SQL

* SQL stands for Structured Query Language.
* SQL is a standard language for storing, manipulating and retrieving data in databases.
* SQL commands are classified into five types. They are

1. DDL commands
2. DML commands
3. DCL commands
4. TCL commands
5. DQL commands.

DDL → Data Definition Language

DML → Data Manipulation Language

DCL → Data Control Language

TCL → Transaction Contral Language.

DQL → Data Query Language.

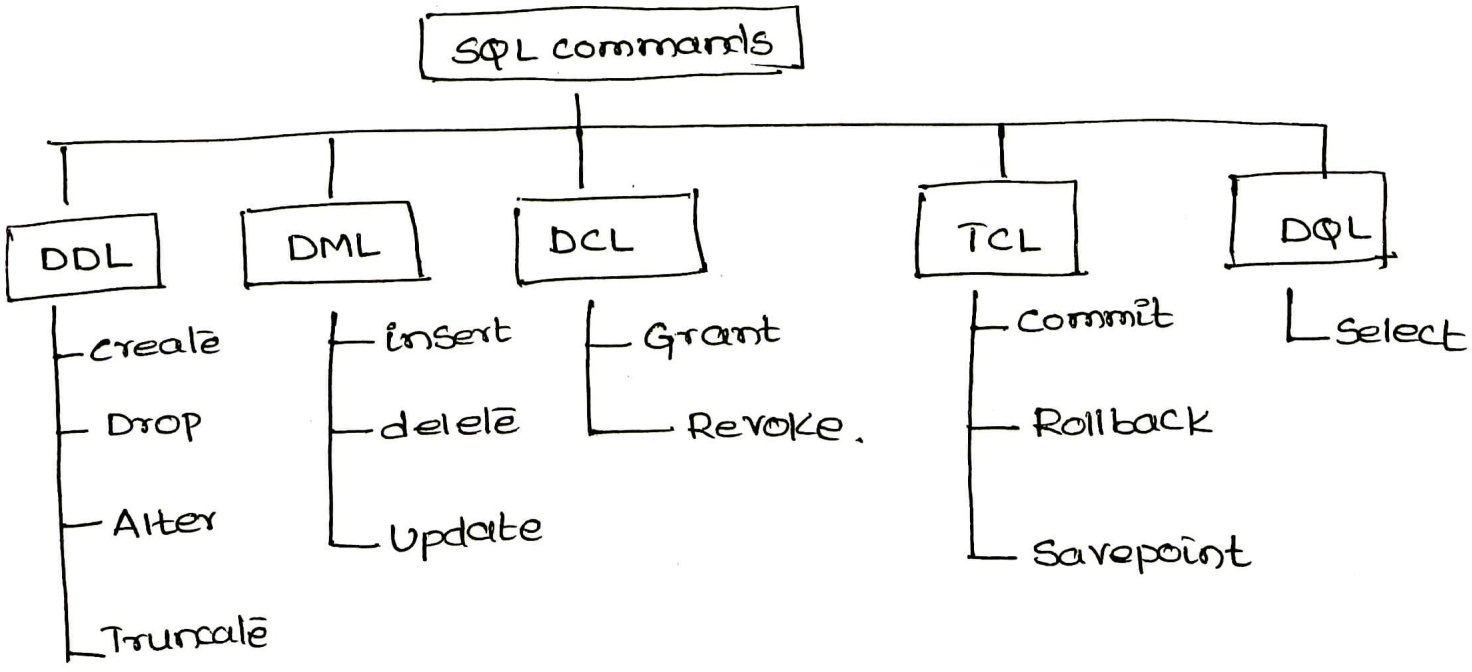


Table :- A table is an arrangement of data in the form of rows and columns.

