UNIT II

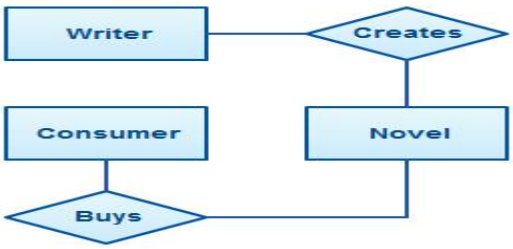
RELATIONAL MODEL:
INTRODUCTION TO THE RELATIONAL MODEL:

Definitions: The relational data model represents data in the form of tables. This model consists of three components.

1) **Data structure** : In RDBMS data are organized in the form of tables with rows and columns.
2) **Data manipulation** : In data manipulation the powerful operations are used to manipulate data stored in the relations.
3) **Data integrity**: In RDBMS, facilities are included to specify business rules, that maintain the integrity of data when they are manipulated.

What is relational model? How to create a relational table? Explain with suitable example

The E.F. Codd proposed the relational data model in 1970 To reduce the draw backs of hierarchical and network models. The relational model is very simple and elegant. A database is a collection of one or more relations. This relation is table with rows and columns. In this, the users can easy to understand and easy to permit the high level languages to query the data. The relational model uses Data Definition Language (DDL) and Data Manipulation Language (Query language) for creating, accessing, manipulating, and querying data in a relational DBMS The relational model represents both data and relationship among data in the form of tables. Each table has multiple columns and each column has a unique name. This is created by create command. Consider the following relational model. SQL> Create table customer(cid number(6), customer_city varchar(20), customer_street varchar(10), customer_city varchar(10));



What is Relation? Define the terms relation schema, relation instance, unary, degree of relation and domain constraints and its types.

Representing the data in the relational model is called relation. A relation contains set of records in the form of two-dimension table which contain rows and columns of data. A relation consists of two things, such as a relation schema and a relation instance.

Relation schema: A relation schema contains the basic information of a table or relation. This information includes the name of the entire table, the names of the column and the data types associated with each column. Example: A relation

schema for the relation called students could be expressed using the following representation, **Students (sid : string, name :string, course: string, age: integer, fees : real);** In this table, contain five column attributes with respective data types such as string, integers, real are associated with it.

Relation instance: A relation instance is a set of rows (also known as records) that when combined together forms the schema of the relation. A relation instance can be a table which each tuple (record) is a row and all rows has same number of fields (columns). Example, cid customer_name Customer_street customer_city 5555 Chaitanya Sri nagar Ongole 6666 Anusha Madhuri nagar Ongole 7777 Nithisha Swathi bazzar Ongole

Relational Database Schema: A relational database schema is a collection of relation schemas, describing one or more relations.

Domain: Domain is synonymous with data type. Attributes is a column in a table. Therefore an attribute domain refers to the database associated with a column.

Relational Cardinality: The relation cardinality is the number of tuples (rows or records) in the relation.

Customer Relation			
cid	customer_name	Customer_ street	customer_city
5555	Chaitanya	Gandhi nagar	Ongole
6666	Bhagya	sujatha nagar	Ongole
7777	Sumathi	jyothi bazzar	Ongole

Relation Degree: The relation degree is the number of fields (column or attribute) in the relation

Tuples/Records: The rows of the table are also known as fields or attributes.

Unary: A relation with only one attribute and have only one degree is called a unary relation.

How to create the relation and modify the relation using SQL:

1) Creating relation in SQL (Structured Query Language) : Relations are created in SQL using create command. SQL> Create table customer(cid number(6), cname varchar(20), accno number(6), amount number(8,2));

2)Inserting records in a table : Records are inserted in a table using an integer command as, SQL> insert into customer(cid, cname, accno, amount) values (5, „badri", 5555,15000.00); or SQL> insert into customer values (5, „badri", 5555,15000.00); or SQL> insert into customer values(&cid, „&cname", &accno, &amount);

3) Deleting tuples from a table : Tuples in the table can be deleted using delete command in SQL

4) Modifying the column values: Values in a particular row can be changed using an update command as, SQL> update customer set cname = „Chaithanya" amount = 15000.00; SQL> update customer set amount = amount + 1000 where accno <= 5555;

DEFINE THE TERMS: DOMAIN, TUPLE, ATTRIBUTE AND RELATION

Relation: A relation in a relational database is described as a table. Table is organized into rows and columns.

Attribute: A column header of a table is known as attribute of a relation.

Beyond the self-explanatory simple or single-valued attributes, there are several types of attributes available.

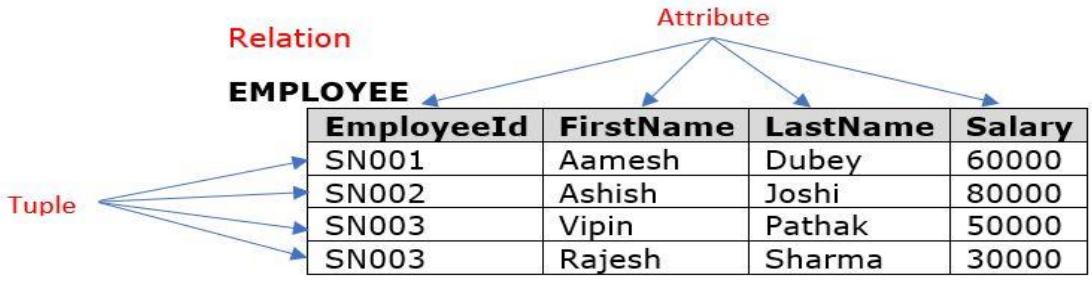
Composite attribute: is an attribute composed of several other simple attributes. For example, the Address attribute of an Employee entity could consist of the Street, City, Postal code and Country attributes.

Multivalued attribute: is an attribute where more than one description can be provided. Example:Employee entity may have more than one Email ID attributes in the same cell.

Derived attribute: as the name implies, these are derived from other attributes, either directly or through specific formula results. For example, the Age attribute of an Employee could be derived from the Date of Birth attribute. In other instances, a formula might calculate the VAT of a certain payment, so that whenever the cell with the attribute Payment is filled, the cell with the derived attribute VAT automatically calculates its value.

