SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Unit 3

Basics of SQL-DDL,DML,DCL,TCL Nested Queries, Views and its **Structure Creation, alternation** Types

Defining Constraints-Primary Key, Foreign Key, Transaction Control Commands Unique, not null, check, IN operator Commit, Rollback, Savepoint

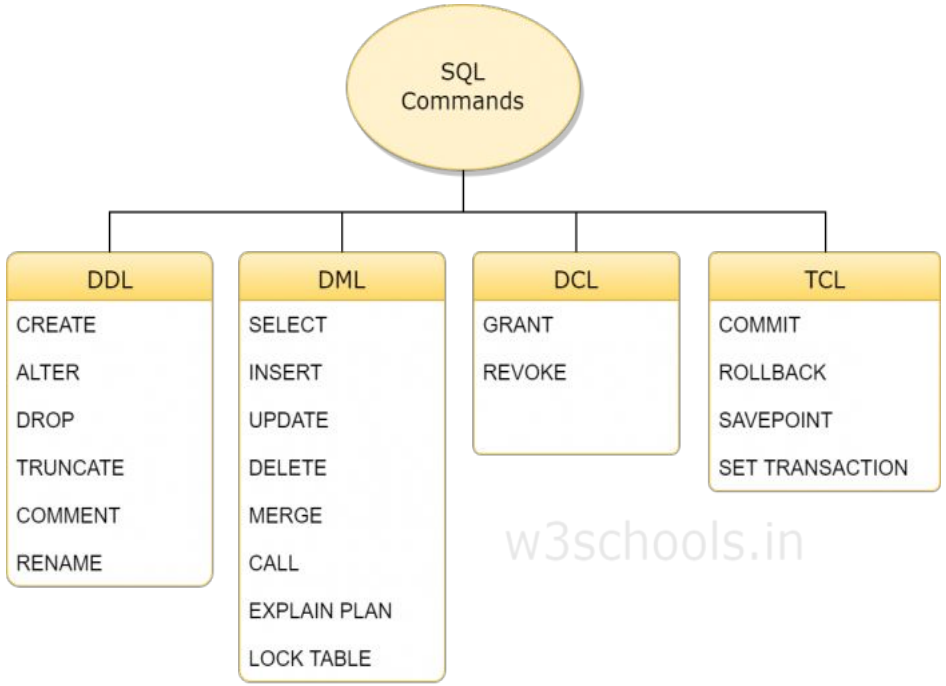
Functions-aggregation functions PL/SQL Concepts- Cursors

Built-in Functions-numeric, date, string

functions, string functions, Set operations, Stored Procedure, Functions Triggers and Exceptional Handling

Sub Queries, correlated sub queries Query Processing

Basics of SQL



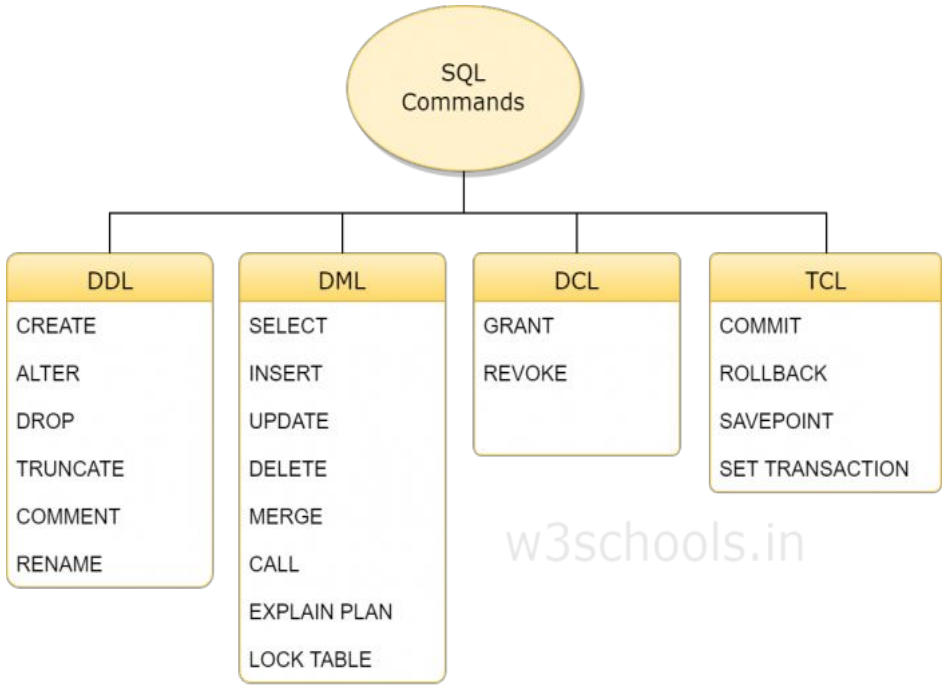
DDL is Data Definition Language statements

- Some examples

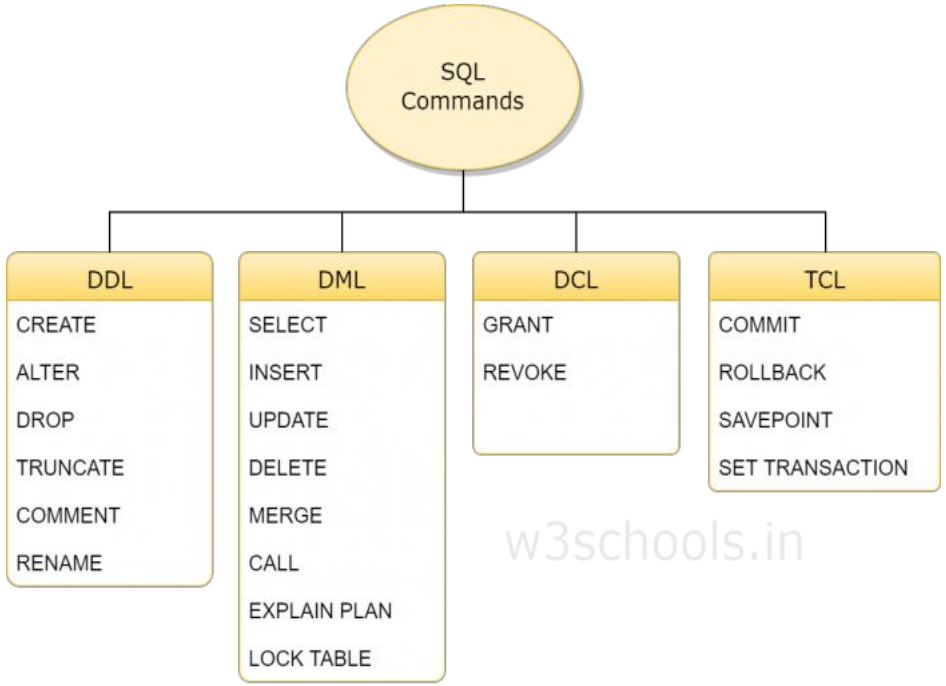
- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table
- COMMENT - add comments to the data dictionary

DML is Data Manipulation Language statements

- Some examples:



- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency



TCL(Transaction Control Language) is a DML

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like what rollback segment to use

DCL is Data Control Language statements

- Some examples:
- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

Data-Definition Language

- Set of definitions expressed by a special language called a data-definition language (DDL).
- The storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language.
- The data values stored in the database must satisfy certain consistency constraints.
- **Domain Constraints:** A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
- Domain constraints are the most elementary form of integrity constraint.
- **Referential Integrity:** There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).
- Database modifications can cause violations of referential integrity.
- **Assertions:** An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions.
- **Authorization:** To differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of authorization.

- Read Authorization - which allows reading, but not modification of data.
- Insert Authorization - which allows insertion of new data but not modification of existing data.
- update authorization - which allows modification but not deletion of data.
- Delete Authorization - which allows deletion of data.

- The output of the DDL is placed in the data dictionary which contains metadata - that is, data about data.
- SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc.

create table department (dept name char (20), building char (15), budget numeric (12,2));

- Execution of the above DDL statement creates the department table with three columns: dept name, building, and budget, each of which has a specific data type associated with it.

CREATING DATABASE TABLE

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- CREATE – creates a new table in the database

- Used to create a table by defining its structure, the data type and name of the various columns, the relationships with columns of other tables etc.

- CREATE TABLE table_name (column_name1 data_type(size), column_name2 data_type(size),…., column_nameN data_type(size));

- E.g.:

  CREATE TABLE Employee(Name varchar2(20), DOB date, Salary number(6));

ALTER - Add a new attribute or Modify the characteristics of some existing attribute.