CREATING A TABLE

The syntax for creating a table is

```
CREATE TABLE <TABLENAME>
(
<COLUMN-1> DATATYPE,
<COLUMN-2> DATATYPE,
    :
<COLUMN-N> DATATYPE
);
```

E.g.

```
CREATE TABLE Student
(
  Rollno    Number,
  Name      Varchar2(20),
  Marks     Number
);
```

INSERTING A RECORD INTO TABLE

The syntax for inserting a record in a table is

```
INSERT INTO <TABLENAME> VALUES(VALUE-1, VALUE-2
                                ... VALUE-N);
```

E.g. `INSERT INTO Student values(1001, 'Suresh', 75);`

INSERT INTO student VALUES (&Rollno, '&Name', &Marks);

This command is used to take input from the keyboard and store it in a table.

UPDATING A RECORD IN A TABLE

The syntax for updating a record in a table is.

UPDATE <TABLENAME> SET <COLUMNNAME>= VALUE

WHERE <COLUMNNAME>= VALUE.

E.g. Update Student set Marks = 80 where Rollno = 1001;

DELETING A RECORD FROM A TABLE

The syntax for deleting a record from a table is.

DELETE FROM <TABLENAME> WHERE <COLUMNNAME>= VALUE;

E.g. Delete from student where Rollno = 1001;

DELETING ALL RECORDS FROM A TABLE

DELETE * FROM <TABLENAME>;

E.g. Delete * from student;

RETRIEVING ALL RECORDS FROM A TABLE

The Select command is used to display all the records in a table.

The Syntax for displaying all records in a table is.

SELECT * FROM <TABLENAME>;

E.g. Select * from student;

RETRIEVING A SPECIFIC RECORD FROM A TABLE

The syntax for retrieving a specific record in a table is

SELECT * FROM <TABLENAME> WHERE <COLUMNNAME> = VALUE;

E.g. Select * from student where rollno = 1001;

ADDING COLUMN TO A TABLE

The alter table command is used to add a column to a table.

Syntax for adding a column to a table is

ALTER TABLE <TABLENAME> ADD <COLUMNNAME> DATATYPE;

E.g. ALTER TABLE Student ADD address Varchar2(20);

DROPPING A COLUMN FROM A TABLE

The syntax for dropping a column from a table is

ALTER TABLE <TABLENAME> DROP COLUMN <COLUMNNAME>

E.g. ALTER TABLE Student DROP COLUMN address;

MODIFYING THE SIZE OF A COLUMN IN A TABLE

The syntax for modifying the size of a column in a table is

ALTER TABLE <TABLENAME> MODIFY <COLUMNNAME> DATATYPE(NEW SIZE);

E.g. ALTER TABLE Student MODIFY address Varchar2(40)

ADDING A PRIMARY KEY

Syntax

ALTER TABLE <TABLENAME> ADD CONSTRAINT <CONSTRAINT NAME> PRIMARY KEY (COLUMN NAME);

E.g.

ALTER TABLE Student ADD CONSTRAINT PK_Student Primary key (Rollno);

DROPPING A PRIMARY KEY

Syntax :-

ALTER TABLE <TABLENAME> DROP PRIMARY KEY;-

E.g.

ALTER TABLE Student DROP PRIMARY KEY;

DROPPING A TABLE

The syntax for dropping a table is.

```
DROP TABLE <TABLENAME>;
```

E.g. DROP TABLE Student;

DIFFERENCE BETWEEN DROP AND TRUNCATE

The DROP command is used to remove the whole database, table, index etc. The structure of the tabledoes not exist.

The TRUNCATE command is used to remove all the rows from the table. The structure of the table exists

GRANT COMMAND

It is used to provide access or privileges on the database objects to the users.

Syntax

```
GRANT   < PRIVILEGE-NAME >
ON      < OBJECT-NAME >
TO      < USER-NAME >;
```

E.g.

```
GRANT   SELECT
ON      STUDENT
TO      KIRAN;
```

REVOKE COMMAND

It removes user access rights or privileges on the database objects.

Syntax :-

```
REVOKE   < PRIVILEGE_NAME >
ON       <OBJECT_NAME>
FROM     <USER_NAME> ;
```

E.g.

```
REVOKE   CREATE TABLE
FROM     KIRAN ;
```

COMMIT COMMAND :-

The commit command is used to save changes invoked by a transaction to the database.