Tuple: A row in a table represents the record of a relation and known as a tuple of a relation

Domain: A set of possible values for a given attribute is known as domain of a relation. A domain is atomic, that means values are indivisible. For example, set of values for attribute FirstName of an EMPLOYEE relation are atomic.



IMPORTANCE OF NULL VALUES:

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank. A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Principles of NULL values:

- Setting a null value is appropriate when the actual value is unknown or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will evaluate to null in any expression: null multiplied by 10 is null.
- When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into that column.

Syntax

The basic syntax of **NULL** while creating a table.

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.A field with a NULL value is the one that has been left blank during the record creation.

Example: The NULL value can cause problems when selecting data. However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results. You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.

Consider the following CUSTOMERS table having the records as shown below.

```
+----+----------+-----+-----------+----------+
| 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        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

Now, following is the usage of the **IS NOT NULL**operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
  FROM CUSTOMERS
```

CONSTRAINTS AND THEIR IMPORTANCE
INTEGRITY CONSTRAINTS OVER RELATION:

What is an integrity constrain? Explain "Integrity constraints over relations".

Integrity constraint is a condition (rule) that ensures the correct insertion of the data and prevents unauthorized data access thereby preserving the consistency of the data. Example, the roll number of a student cannot be a decimal value. The database enforces the constraint that the instance of roll number can have only integer values.

The Integrity constraints can be applied in the following cases.
1. At the time of defining the database, the constraints are must be specified.
2. Validity of the data must be checked while executing the application.

The major integrity constraints are
1) Domain Constraints.
2) Key constraints
3) Referential integrity constraints
4) General Constraints.

1) **DOMAIN CONSTRAINTS** : The domain constraints are used to prevent from null values and allows specified range of values. A domain definition consists domain name, meaning, data type and size (or) length. The domain integrity constraints are classified into two types.

NULL/NOT NULL: The word „NULL" means zero and it is a default value in a column_name. The word „NOT NULL" means not zero, it is used with one or more column_names that do not accept zero"s.
1)zero and null are not equivalent.
2)One null is not equivalent to another null

Eg: **Create table bvsr_table**(**stu_id number(6) not null**);

CHECK constraint : The CHECK constraint can be specified to column_name to allow only a particular range of values. The CHECK constraint contain a condition that must satisfy in each row. These conditions governed by logical expressions (or) Boolean expression. Note: Check constraint cannot be allowed in sub-queries.

Eg: 1)**CREATE TABLE** student(sno **number(6) CHECK** (sno <=100), sname **varchar2(20)**
NOT NULL, course **varchar2(10) CHECK**(course **IN**(„btech","mtech","degree"))); 2)

1) KEY CONSTRAINTS : Key constraints are used in relations to uniquely identify the records of the relation. A Key constraint can be defined as a statement that consists of minimal subset of attribute that uniquely determine a record in a table.

The different types of key constraints are,

i)Entity Integrity Constraint : An entity represents a table and each row of table represents an instance of the entity. Each row in a table can be uniquely identified by using the entity constraint.

This constraint can be classified into two types.

a) Unique constraint

b) Primary key constraint.

a)UNIQUE constraint: It is defined with column to handle unique values. This can be used to prevent from duplication values within the rows of specified column (or) set of columns. It means, it cannot allow duplicate values in the table. But it allows NULL values. It can be defined in more than one column. (i.e up to 16 columns).

Eg: 1) **CREATE TABLE** emp_table(emp_id number(6) **unique**, emp_name char(20));

a)Primary Key : A PRIMARY KEY is a constraint that uniquely identify each row in a table. The primary key cannot accept NULL value and same value (i.e duplicates). A table can allow only one primary key. For example, the primary key for the student is defined to stu_id (or) Employee is emp_id.

Eg: **CREATE TABLE** student(sno number(6) PRIMARY KEY, sname char(20), fname char(20), fee number(7,2), date_of_join data.

ii)Candidate Key: A candidate key is a collection of fields / columns / attributes that uniquely identifies a tuple (record).

Eg: in customer relation (table) the attribute “cid” is a key that uniquely defines a tuple in a relation. No two rows in a “customer” relation of “cid” can have same value.

iii)Composite Key: Composite key consist of more than one attribute that uniquely identifies a tuple in a relation. All the attributes that form a set of keys and all of them taken together determines a unique row in a table.

Eg: set (cid, accno) is a composite key which maintains the uniqueness of each row. Both cid, accno are taken as keys.

iv) Super Key: A super key is a combination of both candidate key and composite key. That is a super key is a set of attributes or a single attribute that uniquely identifies a tuple in a relation.

Eg: consider the super key, {cid, accno, cname}

2) REFERENTIAL INTEGRITY CONSTRAINT :

It establish the relation between two tables when having a common column in both tables. A referential Integrity constrain is a rule that states the relation between primary key value and foreign key value. It means, it establishes the relation between two tables. In this case, the primary key value will be referenced in another table by a foreign key and that cannot be deleted.

The referential integrity constraints are
a) Referenced key
b)Foreign key

a) Referenced key: It is a unique (or) primary key which is defined on a column belongs to the parent table. The referential integrity constraint does not use foreign key to identify the column that make up the foreign key . The foreign key is automatically enforced on the columns.

Eg: CREATE TABLE student(sno number(5) PRIMARY KEY, sname char(20) NOT NULL, fname char(20) NOT NULL, fee number(7,2) NOT NULL, course char2(10));

b) Foreign Key : Foreign key provides links between two tables. Foreign keys are used to maintain data consistency. A foreign key is a set of attributes or a single attribute in one relation that forms a point of candidate key in other relation. A column (or) combination of columns included in the definition of referential integrity which would refer to a referenced key. The foreign key operation is only in one table

Eg: CREATE TABLE dept(deptno number(4) primary key,dname varchar2(20), loc varchar2(20); CREATE TABLE emp(emp_id number(5) primary key, ename char(20), sal number(7,2), hiredate date, deptno number(4) constraint FK FOREIGN KEY(deptno) references dept(deptno))

3) GENERAL CONSTRAINTS

Domain, primary key, foreign key constraints are considered to be a fundamental part of the relational data model and are given special attention in most systems. Sometimes, it is necessary to specify more general constraints.