- ALTER TABLE table_name ADD (column_name1 data_type (size), column_name2 data_type (size),….., column_nameN data_type (size));

E.g.:

ALTER TABLE Employee ADD (Address varchar2(20));

ALTER TABLE Employee ADD (Designation varchar2(20), Dept varchar2(3));

ALTER TABLE table_name MODIFY (column_name data_type(new_size));

E.g.:

ALTER TABLE Employee MODIFY (Name varchar2(30));

ALTER - dropping a column from the table

- ALTER TABLE table_name DROP COLUMN column_name;

E.g.:

ALTER TABLE Student DROP COLUMN Age;

DROP - Deleting an entire table from the database.

DROP TABLE table_name;

E.g.:

DROP TABLE Employee

RENAME – Renaming the table

RENAME old_table_name TO new_table_name;

E.g.:

RENAME Employee TO Employee_details

- TRUNCATE – deleting all rows from a table and free the space containing the table.

TRUNCATE TABLE table_name;

E.g.:

TRUNCATE TABLE Employee_details;

Data Manipulation Language

A DML statement is executed when you

- Add new rows to a table
- Modify existing rows in a table
- Remove existing rows from a table

Add new rows to a table by using the INSERT statement.

1. INSERT INTO table VALUES(value1, value2,..);

- Only one row is inserted at a time with this syntax.
- List values in the default order of the columns in the table
- Enclose character and date values within single quotation marks.
- Insert a new row containing values for each column.

E.g.:

- INSERT INTO Employee VALUES ('ashok', '16-mar-1998', 30000);

2. INSERT INTO table(column1, column2,..)VALUES(value1, value2,..);

- Rows can be inserted with NULL values either
  - by omitting column from the column list or
  - by specifying NULL in the value field.

E.g.:

- INSERT INTO Employee (name, dob, salary) VALUES ('ashok', '16-mar-1998', 30000);

3. INSERT INTO table_name1 SELECT column_name1, column_name2,….,column_nameN FROM table_name2;

- INSERT INTO Employee_details SELECT name, dob FROM Exmployee;

Data-Manipulation Language (DML)

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

- The SQL query language is nonprocedural.

select instructor.name from instructor where instructor.dept name = 'History';

- Queries may involve information from more than one table.

select instructor.ID, department.dept name from instructor, department where instructor.dept name= department.dept name and department.budget > 95000;

Basics of SQL-DDL,DML,DCL,TCL Nested Queries, Views and its Structure Creation, alternation Types

Defining Constraints-Primary Key, Foreign Key, Transaction Control Commands **Unique, not null, check, IN operator** Commit, Rollback, Savepoint

Functions-aggregation functions PL/SQL Concepts- Cursors

Built-in Functions-numeric, date, string

functions, string functions, Set operations, Stored Procedure, Functions Triggers and Exceptional Handling

Sub Queries, correlated sub queries Query Processing

SQL Constraints

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

- **Constraints in SQL**
  - ✔ Not Null
  - ✔ Unique
  - ✔ Primary Key
  - ✔ Foreign Key
  - ✔ Check
  - ✔ Default

- **How to specify constraints?**

- We can specify constraints at the time of creating the table using **CREATE TABLE** statement. We can also specify the constraints after creating a table using **ALTER TABLE** statement.

- **Syntax**
- CREATE TABLE sample_table(column1 data_type(size) constraint_name, column2 data_type(size) constraint_name, column3 data_type(size) constraint_name, ...);

- **sample_table**: Name of the table to be created.
- **data_type**: Type of data that can be stored in the field.
- **constraint_name**: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

NOT NULL

- If we specify a field in a table to be NOT NULL.
- Then the field will never accept null value.
- That is, you will be not allowed to insert a new row in the table without specifying any value to this field.

- **E.g.**
  - CREATE TABLE Student (ID int(6) NOT NULL, NAME varchar(10) NOT NULL, ADDRESS varchar(20));

UNIQUE

- This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.

- **E.g.**
  - CREATE TABLE Student (ID int(6) NOT NULL UNIQUE, NAME varchar(10), ADDRESS varchar(20));

PRIMARY KEY

- Primary Key is a field which uniquely identifies each row in the table.
- If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field.
- In other words we can say that this is combination of NOT NULL and UNIQUE constraints.
- A table can have only one field as primary key.

- **E.g.**
  - CREATE TABLE Student (ID int(6) NOT NULL UNIQUE, NAME varchar(10), ADDRESS varchar(20), PRIMARY KEY(ID));

FOREIGN KEY

- Foreign Key is a field in a table which uniquely identifies each row of a another table.
- That is, this field points to primary key of another table. This usually creates a kind of link between the tables.
- Foreign Key is used to relate two tables. The relationship between the two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
- This is also called a referencing key.
- We use onstraint.

Customer_Detail Table

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

- In **Customer_Detail** table, **c_id** is the primary key which is set as foreign key in **Order_Detail** table.
- The value that is entered in **c_id** which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key.
- This prevents invalid data to be inserted into **c_id** column of **Order_Detail** table.

- **FOREIGN KEY constraint at Table Level**
  - CREATE table Order_Detail(order_id int PRIMARY KEY, order_name varchar(60) NOT NULL, c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));

- **FOREIGN KEY constraint at Column Level**
  - ALTER table Order_Detail ADD FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);

CHECK Constraint

- **CHECK** constraint is used to restrict the value of a column between a range.
- It performs check on the values, before storing them into the database.
- It's like condition checking before saving data into a column.

- **Using CHECK constraint at Table Level**
  - CREATE table Student(s_id int NOT NULL CHECK(s_id > 0), Name varchar(60) NOT NULL, Age int);
- **Using CHECK constraint at Column Level**
  - ALTER table Student ADD CHECK(s_id > 0);

DEFAULT

- This constraint is used to provide a default value for the fields.
- That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.

- **E.g.**
  - CREATE TABLE Student (ID int(6) NOT NULL, NAME varchar(10) NOT NULL, AGE int DEFAULT 18);

Primary Key Vs Foreign Key

PRIMARY KEY VERSUS **FOREIGN KEY**

Primary Key	Foreign Key
A primary key uniquely identifies a record in the relational database table.	A foreign key refers to the field in a table which is the primary key of another table.
A table can contain only one primary key.	A table can contain more than one foreign key.
No two rows can carry duplicate values for a primary key attribute.	A foreign key can contain duplicate values.
A primary key does not allow Null values.	A foreign key can contain Null values.
A primary key constraint can be implicitly defined on the temporary tables.	A foreign key constraint cannot be enforced on local or global temporary tables.
A primary key constraint cannot be dropped from the parent table which referred to the foreign key in the child table.	A foreign key value can be dropped from the child table even if it is referred to the primary key of the parent table.

Primary Key Vs Unique Key

Primary Key	Unique Key
Primary Key can't accept null values.	Unique key can accept only one null value.
By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.	By default, Unique key is a unique non-clustered index.
We can have only one Primary key in a table.	We can have more than one unique key in a table.
Primary key can be made foreign key into another table.	In SQL Server, Unique key can be made foreign key into another table.

Basics of SQL-DDL,DML,DCL,TCL

Structure Creation, alternation

Views and its Types

Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator

Transaction Control Commands Commit, Rollback, Savepoint

Functions-aggregation functions

Built-in Functions-numeric, date, string functions, string functions, Set operations,

PL/SQL Concepts- Cursors

Stored Procedure, Functions

Triggers and Exceptional Handling

Sub Queries, correlated sub queries

Nested Queries,

Query Processing

Aggregate functions in SQL

- In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions are

- Count()
- Sum()
- Avg()
- Min()
- Max()

Count()

- ***Count(*):*** Returns total number of records .i.e 6.
- ***Count(salary):*** Return number of Non Null values over the column salary. i.e 5.
-

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

- ***Count(Distinct Salary):*** Return number of distinct Non Null values over the column salary .i.e 4

- **Sum()**
- ***sum(salary):*** Sum all Non Null values of Column salary i.e., 310
- ***sum(Distinct salary):*** Sum of all distinct Non-Null values i.e., 250.

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

- **Avg()**
- ***Avg(salary)*** = Sum(salary) / count(salary) = 310/5
- ***Avg(Distinct salary)*** = sum(Distinct salary) / Count(Distinct Salary) = 250/4

- **Min(), Max()**
- ***Min(salary):*** Minimum value in the salary column except NULL i.e., 40.
- ***Max(salary):*** Maximum value in the salary i.e., 80.