E.g. COMMIT

ROLLBACK COMMAND :-

The rollback command is used to undo transactions that have not been saved to the database.

E.g. ROLLBACT

SAVE POINT :-

It is used to create a new save point.

Syntax :-

```
SAVEPOINT <Savepoint name> ;
```

GROUP BY clause

It is used to arrange similar data into group.

Syntax :-

```
SELECT  COLUMN-1 , COLUMN-2
FROM    <TABLENAME>
GROUP BY  COLUMN-1 ;
```

E.g.

```
SELECT  SUM ( SAL )
FROM    EMPLOYEE
GROUP BY  DEPTNO ;
```

ORDER BY clause

It is used to sort the data in a column either in ascending order or descending order.

Syntax :-

```
SELECT  COLUMN-1
FROM    <TABLE NAME>
ORDER BY  COLUMN-1 ;
```

E.g.

```
SELECT  empname
FROM    EMPLOYEE
ORDER BY empname ;
```

Employee.

EmpNo	EmpName	EmpSal
1001	Suresh	10,000
1002	vishnu	20,000
1003	vijay	30,000
1004	Avinash	40,000

1. Select sum(EmpSal) from Employee;

   1,00,000

2. Select min(EmpSal) from Employee;

   10,000

3. Select max(EmpSal) from Employee;

   40,000

4. Select avg(EmpSal) from Employee;

   25,000

5. Select count(*) from Employee;

   4

Set Operators

Set operators are used to combine the results obtained from two or more queries into a single result.

There are four set operators. They are

* Union
* Union all
* Intersect
* Minus.

Union :- It combines distinct results of two or more Select statements.

Union All :- It combines all results of two or more Select statements including duplicates.

Intersect :- It returns only the common records obtained from two or more select statements.

Minus :- It returns only those records which are exclusive to the first table.

E.g. Consider two tables customer_jan and customer_feb given below.

customer_jan

cust_id	cust_name	cust_city
1001	Dinesh	Bengaluru
1002	Hemanth	Hyderabad
1003	Suresh	Bengaluru
1004	vishnu	Hyderabad

customer_feb

cust_id	cust_name	cust_city
1004	vishnu	Hyderabad
1005	Akash	chennai
1006	Sumanth	Chennai

(Select * from customer_jan) union (select * from customer_feb);

cust_id	cust_name	cust_city
1001	Dinesh	Bengaluru
1002	Hemanth	Hyderabad
1003	Suresh	Bengaluru
1004	vishnu	Hyderabad
1005	Akash	chennai
1006	Sumanth	Chennai

(Select * from customer_jan) Union all (select * from customer_feb);

cust_id	cust_name	cust_city
1001	Dinesh	Bengaluru
1002	Hemanth	Hyderabad
1003	Suresh	Bengaluru
1004	Vishnu	Hyderabad.
1004	Vishnu	Hyderabad
1005	Akash	chennai
1006	Sumanth	Chennai

(Select * from customer_jan) intersect (select * from customer_feb);

cust_id	cust_name	cust_city
1004	Vishnu	Hyderabad

(Select * from customer_jan) minus (select * from customer_feb);

cust_id	cust_name	cust_city.
1001	Dinesh	Bengaluru
1002	Hemanth	Hyderabad
1003	Suresh	Bengaluru.

View

A view is a virtual table based on the resultset of an SQL statement.

A view is created with the CREATE VIEW statement.

Syntax :-

```
CREATE VIEW <VIEW_NAME> AS
SELECT COLUMN1, COLUMN-2
FROM <TABLENAME>
WHERE CONDITION
```

E.g.

```
CREATE VIEW V1 AS
SELECT CNAME, CNO
FROM CUSTOMER
WHERE country = 'India';
```

Displaying the records in a view

E.g. Select * from V1;

A view can be updated with the CREATE OR REPLACE VIEW statement.

Dropping a view

A view is deleted with the DROP VIEW statement.