Default constraint: The DEFAULT constraint is used to provide a default value for a column. The default value will be added to all new records IF no other value is specified.

Ex: create table sailors(sid number(4),sname varchar2(10),addtress varcahr2(15) default 'ONG'); In general constraints, the domain, primary key and foreign key can also specify the maximum limit.There are two types of general constraints.

1) **Table constraints:** These are applied on a particular table and are checked every time whenever that specific table is updated.

2)**Assertions:** These assertions are applied on collection of tables and are checked every time whenever these tables are updated.

BASIC SQL:

DATABASE SCHEMA IN SQL:

The schema is collection of related objects which are created by user with create command such as tables, views, domains, constraints, character sets, triggers, roles etc., In simple way, schema is nothing but plan how to store the data on secondary storage device and how to extract the from secondary storage device. SQL Commands are used to define the definition to Schema by using DDL commands, Data Definition Language Commands (DDL) : The DDL Commands are used to define database table in schema. It includes
create, alter, drop the tables and establishing constraints. These commands are restricted to production version database

DATA TYPES IN SQL

The data types are used to define the letters or words as a variable name. The variable name is in a group, is called field name or column name. The first letter in a column name must be an alphabet and next letters may be an alphabet or numbers. No special character is allowed except underscore. The column name can hold the data, carry the data and can change the data

Number data type : This data type is used to define a field as a numeric variable to hold the numeric data. The numeric data may be fixed (whole) or floating point values.

Syntax : column name number(size);

- The column name is name of field in a table to store the data.
- The word „number" is reserved word, used to define the column name as numeric type to store numeric values.
- The word „size" specifies „p" and „s" . The „p" means precision that determine maximum length and „s" means scale that determine number of decimal places in the maximum length.

Eg: 1) sno number(5); This defines the word „sno" as integer variable with size 5 to store 5 digit integer data.

2) rate number(7,2); This defines the word „rate" as floating point variable with size 7,2 to store floating point data. 5 integer places and two decimal places.

Character data types : These data types are used to define the field names as character variable to store alphanumeric data.

Char data type: This data type is used to define a field name as character variable to store alphanumeric data. (Maximum size of data 27 bits=255). CHAR on the other hand is store fixed length character data.

Syntax : column name char(size);

- The column name is name of field in a table to store the data.
- Fixed-length character data of length *size* bytes.

- The word „char“ is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word „size“ specifies maximum number of characters can stored in column name.

Eg: sname char(20); It defines the word „sname“ as character variable to store alphanumeric data.

Nchar data type: Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set. Nchar is used to store fixed length Unicode data. It is often used to store data in different languages

Varchar data type: This data type is used to define a field name as character variable to store alphanumeric data up to maximum 2000. VARCHAR is reserved by Oracle to support distinction between NULL and empty string in future, as ANSI standard prescribes

Syntax : column name varchar(size);
- The column name is name of field in a table to store the data.
- The word „varchar“ is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word „size“ specifies maximum number of characters can stored in column name.

Eg: sname varchar(20); It defines the word „sname“ as character variable to store alphanumeric data.

Varchar2 data type: This data type is used to define a field name as character variable to store alphanumeric data up to maximum 2000. VARCHAR2 does not distinguish between a NULL and empty string, and never will.

Syntax : column name varchar2(size);
- The column name is name of field in a table to store the data.
- The word „varchar2“ is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word „size“ specifies maximum number of characters can stored in column name.

Eg: sname varchar2(20); It defines the word „sname“ as character variable to store alphanumeric data.

Date data type : This data type used to define the column name as date type to date in the format DD-MM-YY.

Syntax : column name date;

 The column name is name of field in a table to store the date.
 The word „date‟ is reserved word that define a column name as date variable to store date.

Eg: dob date; It defines the word date of birth as date variable to the date in the format.

LONG data type : This data type is used to define a column name as character type variable to store alphanumeric data up 2GB.

RAW data type : This data type is used to define a column name as binary variable to store binary data upto 255 bytes. The binary data may be digitized picture or image.

LONG RAW data type : This data type is used to define a column name as binary variable to store binary data upto 2GB.

TABLE DEFINITIONS:

- SQL Table is a collection of data which is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple.
- Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

CREATING THE TABLES:

The tables are created in SQL by CREATE command. The CREATE command is used to create a table with column_names. The column_name are called variables or field names.

To create a table, must follow the below steps.

1. Identify the appropriate data type, including length, precision, and scale for each attribute if required.
2. Identify the columns that should accept null values.
3. Identify the columns that need to be unique.
4. Identify the primary key and foreign key for attributes.
5. Determine the values to be inserted in any column for which a default value is specified.
6. Identify any column for which domain specifications may be stated. Eg: CHECK
7. Create index using existed table.

Syntax: CREATE TABLE table_name(column_name1 data type1 [<constraint_type attribute_ name>], column_name2 data type2 [<constraint_type attribute_ name>] ----- ------ [CONSTRAINT constraint_name]);

 The table_name is name of table to store the data in secondary memory.
 The column_name is name of filed name, used to store the data.
 The data type is used to define column_name to store related data. The data types are number(), char(), varchar2() and date.
 The constraint_types are used with column_name to specify the constraint type.

i) PRIMARY KEY
ii) FOREIN KEY
iii) UNIQUE KEY

iv) REFERENCE
v) NOT NULL
vi) CHECK.

Eg: CREATE TABLE student(sno number(5),sname char(20), fname char(20),fee number(7,2),date_of_join date);

Performing the operations on table definitions: Once the table is created, the user can perform operations on table. They are adding the coulmn_names, change the size of column_name, and can remove the column_name and table. These operations are done by using ALTER command, DESC command, DROP command etc.

ALTER command: The ALTER command is used to change the definition of a table.

Syntax: ALTER TABLE table name MODIFY (column_name data type(size)); DROP RENAME

ADD clause : The ADD is a keyword, used to add a column_name and/or constraints to an existing table.

