → C (Transitive dependency)

∴ R_1 | P_id | P_name |

R_2 | P_id | P_Loc |

BCNf :- Should be in 1, 2, 3 Nf
→ Every determining attribute should be a candidate key

R →

Eg ④

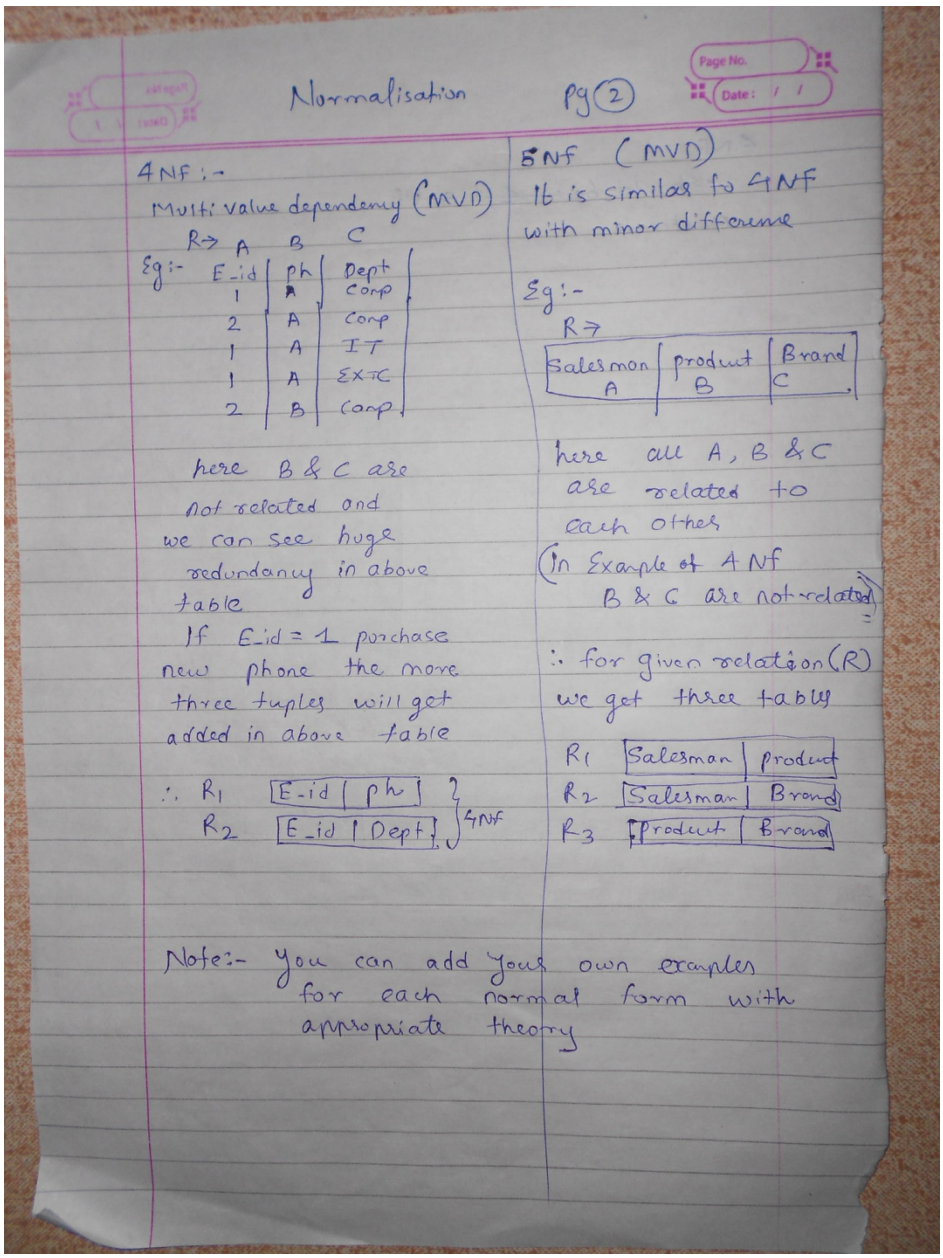
Stud_id	Subject	Teacher
A	B	C

In above Example A → B & C (A: Candidate Key)

but if we know name of subject then we can find respective teachers for that subject ∴ B → C