E.g. DROP VIEW V1;

Types of Views

Views are of two types. They are:

1. Simple view
2. Complex view

Simple view	Complex view
1. It is created from only one table	1. It is created from more than one table.
2. We cannot use group functions like max(), count() etc.	2. We can use group functions like min(), max(), avg() sum() etc.
3. DML operations can be performed	3. DML operations cannot be performed.
4. It cannot contain not null columns from base tables.	4. It can contain not null columns from base tables.

Joins

A Join is a binary operation which allows to combine cartesian product and selection in one single statement.

The purpose of creating a join is to combine the data from two or more tables.

There are mainly two types of joins. They are

1. Inner Join
2. Outer Join.

Inner Join is classified into three types.

1. Theta Join
2. Equi Join
3. Natural Join.

Outer Join is classified into three types.

1. Left outer Join
2. Right outer Join.
3. Full outer Join.

Theta Join :- It allows us to join two tables based on the condition represented by theta.

It is denoted by the symbol $\bowtie_\theta$

It work for all comparison operators

$<$, $>$, $\leq$, $\geq$ etc.

Equi Join :- If the operator used in theta join is restricted to only equality operator then it is known as Equi join

Natural Join :-

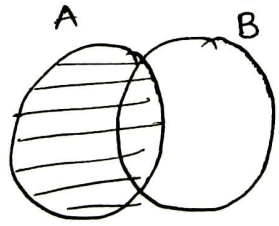
In this type of join, there should be atleast one common attribute between two relations.

The attributes should have the same name and domain.

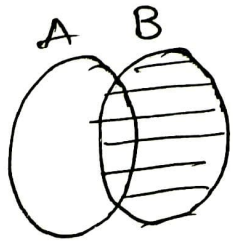
It is a variation of Equii Join

Outer Join :- In this type of join, the table retains each record even if no other matching record exists.

Left Outer Join :- It returns all the records from the table on the left even if no matching records have been found in the table on the right. When no matching record is found in the table on the right, NULL is returned.



Right Outer Join :- It returns all the records from the table on the right even if no matching records have been found in the table on the left. When no matching record is found in the table on the left, NULL is returned.



Left outer Join :-

Select * from customer left outer join order
on customer.custid = order.custid ;

custid	custname	custaddress	custcity	orderid	orderitem	orderprice	custid
1001	Dinesh.	xyz	bengaluru	701	pizza	350	1001
1001	Dinesh	xyz	bengaluru	702	burger	250	1001
1003	Akash	mno	bengaluru	703	Honey cake	450	1003
1002	Hemanth	pqr	bengaluru				
1004	Sumanth.	abc	bengaluru				

Right Outer Join

Select * from customer right outer join order on customer.cid = order.custid ;

custid	custname	custaddress	custcity	orderid	orderitem	order price	custid
1001	Dinesh	xyz	bengaluru	401	pizza	250	1001
1001	Dinesh	xyz	bengaluru	402	Burger	350	1001
1003	akash	mno	bengaluru	403	Honeycake	450	1003
				404	plumcake	550	1005

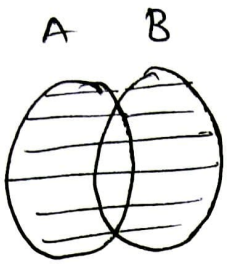
Full Outer Join

```
select * from customer full outer join order
on customer.custid = order.custid;
```

custid	custname	custaddress	custcity	orderid	orderitem	orderprice	custid.
1001	Dinesh	xyz	bengaluru	701	Pizza	350	1001
1001	Dinesh	xyz	bengaluru	702	Burger	250	1001
1003	akash	mmm	bengaluru	703	honeycake	450	1003
1002	hemanth	pqr	bengaluru				
1004	sumanth	abc	bengaluru	704	plumcake	550	1005

Full Outer Join :-

In a Full Outer Join, all records from both relations are included in the result, irrespective of the matching condition.