Syntax : ALTER TABLE table_name ADD(new column_name data type); Eg: ALTER TABLE student ADD(date_of_join date);

MODIFY clause : The MODIFY is a keyword, used to modify the definition of an existing column_name. The MODIFY keyword in the ALTER command cannot make following three changes.

i) can not change the specification of column_name from NULL to NOT NULL when column_name containing nulls.
ii) can not decrease the size of a column_name (or) data type when column_name contain a data. iii)can not define constraints on a column_name except NULL/NOT NULL. Ex: alter table student modify sno number(4);

DROP clause : It is used with ALTER command to remove a constrained from a table. Syntax : ALTER TABLE table_name DROP constraint constraint_name;
Eg: ALTER TABLE emp DROP constraint PK;

RENAME clause: it is used to rename the column name Syntax: alter table table_name rename column old column name to new column name;
Ex: alter table student rename column date_of_join to doj;

DESC command : This command is used to display the definition of an existed table. I.e., it will display the structure of specified table. Syntax : DESC table_name;
Eg: DESC student;

DROP command: This command is used to delete the table name from the secondary memory.

DIFFERENT DML OPERATIONS (INSERT, DELETE ,UPDATE)

How to insert, update, and delete Cloud Spanner data using Data Manipulation Language (DML) statements. You can run DML statements using the client libraries, the Google Cloud Console, and the gcloud command-line tool. You can run Partitioned DML statements using the client libraries and the gcloud command-line tool.

For the complete DML syntax reference, see Data Manipulation Language syntax.

Using DML

DML supports INSERT, UPDATE, and DELETE statements in the Cloud Console, gcloud command-line tool, and client libraries.

ANTI-PATTERN: SENDING AN UPDATE WITHOUT THE PRIMARY KEYCOLUMNIN THE WHERE CLAUSE

```
UPDATE Singers SET FirstName = "Marcel"
WHERE FirstName = "Marc" AND LastName = "Richards";
```

To make the update more efficient, include the SingerId column in the WHERE clause. The SingerId column is the only primary key column for the Singers table:

RECOMMENDED: INCLUDING THE PRIMARY KEY COLUMN IN THE WHERE CLAUSE

```
UPDATE Singers SET FirstName = "Marcel"
WHERE FirstName = "Marc" AND LastName = "Richards" AND SingerId = 1;
```

Concurrency

Cloud Spanner sequentially executes all the SQL statements (SELECT, INSERT, UPDATE, and DELETE) within a transaction. They are not executed concurrently. The only exception is that Cloud Spanner might execute multiple SELECT statements concurrently, because they are read-only operations.

Transaction limits

A transaction that includes DML statements has the same limits as any other transaction. If you have large-scale changes, consider using Partitioned DML.

- If the DML statements in a transaction result in more than 20,000 mutations, the DML statement that pushes the transaction over the limit returns a BadUsage error with a message about too many mutations.
- If the DML statements in a transaction result in a transaction that is larger than 100 MB, the DML statement that pushes the transaction over the limit returns a BadUsage error with a message about the transaction exceeding the size limit.

Mutations performed using DML are not returned to the client. They are merged into the commit request when it is committed, and they count towards the maximum size limits. Even if the size of the commit request that you send is small, the transaction might still exceed the allowed size limit.

Running statements in the Cloud Console

Use the following steps to execute a DML statement in the Cloud Console.

1. Go to the Cloud Spanner **Instances** page.
2. Select your project in the drop-down list in the toolbar.
3. Click the name of the instance that contains your database to go to the **Instance details** page.
4. In the **Overview** tab, click the name of your database. The **Database details** page appears.
5. Click **Query**.
6. Enter a DML statement. For example, the following statement adds a new row to the Singers table.

```
INSERT Singers (SingerId, FirstName, LastName)
    VALUES (1, 'Marc', 'Richards')
```

7. Click **Run query**. The Cloud Console displays the result..

Executing statements with the gcloud command-line tool

To execute DML statements, use the gcloud spanner databases execute-sql command. The following example adds a new row to the Singers table.

```
gcloud spanner databases execute-sql example-db --instance=test-instance \
    --sql="INSERT Singers (SingerId, FirstName, LastName) VALUES (1, 'Marc',
'Richards')"
```

Modifying data using the client library

To execute DML statements using the client library:

- Create a read-write transaction.
- Call the client library method for DML execution and pass in the DML statement.
- Use the return value of the DML execution method to get the number of rows inserted, updated, or deleted.

The following code example inserts a new row into the Singers table.

C++C#GoJavaNode.jsPHPPythonRuby

You use the ExecuteDml() function to execute a DML statement.

View on GitHub

```
void DmlStandardInsert(google::cloud::spanner::Client client) {
    using ::google::cloud::StatusOr;
    namespace spanner = ::google::cloud::spanner;
    std::int64_t rows_inserted;
    auto commit_result = client.Commit(
      [&client, &rows_inserted](
        spanner::Transaction txn) -> StatusOr<spanner::Mutations> {
       auto insert = client.ExecuteDml(
         std::move(txn),
         spanner::SqlStatement(
           "INSERT INTO Singers (SingerId, FirstName, LastName)"
           "  VALUES (10, 'Virginia', 'Watson')"));
       if (!insert) return insert.status();
       rows_inserted = insert->RowsModified();
       return spanner::Mutations{};
      });
    if (!commit_result) {
      throw std::runtime_error(commit_result.status().message());
    }
    std::cout << "Rows inserted: " << rows_inserted;
    std::cout << "Insert was successful [spanner_dml_standard_insert]\n";
  }
```

The following code example updates the MarketingBudget column of the Albums table based on a WHERE clause.

BASIC SQL QUERY(SELECT AND PROJECT)USING WHERE CLAUSE:

SQL (*Structured Query Language*) is used to perform operations on the records stored in the database such as updating records, deleting records, creating and modifying tables, views, etc

The syntax of a simple SQL query and explains its meaning through a conceptual Evaluation strategy. A conceptual evaluation strategy is a way to evaluate the query that is intended to be easy to understand rather than efficient. A DBMS would typically execute a query in a different and more efficient way.