Built in Functions - Numeric Functions in SQL

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **ABS():** It returns the absolute value of a number.
  Syntax: SELECT ABS(-243.5);
  Output: 243.5
- **ACOS():** It returns the cosine of a number.
  Syntax: SELECT ACOS(0.25);
  Output: 1.31811607165281 8
- **ASIN():** It returns the arc sine of a number.
  Syntax: SELECT ASIN(0.25);
  Output: 0.25268025514207865
- **ATAN():** It returns the arc tangent of a number.
  Syntax: SELECT ATAN(2.5);
  Output: 1.190289949682531 7
- **CEIL():** It returns the smallest integer value that is greater than or equal to a number.
  Syntax: SELECT CEIL(25.75);
  Output: 26
- **CEILING():** It returns the smallest integer value that is greater than or equal to a number.
  Syntax: SELECT CEILING(25.75);
  Output: 26
- **COS():** It returns the cosine of a number.
  Syntax: SELECT COS(30);
  Output: 0.15425144988758405

Built in Functions - Numeric Functions in SQL

- **COT():** It returns the cotangent of a number.
  Syntax: SELECT COT(6);
  Output: -3.436353004180128
- **DEGREES():** It converts a radian value into degrees.
  Syntax: SELECT DEGREES(1.5);
  Output: 85.94366926962348
- **DIV():** It is used for integer division.
  Syntax: SELECT 10 DIV 5;
  Output: 2
- **EXP():** It returns e raised to the power of number.
  Syntax: SELECT EXP(1);
  Output: 2.718281828459045
- **FLOOR():** It returns the largest integer value that is less than or equal to a number.
  Syntax: SELECT FLOOR(25.75);
  Output: 25
- **GREATEST():** It returns the greatest value in a list of expressions.
  Syntax: SELECT GREATEST(30, 2, 36, 81, 125);
  Output: 125
- **LEAST():** It returns the smallest value in a list of expressions.
  Syntax: SELECT LEAST(30, 2, 36, 81, 125);
  Output: 2
- **LN():** It returns the natural logarithm of a number.
  Syntax: SELECT LN(2);
  Output: 0.6931471805599453
- **LOG10():** It returns the base-10 logarithm of a number.
  Syntax: SELECT LOG(2);
  Output: 0.6931471805599453

Built in Functions - Numeric Functions in SQL

- **LOG2():** It returns the base-2 logarithm of a number.
  Syntax: SELECT LOG2(6);
  Output: 2.584962500721156
- **MOD():** It returns the remainder of n divided by m.
  Syntax: SELECT MOD(18, 4);
  Output: 2
- **PI():** It returns the value of PI displayed with 6 decimal places.
  Syntax: SELECT PI();
  Output: 3.141593
- **POW():** It returns m raised to the nth power.
  Syntax: SELECT POW(4, 2);
  Output: 16
- **RADIANS():** It converts a value in degrees to radians.
  Syntax: SELECT RADIANS(180);
- **RAND():** It returns a random number.
  Syntax: SELECT RAND();
  Output: 0.33623238684258644
- **ROUND():** It returns a number rounded to a certain number of decimal places.
  Syntax: SELECT ROUND(5.553);
  Output: 6
- **SIGN():** It returns a value indicating the sign of a number.
  Syntax: SELECT SIGN(255.5);
  Output: 1
- **SIN():** It returns the sine of a number.
  Syntax: SELECT SIN(2);
  Output: 0.9092974268256817

Built in Functions - Numeric Functions in SQL

- **SQRT():** It returns the square root of a number.
  Syntax: SELECT SQRT(25);
  Output: 5

- **TAN():** It returns the tangent of a number.
  Syntax: SELECT TAN(1.75);
  Output: -5.52037992250933

- **ATAN2():** It returns the arctangent of the x and y coordinates, as an angle and expressed in radians.
  Syntax: SELECT ATAN2(7);
  Output: 1.42889927219073

- **TRUNCATE():** This doesn't work for SQL Server. It returns 7.53635 truncated to 2 places right of the decimal point.
  Syntax: SELECT TRUNCATE(7.53635, 2);
  Output: 7.53

Built in Functions - String functions in SQL

- **ASCII():** This function is used to find the ASCII value of a character.
  Syntax: SELECT ascii('t');
  Output: 116
- **CHAR_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a word. **Syntax:** SELECT char_length('Hello!');
  Output: 6
- **CHARACTER_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a line.
  Syntax: SELECT CHARACTER_LENGTH('geeks for geeks');
  Output: 15
- **CONCAT():** This function is used to add two words or strings.
  Syntax: SELECT 'Geeks' || ' ' || 'forGeeks';
  Output: 'GeeksforGeeks'
- **CONCAT_WS():** This function is used to add two words or strings with a symbol as concatenating symbol.
  Syntax: SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');
  Output: geeks_for_geeks
- **FIND_IN_SET():** This function is used to find a symbol from a set of symbols.
  Syntax: SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');
  Output: 2
- **FORMAT():** This function is used to display a number in the given format.
  Syntax: Format("0.981", "Percent");
  Output: '98.10%'

Built in Functions - String functions in SQL

- **INSTR():** This function is used to find the occurrence of an alphabet.
  Syntax: INSTR('geeks for geeks', 'e');
  Output: 2 (the first occurrence of 'e')
  Syntax: INSTR('geeks for geeks', 'e', 1, 2);
  Output: 3 (the second occurrence of 'e')
- **LCASE():** This function is used to convert the given string into lower case.
  Syntax: LCASE ("GeeksFor Geeks To Learn");
  Output: geeksforgeeks to learn
- **LEFT():** This function is used to SELECT a sub string from the left of given size or characters.
  Syntax: SELECT LEFT('geeksforgeeks.org', 5);
  Output: geeks
- **LENGTH():** This function is used to find the length of a word.
  Syntax: LENGTH('GeeksForGeeks');
  Output: 13
- **LOCATE():** This function is used to find the nth position of the given word in a string.
  Syntax: SELECT LOCATE('for', 'geeksforgeeks', 1);
  Output: 6
- **LOWER():** This function is used to convert the upper case string into lower case.
  Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');
  Output: geeksforgeeks.org
- **LPAD():** This function is used to make the given string of the given size by adding the given symbol.
  Syntax: LPAD('geeks', 8, '0');
  Output: 000geeks

Built in Functions - String functions in SQL

- **LTRIM():** This function is used to cut the given sub string from the original string.
  Syntax: LTRIM('123123geeks', '123’);
  Output: geeks
- **MID():** This function is to find a word from the given position and of the given size.
  Syntax: Mid ("geeksforgeeks", 6, 2);
  Output: for
- **POSITION():** This function is used to find position of the first occurrence of the given alphabet.
  Syntax: SELECT POSITION('e' IN 'geeksforgeeks’);
  Output: 2
- **REPEAT():** This function is used to write the given string again and again till the number of times mentioned.
  Syntax: SELECT REPEAT('geeks', 2);
  Output: geeksgeeks
- **REPLACE():** This function is used to cut the given string by removing the given sub string.
  Syntax: REPLACE('123geeks123', '123’);
  Output: geeks
- **REVERSE():** This function is used to reverse a string.
  Syntax: SELECT REVERSE('geeksforgeeks.org’);
  Output: ‘gro.skeegrofskeeg’
- **RIGHT():** This function is used to SELECT a sub string from the right end of the given size.
  Syntax: SELECT RIGHT('geeksforgeeks.org', 4);
  Output: ‘.org’
- **RPAD():** This function is used to make the given string as long as the given size by adding the given symbol on the right.
  Syntax: RPAD('geeks', 8, '0’);
  Output: ‘geeks000’
- **RTRIM():** This function is used to cut the given sub string from the original string.
  Syntax: RTRIM('geeksxyxzyyy', 'xyz’);
  Output: ‘geeks’
- **SPACE():** This function is used to write the given number of spaces.
  Syntax: SELECT SPACE(7);
  Output: ‘ ‘

Built in Functions - String functions in SQL

- **STRCMP():** This function is used to compare 2 strings.
  - If string1 and string2 are the same, the STRCMP function will return 0.
  - If string1 is smaller than string2, the STRCMP function will return -1.
  - If string1 is larger than string2, the STRCMP function will return 1.

  Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');
  Output: -1
- **SUBSTR():** This function is used to find a sub string from the a string from the given position.
  Syntax:SUBSTR('geeksforgeeks', 1, 5);
  Output: 'geeks'
- **SUBSTRING():** This function is used to find an alphabet from the mentioned size and the given string.
  Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);
  Output: 'G'
- **SUBSTRING_INDEX():** This function is used to find a sub string before the given symbol.
  Syntax: SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);
  Output: 'www'
- **TRIM():** This function is used to cut the given symbol from the string.
  Syntax: TRIM(LEADING '0' FROM '000123');
  Output: 123
- **UCASE():** This function is used to make the string in upper case.
  Syntax: UCASE ("GeeksForGeeks");
  Output: GEEKSFORGEEKS

Built in Functions - Date functions in SQL

- **NOW():** Returns the current date and time.
  Example: SELECT NOW();
  Output:2017-01-13 08:03:52
- **CURDATE():** Returns the current date.
  Example: SELECT CURDATE();
  Output: 2017-01-13
- **CURTIME():** Returns the current time.
  Example: SELECT CURTIME();
  Output: 08:05:15
- **DATE()**: Extracts the date part of a date or date/time expression.
- **EXTRACT():** Returns a single part of a date/time.