As per BCNf, B should be candidate key

BCNf ⟹ ∴ R_1 → | Stud_id | Subject | R_2 → | Subject | Teacher |



Q3. a Consider the following employee database.
Employee(empname, street, city, date_of_joining)
Works(empname, company_name, salary)
Company(company_name, city)
Manages(empname, manager_name)
Write SQL queries for the following statements:

i) Modify the database so that employee “Sachin ” now lives in “Mumbai”
Ans. update Employee set city =’Mumbai’ where empname = ‘Sachin’;

ii) Find number of employees in each city with date_of_joining as “01-Aug-2017”
Ans. select count(*) from Employee where date_of_joining = ’01-Aug-2017’ groupby city;
or

Ans. elect count(*) from Employee where date_of_joining = ’01-Aug-2017’ groupby city;

iii)List the name of companies starting with letter “A”

Ans. select company_name from Company where company_name like ‘A%’;

iv)

Display empname , manager_name , city of those employees whose date_of_joining is greater than “01-01-2014”

Ans. select empname , manager_name , city from Employee, Manages where Employee.empname = Manages.empname and Employee.date_of_joining=’ 01-01-2014’;

Q3.b Explain DBMS architecture

Ans. The architecture of a database system is greatly influenced by the underlying computer system on which the database is running:

i. Centralized.

ii. Client-server.

iii. Parallel (multi-processor).