The basic form of an SQL query is as follows:

1) **SELECT [DISTINCT] select-list**
2) FROM from-list
3) WHERE qualification

Query must have a SELECT clause, which specifies columns to be retained in the result, and a FROM clause, which specifies a cross-product of tables. The optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

Such a query intuitively corresponds to a relational algebra expression involving selections, projections, and cross-products. The close relationship between SQL and relational algebra is the basis for query optimization in a relational DBMS, execution plans for SQL queries are represented using a variation of relational algebra expressions

we will present a number of sample queries using the following table definitions:
Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves (sid: integer, bid: integer, day: date

Create the sailors Table:

CREATE TABLE sailors (sid number(3),sname varchar2(12),rating number(2),age number(3,1), CONSTRAINT PK_sailors PRIMARY KEY (sid));

Create the boats Table:

CREATE TABLE boats(bid number(3) primary key,bname varchar2(12),color varchar2(10);

Create the reserves Table:

CREATE TABLE reserves (sid number(3),bid number(3),day date, CONSTRAINT PK_reserves PRIMARY KEY (sid, bid, day),

FOREIGN KEY (sid) REFERENCES sailors(sid),
FOREIGN KEY (bid) REFERENCES boats(bid));

Sailors

Sid	Sname	Rating	Age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	40
85	Art	3	25.5
95	Bob	3	63.5

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves

sid	bid	day
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-8
22	104	1998-10-7
31	102	1998-11-10
31	103	1998-11-6
31	104	1998-11-12
64	101	1998-9-5
64	102	1998-9-8
74	103	1998-9-8

SQL>SELECT S.sid, S.sname, S.rating, S.age FROM Sailors AS S WHERE S.rating > 7

This query uses the optional keyword AS to introduce a range variable. Incidentally, when we want to retrieve all columns, as in this query, SQL provides convenient shorthand: we can simply write SELECT *. This notation is useful for interactive querying, but it is poor style for queries that are intended to be reused and maintained because the schema of the result is not clear from the query itself; we have to refer to the schema of the underlying Sailors table.

• The from-list in the FROM clause is a list of table names. A table name can be followed by a range variable; a range variable is particularly useful when the same table name appears more than once in the from-list

. • The select-list is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.
• The qualification in the WHERE clause is a Boolean combination (i.e., an expression using the logical connectives AND, OR, and NOT) of conditions of the form expression op expression, where op is one of the comparison operators {<, <=, =, <>, >=, >}.2 An expression is a column name, a constant, or an (arithmetic or string) expression.
• The DISTINCT keyword is optional. It indicates that the table computed as an answer to this query should not contain duplicates, that is, two copies of the same row. The default is that duplicates are not eliminated.
The syntax of a basic SQL query, they do not tell us the meaning of a query. The answer to a query is itself a relation which is a multi set of rows in SQL considering the following conceptual evaluation strategy:

1. Compute the cross-product of the tables in the from-list.
2. Delete rows in the cross-product that fail the qualification conditions.
3. Delete all columns that do not appear in the select-list.
4. If DISTINCT is specified, eliminate duplicate rows.

ARITHMETIC OPERATIONS&LOGICAL OPERTIONS:

What are SQL operators?

SQL operators are reserved keywords used in the WHERE clause of a SQL statement to perform arithmetic, logical and comparison operations. Operators act as conjunctions in SQL statements to fulfill multiple conditions in a statement.

Since, there are different types of operators in SQL, let us understand the same in the next section of this article on SQL operators.

Types of SQL Operators

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators

Arithmetic Operators

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	Operation	Description
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value

*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Example:

```
1SELECT 40 + 20;

2

3SELECT 40 - 20;

4

5SELECT 40 * 20;

6

7SELECT 40 / 20;

8

9SELECT 40 % 20;
```

Well, that was about the arithmetic operators available in SQL. Next in this article on SQL operators, let us understand the comparison operators available.

Logical Operators

The logical operators are used to perform operations such as ALL, ANY, NOT, BETWEEN etc.

Operator	**Description**
ALL	Used to compare a specific value to all other values in a set
ANY	Compares a specific value to any of the values present in a set.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.

OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.
SOME	Similar to the ANY operator, and is used compares a specific value to some of the values present in a set.

Example:

I am going to consider the Students table considered above, to perform a few of the operations.

Example[ANY]

```
1SELECT * FROM Students

2WHERE Age > ANY (SELECT Age FROM Students WHERE Age > 21);
```

Output:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
5	Vaibhav	Gupta	25

Example[BETWEEN & AND]

```
1SELECT * FROM Students

2WHERE Age BETWEEN 22 AND 25;
```

Output:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
4	Akanksha	Jain	20

examples on the different types of operators to get good practice on writing SQL queries.

SQL FUNCTIONS

A function is a process that will manipulates data values and returns the results. It is similar to an operator. But it should contain the argument. The argument has a constant value.

Rules:

1. If you call a function with an argument, ORACLE converts the argument into the expected **type**.
   Example, Floor(15.7) returns 15. Here, input is float type and output is integer type.

2. If you call a function with a null argument, the function automatically returns null value

Eg: ABS(-5), its absolute value is 5. **Syntax** : SELECT function_name(argument1, argument2 . . . .) FROM DUAL;

The function type is keyword such as ABS(), FLOOR(), MIN(), MAX() etc.

Syntax : SELECT function_name(argument1, argument2 . . . .) FROM DUAL;

- The function type is keyword such as ABS(), FLOOR(), MIN(), MAX() etc.
- The argument is any operand (or) value (or) more than one value.

Types of functions :

The SQL functions are divided into two types based on the function process.

I). Single row functions
II). Group functions

1. Single row functions :

A single row functions perform the process based on the input value and return the output value. In this case, it takes input one type of constant and produces another type.

Eg: is float type and output is integer. The single row functions must use the keyword **dual**.

The single row functions are

i) Number
ii) Character functions
iii) Date functions
iv) Conversion functions
v) Other functions

1)Number functions :

Number functions accept numeric value as input and return numeric value as output. Most of these functions return values that are accurate to 38 decimal digits.

The functions are

1) **ABS (Absolute Value) :** This function returns the absolute value of n.
   Syntax : ABS(n) Where n is numeric value.
   Eg: SELECT ABS(-15) FROM DUAL; It will display the Absolute value as 15.