  Syntax: EXTRACT(unit FORM date);
  SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;

Built in Functions - Date functions in SQL

- **DATE_ADD() :** Adds a specified time interval to a date

  Syntax: DATE_ADD(date, INTERVAL expr type);

  SELECT Name, DATE_ADD(BirthTime, INTERVAL 1 YEAR) AS BirthTimeModified FROM Test;

- **DATE_SUB():** Subtracts a specified time interval from a date. Syntax for DATE_SUB is same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.

- **DATEDIFF():** Returns the number of days between two dates.

  Syntax: DATEDIFF(date1, date2); date1 & date2- date/time expression

SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;

Output: 10

Built in Functions - Date functions in SQL

- **DATE_FORMAT():** Displays date/time data in different formats.
- **Syntax:** DATE_FORMAT(date,format);

- %a-Abbreviated weekday name (Sun-Sat)
- %b-Abbreviated month name (Jan-Dec)
- %c-Month, numeric (0-12)
- %D-Day of month with English suffix (0th, 1st, 2nd, 3rd)
- %d-Day of month, numeric (00-31)
- %e-Day of month, numeric (0-31)
- %f-Microseconds (000000-999999)
- %H-Hour (00-23)
- %h-Hour (01-12)
- %I-Hour (01-12)
- %i-Minutes, numeric (00-59)
- %j-Day of year (001-366)
- %k-Hour (0-23)
- %l-Hour (1-12)
- %M-Month name (January-December)
- %m-Month, numeric (00-12)
- %p-AM or PM
- %r-Time, 12-hour (hh:mm:ss followed by AM or PM)
- %S-Seconds (00-59)
- %s-Seconds (00-59)
- %T-Time, 24-hour (hh:mm:ss)
- %U-Week (00-53) where Sunday is the first day of week
- %u-Week (00-53) where Monday is the first day of week
- %V-Week (01-53) where Sunday is the first day of week, used with %X
- %v-Week (01-53) where Monday is the first day of week, used with %x
- %W-Weekday name (Sunday-Saturday)
- %w-Day of the week (0=Sunday, 6=Saturday)
- %X-Year for the week where Sunday is the first day of week, four digits, used with %V
- %x-Year for the week where Monday is the first day of week, four digits, used with %v
- %Y-Year, numeric, four digits
- %y-Year, numeric, two digits

Example:

```
DATE_FORMAT(NOW(),'%d %b %y')
```

Result:

```
13 Jan 17
```

Set Operation functions in SQL

- The SQL Set operation is used to combine the two or more SQL SELECT statements.

- **Types of Set Operation**

UNION Operation

- **UNION** is used to combine the results of two or more SELECT statements.
- However it will eliminate duplicate rows from its resultset.
- In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union All

- Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example: Using the above First and Second table.

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

Using the above First and Second table.

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	NAME
3	Jackson

Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example

Using the above First and Second table.

Minus query will be:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry

Basics of SQL-DDL,DML,DCL,TCL

Structure Creation, alternation

Views and its Types

Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator

Transaction Control Commands Commit, Rollback, Savepoint

Functions-aggregation functions

Built-in Functions-numeric, date, string functions, string functions, Set operations,

PL/SQL Concepts- Cursors

Stored Procedure, Functions

Triggers and Exceptional Handling

Sub Queries, correlated sub queries

Nested Queries,

Query Processing

Subquery

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.
- **Important rules for Subqueries**
- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command cannot be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

• Subqueries with SELECT statement

- **Syntax**
- SELECT column_name FROM table_name WHERE column_name *expression operator* (SELECT COLUMN_NAME from TABLE_NAME WHERE ...);

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

STUDENT

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ravi	104	Salem	8989898989
Raj	102	Coimbatore	8877665544

- Select NAME, LOCATION, PHONE_NUMBER from DATABASE WHERE ROLL_NO IN (SELECT ROLL_NO from STUDENT where SECTION=’A’);

- **Subqueries with the INSERT Statement**

- Subqueries also can be used with INSERT statements.
- The INSERT statement uses the data returned from the subquery to insert into another table.
- The selected data in the subquery can be modified with any of the character, date or number functions.

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- INSERT INTO Student1 SELECT * FROM Student2

Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- **Subqueries with the UPDATE Statement**

- The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

- To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- UPDATE Student2 SET NAME = ‘geeks’ WHERE LOCATION IN (SELECT LOCATION FROM Student1 WHERE NAME IN (‘Raju’,’

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
geeks	113	coimbatore	7878787878

Subqueries with the DELETE Statement

- The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.
- To delete students from Student2 table whose rollno is same as that in Student1 table and having location as Chennai.

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

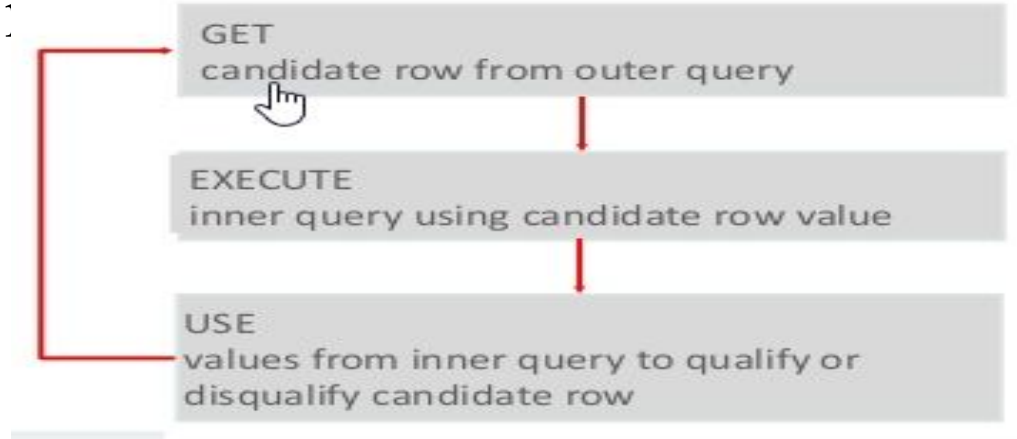
NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- DELETE FROM Student2 WHERE ROLL_NO IN (SELECT ROLL_NO FROM Student1 WHERE LOCATION = 'chennai');

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

SQL Correlated Subqueries

- Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.
- A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.
- A correlated subquery is one way of reading every row in a table and comparing values in each row against related data.
- It is used whenever a subquery must return a different result or set of results for each candidate



Nested Subqueries Versus Correlated Subqueries

- With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query.

- A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

- **NOTE :** You can also use the **ANY** and **ALL** operator in a correlated subquery.

Correlated Subqueries with Select Statement

- Find all the employees who earn more than the average salary in their department.

```
SQL> SELECT empno, sal, deptno
  2  FROM   emp outer
  3  WHERE  sal > (SELECT AVG(sal)
  4                FROM   emp inner
  5                WHERE  outer.deptno = inner.deptno);
```

Each time the outer query is processed the inner query is evaluated.

```
   EMPNO        SAL     DEPTNO
-------- ---------- ---------
    7839       5000         10
    7698       2850         30
    7566       2975         20
...
6 rows selected.
```

Using the Exists Operator

- The EXISTS operator tests for existence of rows in the results set of the subquery.

- If a subquery row value is found the condition is flagged **TRUE** and the search does not continue in the inner query, and if it is not found then the condition is flagged **FALSE** and the search continues in the inner query.
- Find the employee[illegible]eporting to them.

```
SQL> SELECT empno, ename, job, deptno
  2  FROM   emp outer
  3  WHERE  EXISTS (SELECT empno
  4                 FROM   emp inner
  5                 WHERE  inner.mgr = outer.empno);

     EMPNO ENAME      JOB           DEPTNO
---------- ---------- --------- ----------
      7839 KING       PRESIDENT         10
      7698 BLAKE      MANAGER           30
      7782 CLARK      MANAGER           10
      7566 JONES      MANAGER           20
...
6 rows selected.
```

Using the Not Exists Operator

- Find all the departments that do not have any employees

```
SQL> SELECT   deptno, dname
  2  FROM     dept d
  3  WHERE    NOT EXISTS (SELECT '1'
  4                       FROM   emp e
  5                       WHERE  d.deptno = e.deptno);
```

```
    DEPTNO DNAME
---------- ----------
        40 OPERATIONS
```

CORRELATED UPDATE & DELETE

- **CORRELATED UPDATE**
- UPDATE table1 alias1 SET column = (SELECT expression FROM table2 alias2 WHERE alias1.column = alias2.column);
- Use a correlated subquery to update rows in one table based on rows from another table.