iv. Distributed

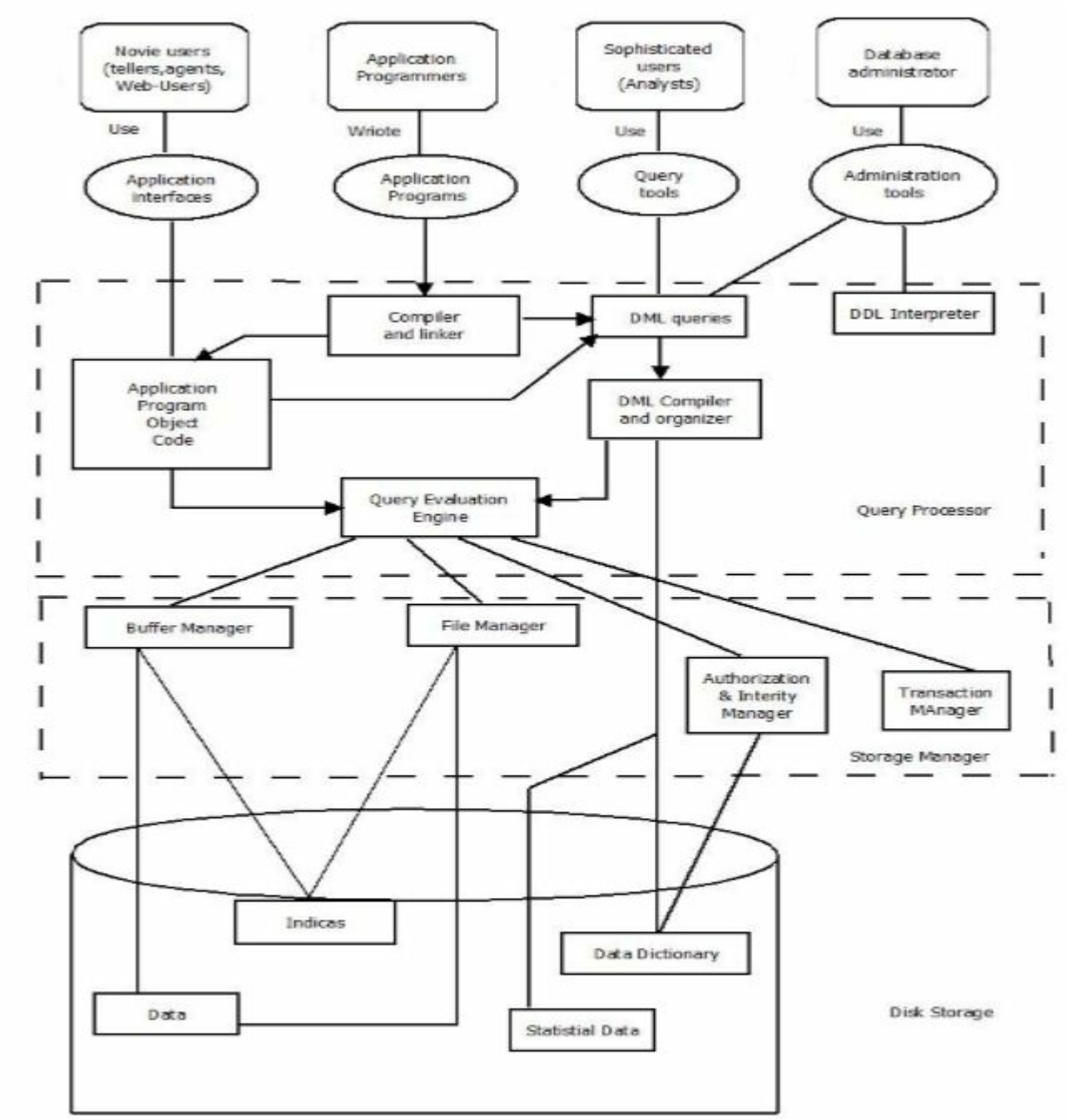


Fig1. Database System Architecture.

Database Users:

Users are differentiated by the way they expect to interact with the system:

Application programmers:

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.

Sophisticated users:

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands.

Specialized users :

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Naïve users :

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
For example, a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

Database Administrator:

Coordinates all the activities of the database system. The database administrator has a good understanding of the enterprise's information resources and needs.
Database administrator's duties include:

Schema definition: The DBA creates the original database schema by executing a set of data definition statements in the DDL.

Storage structure and access method definition.

Schema and physical organization modification: The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Granting user authority to access the database: By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.

Specifying integrity constraints.

Monitoring performance and responding to changes in

requirements.

Query Processor:

The query processor will accept query from user and solves it by accessing the database.

Parts of Query processor:

DDL interpreter

This will interprets DDL statements and fetch the definitions in the data dictionary.

DML compiler

a. This will translates DML statements in a query language into low level instructions that the query evaluation engine understands.

b. A query can usually be translated into any of a number of alternative evaluation plans for same query result DML compiler will select best plan for query optimization.

Query evaluation engine

This engine will execute low-level instructions generated by the DML compiler on DBMS.

Storage Manager/Storage Management:

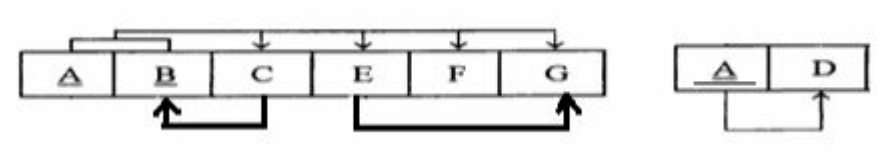
A storage manager is a program module which acts like interface between the data stored in a database and the application programs and queries submitted to the system.
Thus, the storage manager is responsible for storing, retrieving and updating data in the database.

The storage manager components include:

Authorization and integrity manager: Checks for integrity constraints and authority of users to access data.
Transaction manager: Ensures that the database remains in a consistent state although there are system failures.
File manager: Manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
Buffer manager: It is responsible for retrieving data from disk storage into main memory. It enables the database to handle data sizes that are much larger than the size of main memory.
Data structures implemented by storage manager.
Data files: Stored in the database itself.
Data dictionary: Stores metadata about the structure of the database.
Indices: Provide fast access to data items.

Q4 a. Construct a dependency diagram of relation R and normalize it up to the BCNF Normal form

Ans.
Step 1: The relation R is already in 1NF, since all its values are atomic.
Step 2: Converting into 2NF form. A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key



Step 3: Converting into 3NF. A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key

Therefore the Normalized relation are

AB --> CFG

AB--> CFE

E--> G

C--> B

A--> D

This is the final solution in 3NF.