2) **CEIL (Ceiling) :** This function requires float value and returns the integer value. Suppose, the value after decimal point is greater than 0 then it will add one to integer part.
   Syntax : CEIL(n) Where n is numeric value.
   Eg: SELECT CEIL(15.7) FROM DUAL;

3)COS(cosine) : This function is used to returns the cosine of n Syntax : COS(n) Where n is numeric value.

Eg: SELECT COS(180) FROM DUAL; It will display the output value as -.5984601

4)COSH(cosine hyperbolic):This function is used to returns the hyperbolic cosine of n Syntax : COSH(n) Where n is numeric value.

Eg: SELECT COSH(0) FROM DUAL; It will display the output value as 1

5)EXP(exponentiation) : This function is used to returns e to the power of n Where e is 2.71828183
Syntax : EXP(n) Where n is numeric value
. Eg: SELECT EXP(4) FROM DUAL;
It will display the e to the 4th power as 54.59815

6)FLOOR() : This function is used to returns the integer i.e. it will drop the decimal part .
Syntax : FLOOR(n) Where n is numeric value.
Eg: SELECT FLOOR(15.7) FROM DUAL; It will display the output value as 15.

7) LN(Logarithm to base E) : This function is used to returns the natural logarithm of n. where n is greater than 0.
Syntax : LN(n) Where n is numeric value.
Eg: SELECT LN(95) FROM DUAL; It will display the natural log of 95 as 4.5538769

8) LOG(Logarithm to base 10): This function is used to returns the logarithm, base m of n. The base „m" can be any positive number other than 0 or 1 and „n" can be any positive number. Syntax:LOG(m,n) Where „m" can be positive number and „n" can be any positive number.
Eg: SELECT LON(10,100) FROM DUAL; It will display the log base 10 of 100 as 2.

9) MOD(modulus): This function is used to returns the remainder of m divided by n. It returns m if n is 0.
Syntax: MOD(m,n) Where „m" and „n, are numeric values
. Eg: **SELECT MOD(45,12) FROM DUAL**; It will display the remainder value of m/n as 9.

11) POWER(): This function is used to returns the m to the power of n. Syntax: POWER(m,n) Where „m" and „n, are numeric values. Eg: SELECT POWER(3,2) FROM DUAL; It will display the 3 to the power of 2 is as 9.

12) ROUND(): This function is used to return the n places with rounding from the value m. The m must be the decimal point. If n is omitted, it return the 0 places. The n must be an integer.
Syntax: ROUND(m,n) Where „m" and „n, are numeric values.
Eg: SELECT ROUND(15.193,

1) FROM DUAL; It will display the output as 15.2.
   Eg: SELECT ROUND(15.193,
2) FROM DUAL; It will display the output as 15.19
   Eg: SELECT ROUND(15.193,3) FROM DUAL; It will display the output as 15.193.
   Eg: SELECT ROUND(15.193,-1) FROM DUAL; It will display the output as 20.

13) SIGN(): This function is used to returns -1 if n < 0, returns **0** if n = 0, (or)returns 1 if n > 0.
Syntax: SIGN(n) Where „n" is numeric value.
Eg: SELECT SIGN(-15) FROM DUAL; It will display the output as -1 because –15 is less than zero.

14) SIN(): This function is used to returns sine value of n.
Syntax: SIN(n) Where „n" is numeric value.
Eg: SELECT SIN(60) FROM DUAL; It will display the output as -.3048106.

15) SINH(): This function is used to returns hyperbolic sine value of n. Syntax: SINH(n) Where „n" is numeric value.
Eg: SELECT SINH(1) FROM DUAL; It will display the output as 1.1752012.

16) SQRT(): This function is used to returns the square root of n. The n value must not be negative. Sqrt returns a "real" result.
Syntax: SQRT(n) Where „n" is numeric value.
Eg: SELECT SQRT(16) FROM DUAL; It will display the Square root of 16 as 4.

17) TAN(): This function is used to returns the tangent of n.
Syntax: TAN(n) Where „n" is numeric value.
Eg: SELECT TAN(90) FROM DUAL; It will display the tangent value of 90 as -1.9952.

18) TANH(): This function is used to returns the hyperbolic tangent of n.
Syntax: TANH(n) Where „n" is numeric value.
Eg: SELECT TANH(90) FROM DUAL; It will display the hyperbolic tangent value of 90 as 1.

19) TRUNC(): This function is used to truncate the decimal values and display specified number of decimal places from the given number.
Syntax: TRUN(m , n) Where „m" and „n" are numeric values.
Eg: SELECT TRUN(15.79655,1) FROM DUAL; It will truncate the decimal values based on „n" value. Here n is 1. i.e. 15.7 Eg: SELECT TRUN(15.79655,3) FROM DUAL; It will truncate the decimal values based on the „n" value. Here n is 3 i.e. 15.796 Eg: SELECT TRUN(15.79655,4) FROM DUAL; It will truncate the decimal values based on the „n" value. Here „n" is 4. i.e. 15.7965

II) **CHARACTER FUNCTIONS**:

This function is used accept single row character as input and can return both character and number values. The character type value (or) column may be defined with the keyword char or varchar2.
The character functions are

1**) CHR () :** This function is used to returns the character value.
I.e. ASCII character.
Syntax : chr(n) where n is integer number.
Eg: SELECT CHR(75) FROM DUAL; It will the Ascii character of 75 as „K".

2) CONCAT() : This function is used to join the two character strings
. Syntax: CONCAT(char1,char2) The char1 and char2 are character data or character variables.
Eg: SELECT CONCAT(„vasu‟,‟ reddy‟) FROM DUAL; It will concatenate (join) the two words „vasu‟ and „reddy‟ as „vasu reddy‟ Eg: SELECT CONCAT(empname,‟ is a boy‟) FROM EMP; It will concatenate employee name with „s a person‟ . For example, if empname is hari means, it will display as hari is a person.