Customer

custid	custname	custaddress	custcity
1001	Dinesh	xyz	Bengaluru
1002	Hemanth	pqr	Bengaluru
1003	Akash	mno	Bengaluru
1004	Sumanth	abc	Bengaluru

Order

orderid	orderitem	orderprice	custid
701	pizza	350	1001
702	Burger	250	1001
703	Honeycake	450	1003
704	plumcake	550	1005

Natural Join

```
Select * from customer inner join order
on customer.custid = order.custid ;
```

custid	custname	custaddress	Custcity	orderid	orderitem	order price	custid
1001	Dinesh	xyz	bengaluru	701	Pizza	350	1001
1001	Dinesh	xyz	bengaluru	702	Burger	250	1001
1003	Akash	mno	bengaluru	703	Honeycake	450	1003

String functions in SQL.

S.No	Function	Description
1.	char_length( )	Returns the number of characters in the given String
2.	concat( )	Returns the concatenation of two strings.
3.	lcase( )	Returns the lowercase characters of the given String
4.	Ucase( )	Returns the given string in uppercase.
5.	lower( )	Returns the given string in lowercase
6.	upper( )	Returns the given string in upper case.
7.	ltrim( )	Returns the string after removing leading spaces in the given string
8.	rtrim( )	Returns the string after removing trailing spaces in the given string.

S.NO	function	Description.
9.	trim( )	Returns the string after removing leading and trailing
10.	reverse( )	Returns the reverse of the given string
11.	strcmp( )	Compares two strings and returns either -1, +1 & 0.
12.	instr( )	Returns the position of the first occurrence of the substring in mainstring.
13.	locate( )	Returns the position of the first occurrence of the substring in mainstring.
14.	bin( )	Returns a ~~binary~~ String representation of the binary value.
15.	oct( )	Returns a string representation of the octal value.
16.	hex( )	Returns a string representation of the hexadecimal value.

Recursive Queries

Recursion is implemented in SQL using common table expressions (CTE).

DB2, Microsoft SQL Server, Oracle and PostgreSQL support recursive queries using CTEs.

CTE

* CTE stands for Common Table Expression.

  It is a temporary named result set that can be referenced within a SELECT, INSERT, UPDATE or DELETE statement.

(or)

* A CTE is a named temporary result set that exists within the scope of a single statement and that can be referred to later within that statement, possibly multiple times.

To specify common table expressions, use a WITH clause that has one or more comma-separated subclauses.

E.g.

```
WITH
    cte1 as (select a, b from table1),
    cte2 as (select c, d from table2)
Select b, d from cte1 join cte2
Where cte1.a = cte2.c
```

Recursive Common Table Expressions

A recursive common table expression is one having a subquery that refers to its own name.

```
WITH RECURSIVE cte(n) AS
(
    SELECT 1
    UNION ALL
    SELECT n+1 FROM cte WHERE n<5
)

Select * from cte;
```

Output

n
1
2
3
4
5

The recursive select part must not contain these constructs.

1. AGGREGATE FUNCTION(SUM)
2. GROUP BY
3. ORDER BY
4. DISTINCT

Triggers

A trigger defines a set of actions that are performed when an event occurs in a database.

Advantages

* Enforcing complex business rules that cannot be established using integrity constraints
* preventing invalid transactions
* Gathering statistical information on table accesses.
* Generating value automatically for derived columns
* Auditing sensitive data.

Types of Triggers

Triggers can be classified into three types.

1. Level Triggers
2. Event Triggers
3. Timing Triggers.

Level Triggers :-

They are of two types. They are

1. Row level triggers
2. Statement level triggers.

Row level triggers

* It fires for every record that got affected with the execution of DML statements like INSERT, DELETE AND UPDATE.
* It always use a FOR EACH ROW clause in a triggering statement

Statement level triggers

* It fires once for each statement that is executed.

Event Triggers

There are three types of event triggers.

1. DDL Event trigger

   It fires with the execution of every DDL statement (CREATE, ALTER, DROP, TRUNCATE)

2. DML Event trigger

   It fires with the execution of every DML statement (INSERT, DELETE, UPDATE)

3. Database event trigger

   It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

Timing triggers

There are two types of timing triggers.

1. Before trigger

   It fires before executing DML statement.