Step 4: For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

It should be in the **Third Normal Form**

And, for any dependency A → B, A should be a **super key**

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**. The above solution is in BCNF since no non key attribute determines another non key attribute.

Q4.b Explain different types of relational algebra operations.

Ans. Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their

input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , > , ≤.

For example

$\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A_1, A_2, A_n}(r)$

Where A_1, A_2, A_n are attribute names of relation **r**

Duplicate rows are automatically eliminated, as relation is a set.

For example

$\prod_{subject, author}(Books)$

Selects and projects columns named as subject and author from the relation Books.

Union Operation ( )

It performs binary union between two given relations and is defined as −

r s = { t | t r or t s}

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

r, and **s** must have the same number of attributes.
Attribute domains must be compatible.
Duplicate tuples are automatically eliminated.

∏ author (Books) ∏ author (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation r s

Finds all the tuples that are present in **r** but not in **s**

∏ author (Books) − ∏ author (Articles)

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q r and t s}

σauthor = 'tutorialspoint'(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ

Notation $\rho_x(E)$

Where the result of expression **E** is saved with name of **x**

Additional operations are −

- Set intersection
- Assignment
- Natural join

Q5.a Explain Cursors and its types with example?

Ans. A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

S.No Attribute & Description 1

%FOUND

Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

%NOTFOUND

The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

%ISOPEN

Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

%ROWCOUNT

Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

Example

We will be using the CUSTOMERS table we had created and used in the previous chapters.

```
Select * from customers;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected −

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
```

```
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
6 customers selected

PL/SQL procedure successfully completed.
```

If you check the records in customers table, you will find that the rows have been updated −

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for

fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
   c_id customers.id%type;
   c_name customerS.No.ame%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

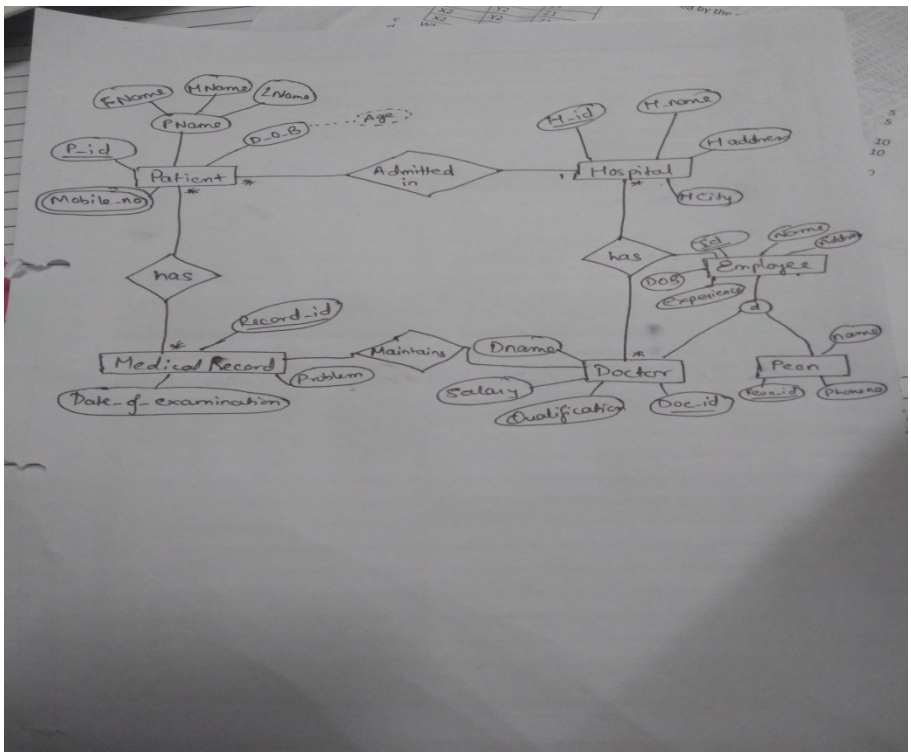
When the above code is executed at the SQL prompt, it produces the following result −

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

Q5. b Draw EER diagram for Hospital Management System showing constraints on generalisation and specialisation

Ans.



Q 6] a

Types of Entities

Ans. Entity –

An entity is an object that are represented in the database. For example Mohit, Vasu, CSE306 etc.