- **CORRELATED DELETE**

- DELETE FROM table1 alias1 WHERE column1 operator (SELECT expression FROM table2 alias2 WHERE alias1.column = alias2.column);
- Use a correlated subquery to delete rows in one table based on the rows from another table.

Processing a correlated subquery Using the Exists Operator - E.g.

What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
        (SELECT *
          FROM Product _T
                WHERE ProductID = OrderLine_T.ProductID
                    AND Productfinish = 'Natural Ash');
```

	OrderID	ProductID	OrderedQuantity
▶	1001	1	1
	1001	2	2
	1001	4	1
	1002	3	5
	1003	3	3
	1004	6	2
	1004	8	2
	1005	4	4
	1006	4	1
	1006	5	2
	1007	1	3
	1007	2	2
	1008	3	3
	1008	8	3
	1009	4	2
	1009	7	3
	1010	8	10
*	0	0	0

		ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
▶	⊞	1	End Table	Cherry	$175.00	10001
	⊞	2	Coffee Table	Natural Ash	$200.00	20001
	⊞	3	Computer Desk	Natural Ash	$375.00	20001
	⊞	4	Entertainment Center	Natural Maple	$650.00	30001
	⊞	5	Writer's Desk	Cherry	$325.00	10001
	⊞	6	8-Drawer Dresser	White Ash	$750.00	20001
	⊞	7	Dining Table	Natural Ash	$800.00	20001
	⊞	8	Computer Desk	Walnut	$250.00	30001
*		(AutoNumber)			$0.00	

1. The first order ID is selected from OrderLine_T: OrderID =1001.
2. The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as *true* and the order ID is added to the result table.
3. The next order ID is selected from OrderLine_T: OrderID =1002.
4. The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as *true* and the order ID is added to the result table.
5. Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 302.

Note: Only the orders that involve products with Natural Ash will be included in the final results.

Basics of SQL-DDL,DML,DCL,TCL **Views and its Types**

Structure Creation, alternation

Defining Constraints-Primary Key, Foreign Key, Transaction Control Commands
Unique, not null, check, IN operator Commit, Rollback, Savepoint

Functions-aggregation functions PL/SQL Concepts- Cursors

Built-in Functions-numeric, date, string

functions, string functions, Set operations, Stored Procedure, Functions
Triggers and Exceptional Handling

Sub Queries, correlated sub queries Query Processing

Nested Queries,

SQL Views

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

Student Details

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Student Marks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Creating a View

View can be created using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

Syntax

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

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

- **Creating View from a single table**
- CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM StudentDetails WHERE S_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.

SELECT * FROM DetailsView;

Student Details

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Output

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- **Creating View from multiple tables**
- In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks.
- To create a View from multiple tables we can simply include multiple tables in the SELECT statement.
    - CREATE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;
- To display data of View MarksView:
    - SELECT * FROM MarksView;

Student Details

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Student Marks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Output

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS

- SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax

```
DROP VIEW view_name;
```

view_name: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView.**

```
DROP VIEW MarksView;
```

UPDATING VIEWS

- There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

CREATE OR REPLACE VIEW

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

Syntax

CREATE OR REPLACE VIEW view_name AS SELECT column1,coulmn2,.. FROM table_name WHERE condition;

For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;

If we fetch all the data from MarksView now as:

SELECT * FROM MarksView;

Output

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row in a view

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax:
INSERT INTO view_name(column1, column2 , column3,..) VALUES(value1, value2, value3..); **view_name**: Name of the View

Example:
In the below example we will insert a new row in the View DetailsView which we have created above in the example of "creating views from a single table".

```
INSERT INTO DetailsView(NAME, ADDRESS) VALUES("Suresh","Gurgaon");
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

Deleting a row from a View

- Deleting rows from a view is also as simple as deleting rows from a table.
- We can use the DELETE statement of SQL to delete rows from a view.

- Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.
- **Syntax**
- DELETE FROM view_name WHERE condition;
- **view_name**:Name of view from where we want to delete rows
- **condition**: Condition to select rows

In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

```
DELETE FROM DetailsView WHERE NAME="Suresh";
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

Uses of a View

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

1. **Restricting data access**
   Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity**
   A view can hide the complexity that exists in a multiple table join.
3. **Simplify commands for the user**
   Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries**
   Views can be used to store complex queries.
5. **Rename Columns**
   Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to to hide the names of the columns of the base tables.
6. **Multiple view facility**
   Different views can be created on the same table for different users.

Basics of SQL-DDL,DML,DCL,TCL Nested Queries, Views and its Structure Creation, alternation Types

Defining Constraints-Primary Key, Foreign Key, **Transaction Control Commands** Unique, not null, check, IN operator **Commit, Rollback, Savepoint**

Functions-aggregation functions PL/SQL Concepts- Cursors

Built-in Functions-numeric, date, string

functions, string functions, Set operations, Stored Procedure, Functions Triggers and Exceptional Handling

Sub Queries, correlated sub queries Query Processing

TRANSACTION

- A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.
- A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.
- Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

- Transaction Control language is a language that manages transactions within the database. It is used to execute the changes made by the DML statements.

TCL Commands

Transaction Control Language (TCL) Commands are:

Commit − It is used to save the transactions in the database.

Rollback − It is used to restore the database to that state which was last committed.

Savepoint − The changes done till savpoint will be unchanged and all the transactions after savepoint will be rolled back.

Example

- Given below is an example of the usage of the TCL commands in the database management system (DBMS)

```
BEGIN TRANSACTION
UPDATE employees
SET empname='bob'
WHERE empid='001'

UPDATE employees
SET empname ='bob'
WHERE city='hyderabad'

IF @@ROWCOUNT=5
   COMMIT TRANSACTION
ELSE
   ROLLBACK TRANSACTION
```

Difference between Commit, rollback and savepoint of TCL commands

Sno.	Rollback	Commit	Savepoint
1.	Rollback means the database is restored to the last committed state	DML commands saves modification and it permanently saves the transaction.	Savepoint helps to save the transaction temporarily.
2.	Syntax- ROLLBACK [To SAVEPOINT_NAME];	Syntax- COMMIT;	Syntax- SAVEPOINT [savepoint_name;]
3.	Example- ROLLBACK Update5;	Example- SQL> COMMIT;	Example- SAVEPOINT table_create;

Properties of Transactions

- Transactions have the following four standard properties, usually referred to by the acronym **ACID**.

Atomicity − ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.

Consistency − ensures that the database properly changes states upon a successfully committed transaction.

Isolation − enables transactions to operate independently of and transparent to each other.

Durability − ensures that the result or effect of a committed transaction persists in case of a system failure.

Transaction Control

The following commands are used to control transactions.

COMMIT − to save the changes.

ROLLBACK − to roll back the changes.

SAVEPOINT − creates points within the groups of transactions in which to ROLLBACK.

COMMIT

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only.

They cannot be used while creating tables or dropping them because these operations are automatically committed in the database

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

Example

Consider the CUSTOMERS table having the following records

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
7	Muffy	24	Indore	10000.00

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> COMMIT;
```

Thus, two rows from the table would be deleted and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows :

```
ROLLBACK;
```

Example

Consider the CUSTOMERS table having the following records

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
7	Muffy	24	Indore	10000.00

Following is an example, which would delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

```
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
7	Muffy	24	Indore	10000.00

SAVEPOINT

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
  The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

- This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Following is an example where you plan to delete the three different records from the CUSTOMERS table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

- **Example**

Consider the CUSTOMERS table having the following records.

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
7	Muffy	24	Indore	10000.00

The following code block contains the series of operations.

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone −

```
SQL> ROLLBACK TO SP2;
Rollback complete.
```

Notice that only the first deletion took place since you rolled back to SP2.

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  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 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

RELEASE SAVEPOINT Command:
The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows:

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

Basics of SQL-DDL,DML,DCL,TCL Nested Queries, Views and its Structure Creation, alternation Types

Defining Constraints-Primary Key, Foreign Key, Transaction Control Commands Unique, not null, check, IN operator Commit, Rollback, Savepoint

Functions-aggregation functions **PL/SQL Concepts- Cursors**

Built-in Functions-numeric, date, string

functions, string functions, Set operations, Stored Procedure, Functions Triggers and Exceptional Handling

Sub Queries, correlated sub queries Query Processing

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Introduction to PL/SQL

- Procedural Language extension for SQL
- Oracle Proprietary
- 3GL Capabilities
- Integration of SQL
- Portable within Oracle data bases
- Callable from any client

Structure of PL/SQL

- PL/SQL is Block Structured

  A block is the basic unit from which all PL/SQL programs are built. A block can be named (functions and procedures) or anonymous

- Sections of block

  1- Header Section

  2- Declaration Section

  3- Executable Section

  4- Exception Section

Structure of PL/SQL

HEADER

Type and Name of block

DECLARE

Variables; Constants; Cursors;

BEGIN

PL/SQL and SQL Statements

EXCEPTION

Exception handlers

END;

Structure of PL/SQL

```
DECLARE
    a number;
    text1 varchar2(20);
      text2 varchar2(20) := “HI”;
BEGIN
      --------- --------- ---------
END;
```

Important Data Types in PL/SQL include NUMBER, INTEGER, CHAR, VARCHAR2, DATE etc

to_date(‘02-05-2007','dd-mm-yyyy') { Converts String to Date}

Structure of PL/SQL

- Data Types for specific columns

Variable_name Table_name.Column_name%type;

This syntax defines a variable of the type of the referenced column on the referenced table

PL/SQL Control Structure

- PL/SQL has a number of control structures which includes:
- Conditional controls
- Iterative or loop controls.
- Exception or error controls

- These control structure, can be used singly or together, that allow the PL/SQL developer to direct the flow of execution through the program.

PL/SQL Control Structure

- **Conditional Controls**

  IF....THEN....END IF;

  IF....THEN...ELSE....END IF;

  IF....THEN...ELSIF....THEN....ELSE....END IF;

PL/SQL Control Structure

- LOOP

  ...SQL Statements...

  EXIT;

  END LOOP;

- WHILE loops

  WHILE condition LOOP

  ...SQL Statements...

  END LOOP;

- FOR loops

  FOR <variable(numeric)> IN [REVERSE] <lowerbound>..<upperbound> LOOP .... END LOOP;

Cursor

- 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**.
- 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.
- In PL/SQL, the most recent implicit cursor is the **SQL cursor**, which has the following attributes.

Implicit Cursors

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.
2	**%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.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Example

The following program will update the employee 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;
```

Output:

```
6 customers selected
```

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

Cursor-Declaration

- **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

```
DECLARE
  c_id customers.id%type;
  c_name customer.name%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;
/
```

Basics of SQL-DDL,DML,DCL,TCL

Structure Creation, alternation

Views and its Types

Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator

Transaction Control Commands
Commit, Rollback, Savepoint

Functions-aggregation functions

Built-in Functions-numeric, date, string functions, string functions, Set operations,