3)INITCAP() : This function is used to display the first letter of each word in uppercase, all other letters in lowercase. It is allowed in alphabetic characters only. Syntax: INITCAP(char) The char must be alphabetic data only.
Eg: SELECT INITCAP(„the indian lady‟) FROM DUAL; It will display the first letter of each word in upper case. I.e. it will display as „The Indian Lady‟

4)LOWER() : This function is used to display all letters in lower case. Syntax: LOWER(char) The char must be alphabetic type data only.
Eg: SELECT LOWER(„INDIA‟) FROM DUAL; It will display the upper case INDIA in lower case letters as „india‟

5)LPAD() : This function is used to display the specified symbol if „n‟ is more than the given character data. Syntax: LPAD(char) The char is alphanumeric type data.
Eg: SELECT LPAD(„INDIA‟,5,‟*‟) FROM DUAL; It will display word as INDIA only.

6)LTRIM() : This function is used to remove the characters from the left of character data. That is, if the specified right most characters are same of first two characters in the character data.
Syntax: LTRIM(char) The char is alphanumeric type data.
Eg: SELECT LTRIM(indian‟,‟in‟) FROM DUAL; It will remove first two character and display output as „dian‟. Eg: SELECT LTRIM(„indian‟,‟ab‟) FROM DUAL; It will not remove first two character, because the given characters are not same as the first two characters. So, the output is indian.

7) NLS_INITCAP() : This function is used to display the first character in upper case for all words which in the character data. It is similar to INITCAP() function. Syntax: NLS_INITCAP(char) The char is alphabetic only. Eg: SELECT NLS_INITCAP(„india is my country‟) FROM DUAL; It will display the output as follows. „India Is My Country. In this first letter upper case letter.

8)NLS_LOWER() : This function is used to display all characters in lower case. It is same as the LOWER() function.
Syntax: NLS_LOWER(char) The char must be alphabetic type data only.
Eg: SELECT NLS_LOWER(„INDIA‟) FROM DUAL; It will display the upper case word „INDIA‟ in lower case letters as „india‟

9)NLS_UPPER() : This function is used to display all characters in upper case. It is same as the UPPER() function.

Syntax: NLS_UPPER(char) The char must be alphabetic type data only.

Eg: SELECT NLS_UPPER(„ india ‟) FROM DUAL; It will display the lower case word „inida‟ in upper case letters as „INDIA‟

10)REPLACE function() : This function is used to replace the letter or word with given letter or word.

Syntax: REPLACE(char) The char is alphanumeric data.

Eg: SELECT NLS_UPPER(„ india ‟) FROM DUAL; It will display the lower case word „inida‟ in upper case letters as „INDIA‟

11)REPLACE function() : This function is used to replace the letter or word with given letter or word.

Syntax: REPLACE(char) The char is alphanumeric data

. Eg: SELECT REPLACE(„ India is my Country‟,‟India‟,‟INDIA‟) FROM DUAL; It will display the out put as follows. „INDIA is my Country‟ In this, the first word „India‟ is replaced by „INDIA‟. Eg: SELECT REPLACE(„ NSS students are obediance‟,‟NSS‟,‟SSN‟) FROM DUAL; It will display the input „NSS students are obedience‟ as „SSN students are obedience‟.

12)RPAD function() : This function is used to replicate the given character a specified number of time. That is, if the given number is grater than the stirng(character data) it will print the given character.

Syntax: RPAD(char) The char is alphanumeric data.

Eg: SELECT REPLACE(„ India‟,8,‟*‟) FROM DUAL; It will display the out put as follows. „India***‟ It displays 3 stars at right side of the string. Because the number is greater than the string(india).

12) RTRIM() : This function is used to remove the characters from the right of character data. That is, if the specified characters are same of last characters in the character data then it removes those characters.

Syntax: RTRIM(char) The char is alphanumeric type data. Eg: SELECT RTRIM(„Indian‟,‟an‟) FROM DUAL; It will remove last two character and display output as „Indi‟.

Eg: SELECT RTRIM(„Indian‟,‟ab‟) FROM DUAL; It will not remove last two character, because the given characters are not same as the first two characters. So, the output is Indian.

13) SUBSTR() : This function is used to display the character from specified position to the specified number characters in the given character data.

Syntax: SUBSTR(char,pos, n)

The char is alphanumeric type data.

The pos is position in the given data.

The n is number that specifies number characters from the position.

Eg: SELECT SUBSTR(„SSN COLLEGE‟,5,7) FROM DUAL; It will display the character data from 5th position to 7 characters. I.e. „COLLEGE‟ Eg: SELECT SUBSTR(„SSN COLLEGE‟,1,3) FROM DUAL; It will display the character data from 1st position to 3 characters. I.e. „SSN‟

14) UPPER() : This function is used to display all characters in upper case. It is same as the NLS_UPPER() function.
Syntax: UPPER(char) The char must be alphabetic type data only.
Eg: SELECT UPPER(„ india ‟) FROM DUAL; It will display the lower case word „inida‟ in upper case letters as „INDIA‟

15) ASCII() : This function is used to convert the given character into equivalent ascii code.

Syntax: ASCII(char) The char must be alphabetic type data only. Eg: SELECT ASCII(„A‟) FROM DUAL; It will display the equivalent ASCII value 65 for given character „A‟
Eg: SELECT ASCII(„a‟) FROM DUAL; It will display the equivalent ASCII value 97 for given character „a‟

16) INSTR() : This function is used to display the specified position character at what location in the given character data.
Syntax: INSTR(char,pos, n) The char is any alphabetic type data. The pos is used to specify that the character is at what position. The „n‟ specifies how many times the character is in the data.

Eg: SELECT INSTR(„we are all ssn studnets‟,‟s‟,12,2) FROM DUAL;
It will display the equivalent ASCII value 65 for given character „A‟ Eg: SELECT ASCII(„a‟) FROM DUAL.

17) LENGTH() : This function is used to display the length of given character data (string). It count the null space also.
Syntax: LENGTH(char) The char is any alphanumeric type data.
Eg: SELECT LENGTH(„ssn college‟) FROM DUAL; It will display the length of character data (string). I.e. 11.

III)DATE functions : Date functions operate on values of DATE data type. All date functions return a value of DATE data type, except the MONTHS_BETWEEN function, which returns a number.