An entity is represented or defined by set of attributes. Attributes are the properties used to describe an entity. For example, a STUDENT entity may have a Name, Roll number, Class, Marks etc. where STUDENT is the entity and name roll number class marks are the attributes.

Basic Types of Entity –

Strong Entity Types

Weak Entity Types

Strong Entity Type – are the entities which has a key attribute in its attribute list or a set that has a primary key. The strong entity type is also called regular entity type. For Example,

Strong Entity Type(Entity and its Types)

The Student's unique RollNo will identify the students. So, RollNo is set to be the Primary Key of the STUDENT entity, & Hence STUDENT is a strong entity type because of its key attribute.

Q 6] b

Authorization in SQL

Ans. Authorization is finding out if the person, once identified, is permitted to have the resource.
Authorization explains that what you can do and is handled through the DBMS unless external security procedures are available.

Database management system allows DBA to give different access rights to the users as per their requirements.

Basic Authorization we can use any one form or combination of the following basic forms of authorizations

i. Resource authorization:-Authorization to access any system resource. e.g. sharing of database, printer etc.

ii. Alternation Authorization:- Authorization to add attributes or delete attributes from relations

iii. Drop Authorization:-Authorization to drop a relation.

Granting of privileges:

i. A system privilege is the right to perform a particular action,or to perform an action on any schema objects of a particular type.

ii. An authorized user may pass on this authorization to other users.This process is called as ganting of privileges.

iii. Syntax:

```
GRANT <privilege list>
ON<relation name or view name>
TO<user/role list>
```

iv. Example:

The following grant statement grants user U1,U2 and U3 the select privilege on Emp_Salary relation:

```
GRANT  select
ON Emp_Salary
TO U1,U2 and U3.
```

Revoking of privileges:

i. We can reject the privileges given to particular user with help of revoke statement.

ii. To revoke an authorization, we use the revoke statement.

iii. Syntax:

```
REVOKE <privilege list>
ON<relation name or view name>
FROM <user/role list>[restrict/cascade]
```

iv. Example:

The revocation of privileges from user or role may cause other user or roles also have to loose that privileges.This behavior is called cascading of the revoke.

```
Revoke select
ON  Emp_Salary
FROM U1,U2,U3.
```

Some other types of Privileges:

i. **Reference privileges:**

SQL permits a user to declare foreign keys while creating relations.

Example: Allow user U1 to create relation that references key 'Eid' of Emp_Salary relation.

```
GRANT REFERENCES(Eid)
ON Emp_Salary
TO U1
```

ii. **Execute privileges:**

This privileges authorizes a user to execute a function or procedure.

Thus,only user who has execute privilege on a function Create_Acc() can call function.

```
GRANT EXECUTE
ON Create_Acc
TO U1.
```

Q 6 c Views in SQL

Ans. SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Q 6 d

B Tree

Ans. B-Tree Index Files

B-tree indices are similar to B $^{+}$ -tree indices.

Difference is that B-tree eliminates the redundant storage of search key values.

In B $^{+}$ -tree of Figure 11.11, some search key values appear twice.

A corresponding B-tree of Figure 11.18 allows search key values to appear only once.

Thus we can store the index in less space.

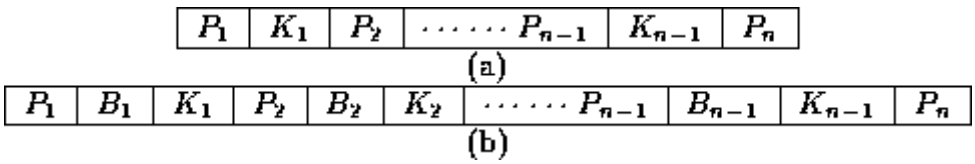


Figure 11.8: Leaf and nonleaf node of a B-tree.

Advantages:

Lack of redundant storage (but only marginally different).

Some searches are faster (key may be in non-leaf node).

Disadvantages:

Leaf and non-leaf nodes are of different size (complicates storage)

Deletion may occur in a non-leaf node (more complicated)

Generally, the structural simplicity of B $^{+}$ -tree is preferred.