PL/SQL Concepts- Cursors

Stored Procedure, Functions

Triggers and Exceptional Handling

Sub Queries, correlated sub queries

Nested Queries,

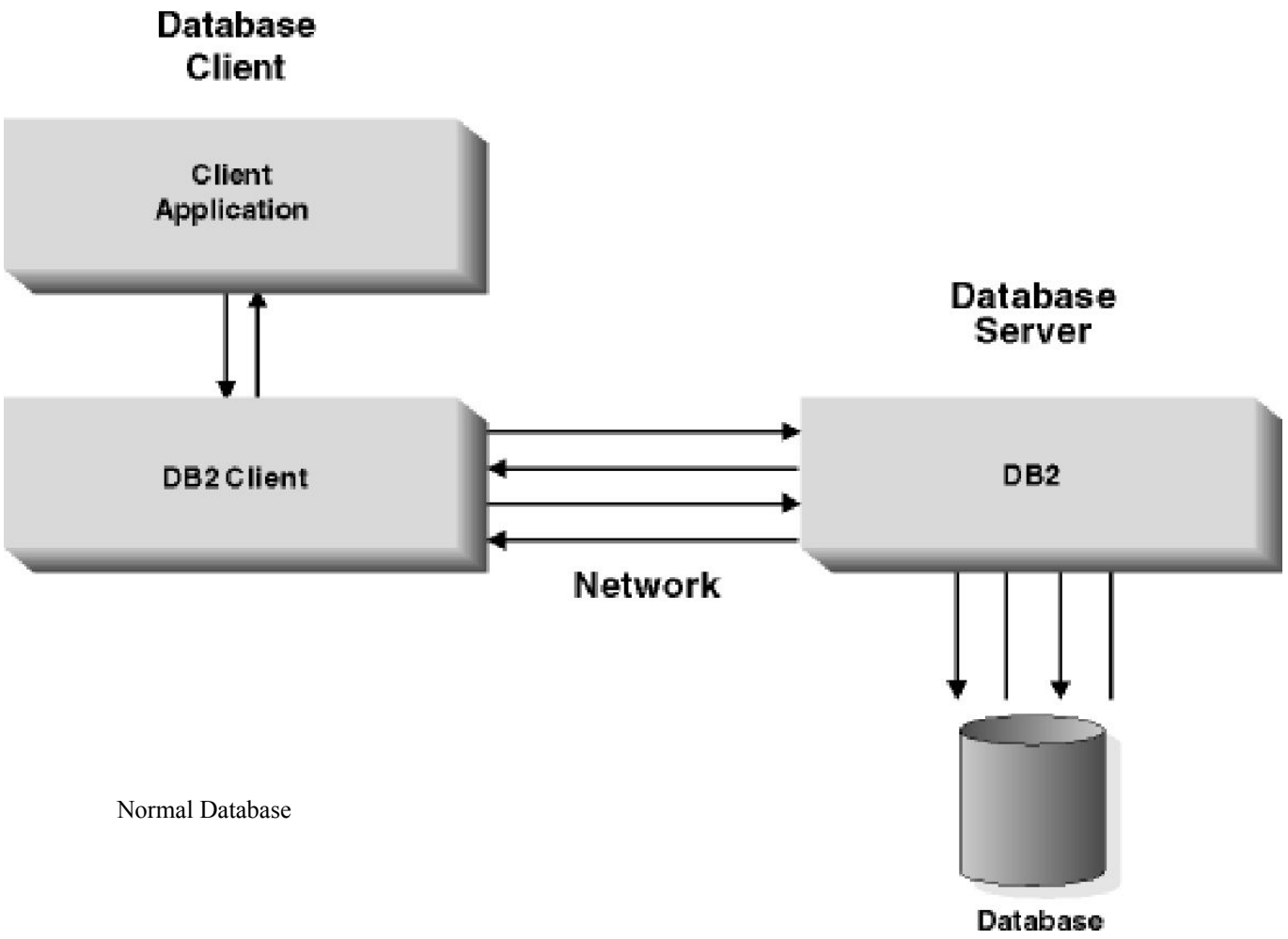
Query Processing

Stored Procedure

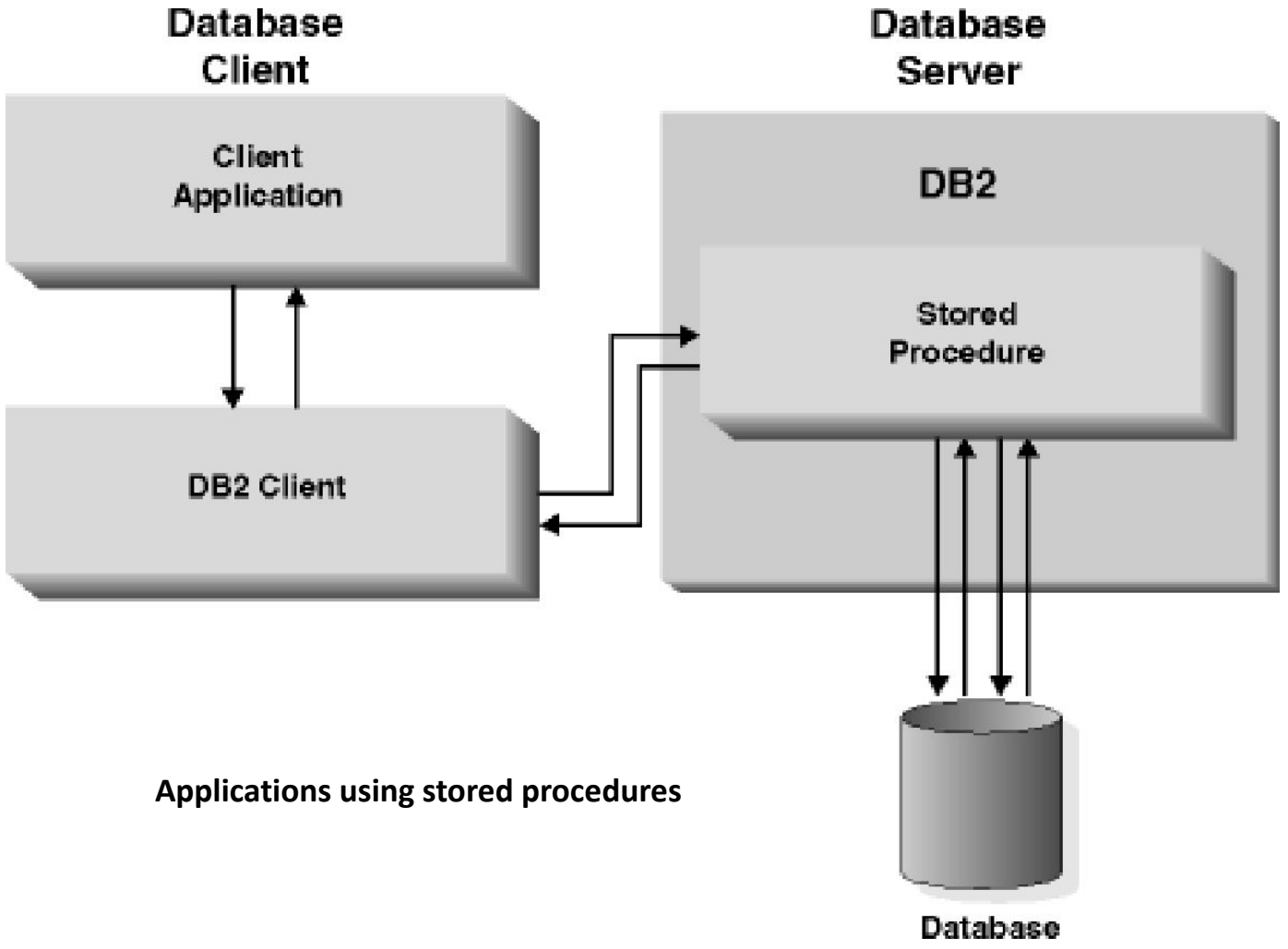
- A stored procedure in SQL is a group of SQL statements that are stored together in a database.
- Based on the statements in the procedure and the parameters you pass, it can perform one or multiple DML operations on the database, and return value, if any.
- Stored Procedure is a function in a shared library accessible to the database server
- can also write stored procedures using languages such as C or Java
- : Reduced network traffic
- The more SQL statements that are grouped together for execution, the larger the savings in network traffic

Advantages of stored procedure

- Reusable: As mentioned, multiple users and applications can easily use and reuse stored procedures by merely calling it.
- Easy to modify: You can quickly change the statements in a stored procedure as and when you want to, with the help of the ALTER TABLE command.
- Security: Stored procedures allow you to enhance the security of an application or a database by restricting the users from direct access to the table.
- Low network traffic: The server only passes the procedure name instead of the whole query, reducing network traffic.
- Increases performance: Upon the first use, a plan for the stored procedure is created and stored in the buffer pool for quick execution for the next time.



Normal Database



Applications using stored procedures

Writing Stored Procedures

- CREATE or REPLACE PROCEDURE name(parameters)
- AS
- variables;
- BEGIN;
- //statements;
- END;

Three types of parameters are:

- IN: It is the default parameter that will receive input value from the program
- OUT: It will send output value to the program
- IN OUT: It is the combination of both IN and OUT. Thus, it receives from, as well as sends a value to the program

EXAMPLE:

```
CREATE PROCEDURE UPDATE_SALARY_1        (1)
 (IN EMPLOYEE_NUMBER CHAR(6),           (2)
 IN RATE INTEGER)                       (2)
 LANGUAGE SQL                           (3)
 BEGIN
   UPDATE EMPLOYEE                      (4)
   SET SALARY = SALARY * (1.0 * RATE / 100.0 )
   WHERE SSN = EMPLOYEE_NUMBER;
 END
```

LANGUAGE value of SQL and the BEGIN...END block, which forms the procedure body, are particular to an SQL procedure

1)The stored procedure name is UPDATE_SALARY_1.

2)The two parameters have data types of CHAR(6) and INTEGER. Both are input parameters.

3)LANGUAGE SQL indicates that this is an SQL procedure, so a procedure body follows the other parameters.

4)The procedure body consists of a single SQL UPDATE statement, which updates rows in the employee table.

Some Valid SQL Procedure Body Statements

- CASE statement
- FOR statement
- GOTO statement
- IF statement
- ITERATE statement
- RETURN statement
- WHILE statement

- **Invoking Procedures**

  Can invoke Stored procedure stored at the location of the database by using the SQL CALL statement

- **Nested SQL Procedures:**

  To call a target SQL procedure from within a caller SQL procedure, simply include a CALL statement with the appropriate number and types of parameters in your caller.

```
CREATE PROCEDURE NEST_SALES(OUT budget DECIMAL(11,2))
        LANGUAGE SQL
        BEGIN
          DECLARE total INTEGER DEFAULT 0;
          SET total = 6;
          CALL SALES_TARGET(total);
          SET budget = total * 10000;
        END
```

CONDITIONAL STATEMENTS:

```
IF <condition> THEN
    <statement(s)>
ELSE
    <statement(s)>
END IF;
```

Loops

```
LOOP
    ……
    EXIT WHEN <condition>
    ……
END LOOP;
```

EXAMPLE :

```
CREATE PROCEDURE UPDATE_SALARY_IF
        (IN employee_number CHAR(6), IN rating SMALLINT)
        LANGUAGE SQL
        BEGIN
SET counter = 10;
WHILE (counter > 0) DO
          IF (rating = 1)
                    THEN UPDATE employee
               SET salary = salary * 1.10, bonus = 1000
               WHERE empno = employee_number;
           ELSEIF (rating = 2)
                    THEN UPDATE employee
               SET salary = salary * 1.05, bonus = 500
               WHERE empno = employee_number;
          ELSE UPDATE employee
               SET salary = salary * 1.03, bonus = 0
               WHERE empno = employee_number;
          END IF;
SET counter = counter – 1;
           END WHILE;
         END
       @
```

Triggers

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- Triggers are composed to be executed in light of any of the accompanying occasions.
  - A database control (DML) statement (DELETE, INSERT, or UPDATE).
  - A database definition (DDL) statement (CREATE, ALTER, or DROP).
  - A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

The syntax of Triggers in SQL–

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

Create [OR REPLACE] TRIGGER trigger_name: It makes or replaces a current trigger with the trigger_name

EXAMPLE

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.salary - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary);
dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

After creating a Trigger, use it in the PL/SQL code for putting it in to action.


```
DECLARE

total_rows number(2);

BEGIN

UPDATE customers

SET salary = salary + 5000;

IF sql%notfound THEN

dbms_output.put_line('no customers updated');

ELSIF sql%found THEN

total_rows := sql%rowcount;

dbms_output.put_line( total_rows || ' customers updated ');

END IF;

END;

/
```

Advantages of Triggers

- Triggers can be written for the following purposes –
- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Basics of SQL-DDL,DML,DCL,TCL

Structure Creation, alternation

Views and its Types

Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator

Transaction Control Commands
Commit, Rollback, Savepoint

Functions-aggregation functions

Built-in Functions-numeric, date, string functions, string functions, Set operations,