2. After trigger

   It fires after executing DML statement

Syntax for creating a trigger

```
CREATE OR REPLACE TRIGGER <trigger_name>
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE ON <table_name>
REFERENCING ( OLD AS O, NEW AS N)
FOR EACH ROW WHEN ( test_condition)
DECLARE
    -- variable declaration;
BEGIN
    -- Executable statements;
EXCEPTION
    -- Error handling statements
END <trigger_name>;
END;
```

E.g.

```
CREATE OR REPLACE TRIGGER CheckAge
BEFORE
INSERT OR UPDATE ON Student
FOR EACH ROW
BEGIN
    IF :new.age > 30 THEN
        raise_application_error(-20001, 'Age should not
                                be greater than 30');
    END IF;
END;
```

Embedded SQL

The SQL standard defines embeddings of SQL in a variety of programming languages such as C, Java etc.

A language in which SQL queries are embedded is referred to as a host language.

An embedded sql program must be processed by a special preprocessor prior to compilation.

The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow run-time execution of the database accesses.

'EXEC SQL' statement is used to identify embedded SQL statements.

```
EXEC SQL

    Statement-1;

    .
    .
    .

    Statement-n;

END-EXEC
```

Before executing any SQL statements, the program must first connect to the database.

This is done using

```
EXEC SQL connect to server User User-name
END-EXEC
```

Here

Server identifies the server to which a connection is to be established.

User-name refers to the user name through which a user is connected.

A database modification request takes the form

```
EXEC SQL
    < insert, delele or update >
END-EXEC.
```

Variables can be declared in the host-language using the following syntax.

```
EXEC SQL BEGIN DECLARE SECTION
    .
    .
    .
EXES SQL ENID DECLARE SECTON.
```

E.g.

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION.
```

Exceptions

The SQL-92 Standard recognizes two special variables for reporting errors.

1. SQLCODE
2. SQLSTATE

SQLCODE :-

It is defined to return some negative value when an error condition arises without specifying what error a particular negative integer denotes.

SQLSTATE :-

It associates predefined values with several common error conditions

Embedding SQL Statements

SQL statements must be prefixed by EXEC SQL. when embedded in a host language.

E.g.

```
EXEC SQL
Insert int Sailors values ( :c_sname, :c_sid,
                            :c_rating, :c_age);
```

The SQLSTATE variable should be checked for errors and exceptions after each embedded SQL statement

```
EXEC SQL WHENEVER [ SQLERROR | NOTFOUND ]
                  [ CONTINUE | GOTO stmt ]
```

(5). Explain about Dynamic SQL

Dynamic SQL

Dynamic SQL is a programming technique that could be used to write SQL queries during runtime.

To run a dynamic SQL statement, run the stored procedure SP_executesql as shown below:

```
EXEC  SP_executesql  N 'query';
```

Use prefix N with the SP_executesql to use dynamic SQL as a unicode string.

Steps to use Dynamic SQL:-

1. Declare two variables, @var1 for holding the name of the table and @var2 for holding the dynamic SQL:

```
DECLARE
  @var1  NVARCHAR(MAX),
  @var2  NVARCHAR(MAX);
```

2. Set the value of the @var1 variable to table_name.

```
SET @var1 = N 'table_name';
```

3. Create the dynamic SQL by adding the SELECT statement to the table name parameter;

```
SET @var2 = N'SELECT * FROM' + @var1;
```

4. Run the SP_executesql @var2;

E.g.