1) **ADD_MONTHS()** : This function is used to return the DAY plus n months. It means, return a date by adding specified number of months. For example, the give date is „17-dec-08‟, it will be display as „17-jan-81‟ when we added „one‟ to it.

   Syntax : ADD_MONTHS(d,n) The d may be date (or) date variable. The n is the integer number, it is used to add number of months to the month.
   Eg : select hiredate from emp; It will display 14 rows of column hiredate values.
   Eg: select add_months(hiredate,1) from emp; It will display 14 rows of column hiredate values from emp with adding one month Eg: select

add_months(hiredate,3) from emp; It will display 14 rows of column hiredate alues from emp with adding 3 months

2)LAST_DAY(): This function returns the date corresponding to the last day of the months. Suppose, the system date is 17-sep-08. The Last day of this month is 31-sep-08.
Syntax : Last_day(Column_name); The column_name must be date type.
Eg: SELECT sysdate, last_day(sysdate) from dual; It will display system date and last day of today system date. I..e Today date is 17-sep-08 then last day is 31-sep-08.

3)MONTHS_BETWEEN(): This function returns a number of months between dates i.e date1 and date2. If date 1 is later than date2,then result is negative. If the date1 is earlier than date2, result is negative.
Syntax: MONTHS_BETWEEN(date1,date2) The date1 and date2 are date type.
Example, 1)select months_between(„17-sp-08“,“1-oct-08“) from dual;
2)select months_between(hiredate,sysdate) from emp;

4)NEXT_DAY(): This function is used to return the date of the first week day. It means later than the date „d“. Syntax: NEXT_DAY(d,char)

- The „d“ is date value (or) date type identifier (or) sysdate keyword.
- The char is any week day such as „Monday“,“Tuesday“, ...

This day must later day of „d“
Eg: Select next_day(„17-sep-08“,“Thursday“) from dual;

- In this „17-sep-08“ is today date and its day is „Wednesday“ . But, it will display the next day of Thursday. That is „Friday“ . So, it will display Friday date as „19-sep-08“.

5)NEW_TIME(): This function is used to return the date and time.
Syntax: NEXT_TIME(d,z1,z2)

- The „d“ is date value (or) date type identifier (or) sysdate keyword.
- The z1 and z2 are time zones
- The z2 is used to display the date of z1 format in z2 format.

i.e if z1 is AST format ad z2 is CST format, then the date of z1 format will be displayed in z2 format.

Time zones are

1. AST/ADT -> Atlantic standard
2. BST/BDT -> Bearing standard
3. CST/CDT -> Central standard
4. EST/EDT -> Eastern standard
5. GMT -> Green which mean Time
6. HST/HDT -> Alaska – Hawli standard Time
7. MST/MDT -> Mountain standard time
8. NST -> New found Land standard time
9. PST/PDT -> Pacific standard time

10. YST/YDT -> Yukan standard time

6) ROUND(): This function returns the date which rounded to the unit specified by the format „fmt“ . It means, it round to the nearest day. Syntax: ROUND(d, fmt)

- The letter „d“ represents date.
- The word „fmt“ represents year, month (or) day

Eg: 1)SELECT hiredate, ROUND(hiredate,“year“) from emp;
Eg: 2)SELECT hiredate, ROUND(hiredate,“month“) from emp;
Eg: 3)SELECT hiredate, ROUND(hiredate,“day“) from emp;
Eg: 4)SELECT ROUND(sysdate) from dual;
Eg: 5)SELECT ROUND(sysdate,“year“) from dual;
Eg: 6)SELECT ROUND(sysdate,“month“) from dual;
Eg: 7)SELECT ROUND(sysdate,“day“) from dual;

7)SYSDATE: This keyword is used to return the current date and time. Syntax : SYSDATE;
Eg: SELECT SYSDATE FROM DUAL; Eg: SELECT SYSDATE „mm-dd-yyyy hh24“ NOW FROM DUAL;

8) TRUNC() : This function is used to returns date with the time portion of the day truncated which is specified by format „fmt“. If „fmt“ is omitted, the date is converted to the nearest day.
For example, if the SYSDATE is „27-jan-99“, then truncated result is 01-jan-99. It means, 27 truncated as 01 in the date. Syntax: TRUNC(d, fmt)
Eg: 1)SELECT TRUNC(sysdate,“year“) from dual;
Eg: 2)SELECT TRUNC(sysdate,“month“) from dual;
Eg: 3)SELECT TRUNC(sysdate,“day“) from dual;
Eg: 4)SELECT TRUNC(hiredate,“year“) from emp;
Eg: 5)SELECT TRUNC(hiredate,“month“) from emp; Eg:

IV : CONVERSION FUNCTIONS : The conversion functions are used to convert a value from one „data type“ to another. It means, char type to number type and so on. The Conversion functions are

1) TO_CHAR(): The function is used to convert date into varchar2 type in the default date format.

Syntax : TO_CHAR(date,[fmt]);
Eg: 1)SELECT TO_CHAR(sysdate,“dd month yyyy“) from dual; It will display system date as 18 September 2008.
Eg: 2)SELECT TO_CHAR(sysdate,“yyyy “years” mm “month” dd “days”“) from dual; It will display system date as 2008 years September month 18 days.

2) TO_NUMBER():This function allows the conversion of string (varchar2) type data (number) into number type in the specified format.

Syntax: TO_NUMBER(char[,fmt]);
 The word „char“ is character type data (or) variable

 The word „fmt“ is optional that represent format (i.e. AST/BST/CST etc.,

Eg: SELECT TO_NUMBER(„100“) FROM DUAL;

3) TO_DATE() :This function is used to converts **character type** data into **date type** data.

Syntax: TO_DATE(char[,fmt]);
 The word „char“ is character type data (or) variable
 The word „fmt“ is optional that represent format (i.e. AST/BST/CST etc.,

Eg: SELECT („September 18 2008“ , „ month-dd-yyyy“) from dual.

4)ROWIDTOCHAR() :This function is used to convert **ROWID** value into **character type** value(I.e varchar2 type).
Syntax: ROWIDTOCHAR(rowid);

- The word „rowid“ is a keyword that generate numbers automatically at the time of adding rows into table.

Eg: SELECT ROWIDTOCHAR(rowid) from dual/emp;