PL/SQL Concepts- Cursors

Stored Procedure, Functions

Triggers and Exceptional Handling

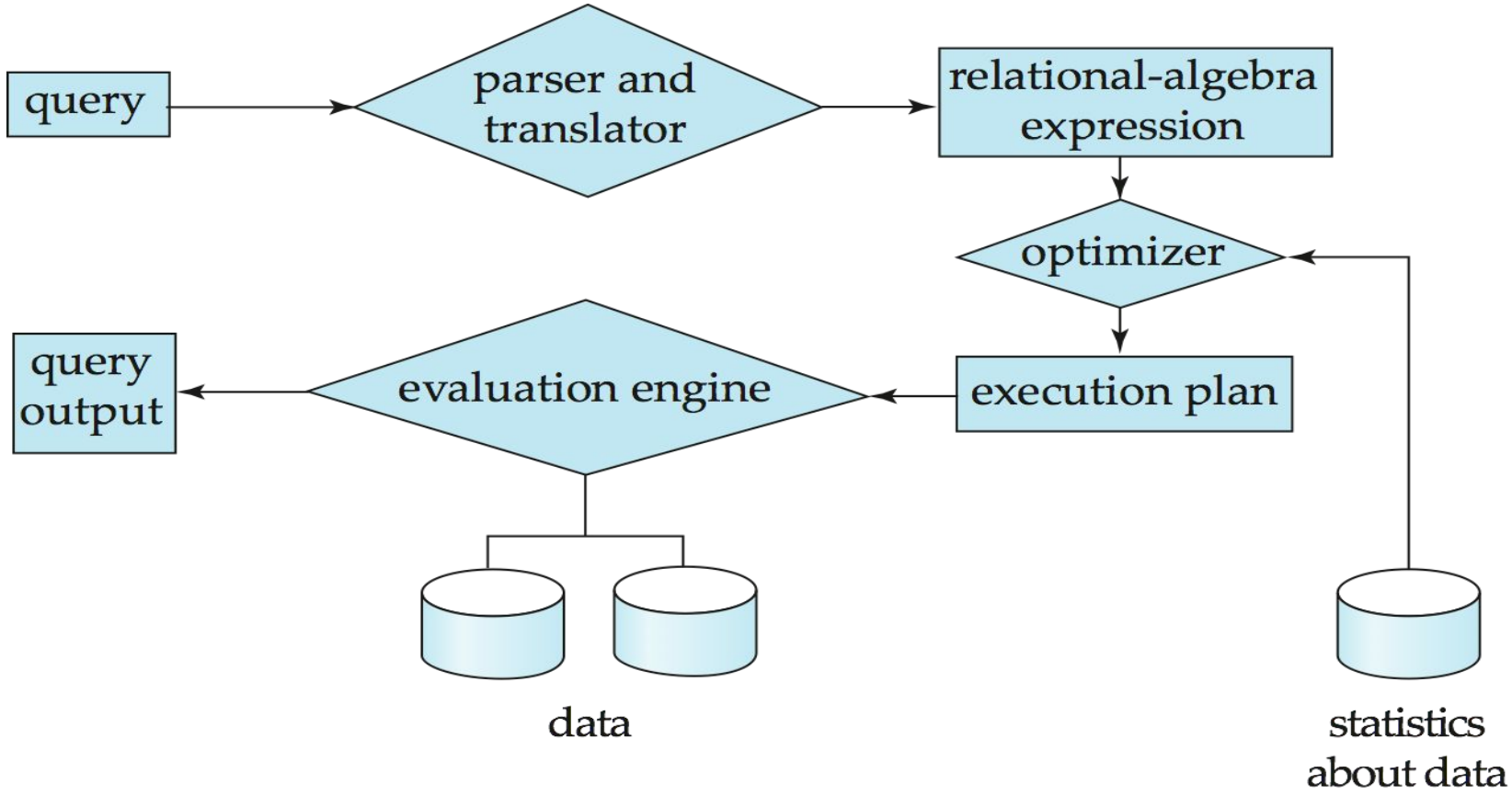
Sub Queries, correlated sub queries

Nested Queries,

Query Processing

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

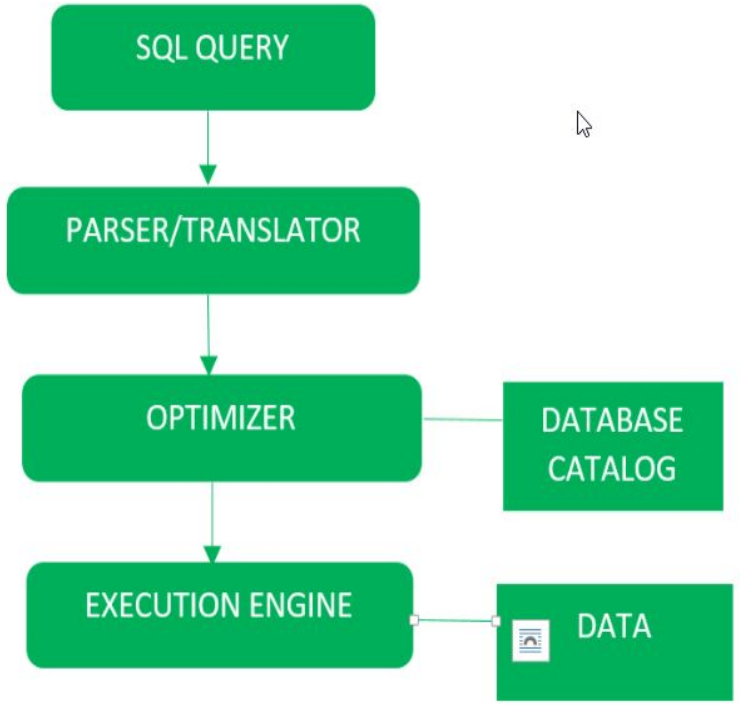
Transaction Management

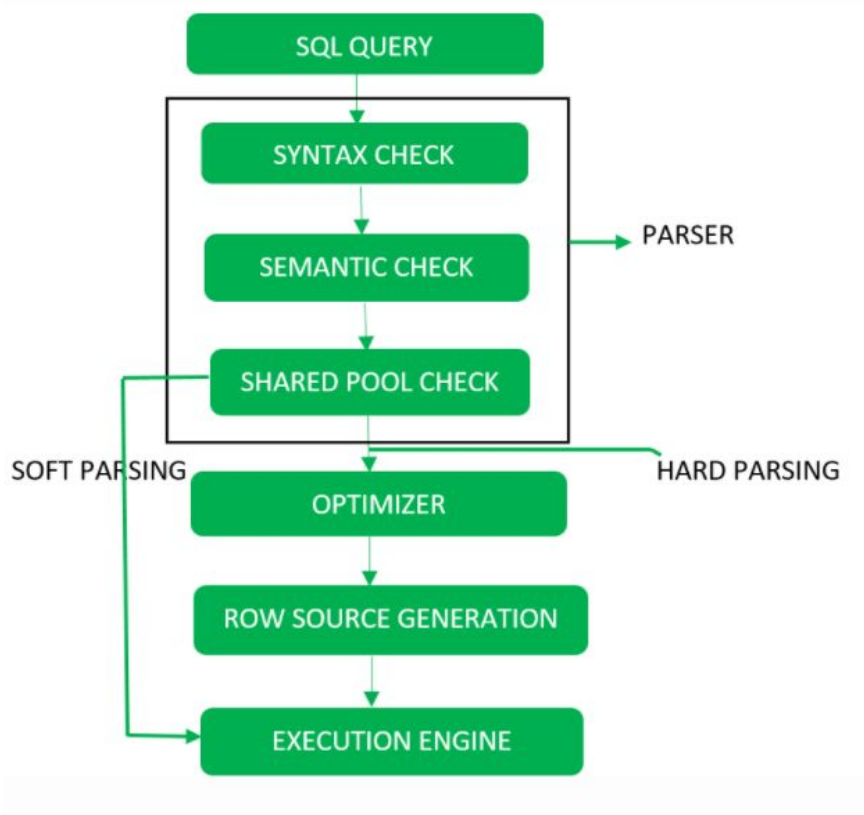
- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Query Processing

- **Query Processing** includes translations on high level Queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result.

- **Block Diagram of Query Processing** **Detailed DiagramQuery Processing**





- **Step-1**
- **Parser:** During parse call, the database performs the following checks- Syntax check, Semantic check and Shared pool check, after converting the query into relational algebra.
- Parser performs the following
- **Syntax check –** concludes SQL syntactic validity.
  - SELECT * FROM employee;

1. **Semantic check –** determines whether the statement is meaningful or not. Example: query contains a tablename which does not exist is checked by this check.
2. **Shared Pool check –** Every query possess a hash code during its execution. So, this check determines existence of written hash code in shared pool if code exists in shared pool then database will not take additional steps for optimization and execution.

- **Hard Parse and Soft Parse**
- If there is a fresh query and its hash code does not exist in shared pool then that query has to pass through from the additional steps known as hard parsing
- If hash code exists then query does not passes through additional steps. It just passes directly to execution engine. This is known as soft parsing.

- **Step-2**
  Optimizer
- During optimization stage, database must perform a hard parse at least for one unique DML statement and perform optimization during this parse. This database never optimizes DDL unless it includes a DML component such as subquery that require optimization.

- It is a process in which multiple query execution plan for satisfying a query are examined and most efficient query plan is satisfied for execution.
-
  Database catalog stores the execution plans and then optimizer passes the lowest cost plan for execution.

- **Row Source Generation**
- The Row Source Generation is a software that receives a optimal execution plan from the optimizer and produces an iterative execution plan that is usable by the rest of the database.
- The iterative plan is the binary program that when executes by the sql engine produces the result set.
- **Step-3**
  Execution Engine
- Finally runs the query and display the required result.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)