```
DECLARE
    @tab    NVARCHAR(128)
    @st     NVARCHAR(200);

SET @tab = N'Employee';
SET @st = N'SELECT * FROM' + @tab;

EXEC  SP_executesql @st;
```

(1) Differentiate between OLTP and OLAP

Feature	OLTP	OLAP
characteristic	Operational processing	Informational processing
Orientation	Transaction	Analysis
User	clerk, DBA, Database professional	Knowledge Worker
Function	Day-to-Day operations	Long-term informational requirements, Decision support
Database Design	ER based, Application oriented	Star/snowflake, subject-oriented
Data	Current; guaranteed up-to-date	historical; accuracy maintained over time.
Summarization	primitive, highly-detailed	summarized, consolidated.
View	Flat Detailed, Flat-relational	summarized, Multidimensional
Unit of work	Short, Simple transaction	complex query
Access	Read/Write	mostly read
Focus	Data in	Information out
(2) Operations	Index / hash on primary key	lots of scans

Feature	OLTP	OLAP
Number of records accessed	tens	millions
Number of users	thousands	hundreds.
Database Size	100 MB to GB	100 GB to TB
priority	high performance, high availability	high flexibility
Metric	transaction throughput	query throughput, response time.

② Explain about OLAP operations.

* OLAP stands for Online Analytical Processing.
* OLAP provides a user-friendly environment for interactive data analysis.
* OLAP can be performed in data warehouses / data marts using the Multidimensional data model.
* Typical OLAP operations include
  * Roll-up
  * Drill-down
  * Slice and dice
  * pivot
  * Drill-across
  * Drill-through
  * Ranking

Roll-up operation :-

It is also known as Drill-up operation.

It performs aggregation on a data cube, either by climbing up a concept hierarchy for a dimension or by dimension reduction.

Drill-down operation.

It is the reverse of Roll-up operation

It navigates from less detailed data to more detailed data.

Drill-down can be realized by either stepping down a concept hierarchy for a dimension or introducing additional dimensions.

* Slice-and-dice operation.

The slice operation performs a selection on one dimension of the given cube, resulting in a subcube.

the dice operation defines a subcube by performing a selection on two or more dimensions.

* pivot operation

It is a visualization operation that rotates the data axes in view inorder to provide an alternative representation of the data.

* Drill-across operation

The drill-across executes queries involving more than one fact table.

* Drill-through operation

The drill-through operation makes use of relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables.

UNIT-II (Questions from previous papers)

(1) (a) Discuss about outer join with examples

(b) Write a short note on triggers.

(2) Explain about set comparison operators, aggregate operators with example.

(3)(a) Show that, in SQL, <>all is identical to NOT IN

(b) Write the following queries in SQL using the university schema:

(i) Create a new course "CS-001" titled "Weekly Seminar" with 0 credits

(ii) Create a section of this course in Autumn 2009 with Sec id of 1

(iii) Enrol every student in the computer science department in the above section.

(iv) Delete enrolments in the above section where the student's name is Chavez.

(v) Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course.

(4) Describe how the theta-join operation can be extended so that tuples from the left, right or both relations are not lost from the result of a theta join.

(5) (a) The execution of a trigger can cause another action to be triggered. Most database systems place a limit on how deep the nesting can be. Explain why they might place such a limit.

(5) (b) Suppose there are two relations r and s, such that the foreign key B of r references the primary key A of s. Describe how the trigger mechanism can be used to implement the on delete cascade option, when a tuple is deleted from s.

(6) (a) What is a view in SQL? How is it defined? Explain with an example

(b) What are aggregate functions? List the aggregate functions supported by SQL.

(7) (a) Explain about Nested queries with an example.

(b) Write about logical connectives with an example.

(8) What is a join? Explain about various types of joins.

(9) Differentiate Nested loop join, Hash join and Merge join.

(10) Distinguish between integrity constraints and triggers.

(11) What is a foreign key? What are the different ways that SQL provides to handle foreign key violations?

(12) Consider a database with the following schema:

Person (name, age, gender) – name is a key.

Frequents (name, pizzeria) where (name, pizzeria) is a key

Eats (name, pizza) where (name, pizza) is a key

Serves (pizzeria, pizza, price) where (pizzeria, pizza) is a key.

Write SQL queries for the following.

(i) find all pizzerias frequented by atleast one person under the age of 18

(ii) find the names of all females who eat either mushroom or pepperoni pizza or both.

(iii) find the names of all females who eat both. mushroom and pepperoni pizza.

(iv) find all pizzerias that serve atleast one pizza that Amy eats less than $10.00