**Database**:

-Collection of organized data, store and retrieved digitally from remote or a local computer system.

**DBMS**:

-software responsible for creation/retrieval/update/management of the database.

**RDBMS vs. DBMS**:

-RDBMS stores data in the form of a collection of tables and relations can be defined between the common fields of these tables.

-MySQL, Microsoft SQL Server, Oracle, IBM DB2, Amazon Redshift are based on RDBMS.

**SQL**:

-Structured Query Language. Standard language for RDBMS.

**SQL vs. MYSQL**:

-SQL is standard language for retrieving and manipulating structured databases.

-On the contrary, MySQL is a RDBMS used to manage SQL databases

**Tables and Fields**:

-Tables are organized collections of data stored in the form of rows and columns.

-Columns are called fields and Rows are records.

**Constraints**: specifying the rules concerning data in the table.

NOTNULL -> Restricts NULL value from being inserted into a column

CHECK -> Verifies that all calues in a field satisfy the condition

DEFAULT -> Automatically assigns a default value if no value has been specified for the field

UNIQUE -> Ensures unique values to be inserted into the field

INDEX -> Indexes a field providing faster retrieval of records

PRIMARY KEY -> Uniquely identifies each record in a table

FOREIGN KEY -> Ensures referential integrity for a record in another table.

**Primary Key**:

-constraint that uniquely identifies each row in a table.

-must contain UNIQUE values and has an implicit NOT NULL constraint

-a table in SQL is strictly restricted to have one and only primary key, which is comprised of one or many columns.

\*\*Create table with a single field as primary key \*\*

CREATE TABLE Students (

ID int NOT NULL

Name varchar (255)

PRIMARY KEY (ID)

)

\*\* Create table with multiple field as primary key \*\*

CREATE TABLE Students (

ID int NOT NULL

LastName varchar (255)

FirstName varchar (255)

CONSTRAINT PK\_Student

PRIMARY KEY (ID, FirstName)

)

\*\* Set a column as primary key \*\*

ALTER TABLE Students

ADD PRIMARY KEY (ID)

\*\* Set multiple key as primary key \*\*

ALTER TABLE Students

ADD CONSTRAINT PK\_Student

PRIMARY KEY (ID, FirstName);

**Unique Constraint**:

-ensures that all values in a column are different.

-helps identify each row uniquely.

-there can me multiple unique key in a row.

\*\* Create table with single field as unique \*\*

CREATE TABLE Students (

ID int NOT NULL UNIQUE

Name varchar(255)

)

\*\* create table with multiple field as unique \*\*

CREATE TABLE Students(

ID int NOT NULL

LastName varchar(255)

FirstName varchar(255)

CONSTRAINT PK\_Student

UNIQUE (ID, FirstName)

)

\*\* Set a column as unique \*\*

ALTER TABLE Students

ADD UNIQUE (ID)

\*\* Set multiple key as unique \*\*

ALTER TABLE Students

ADD CONSTRAINT PK\_Student

UNIQUE (ID, FirstName);

**Foreign Key**:

-comprises of single/collection of fields in a table that refer to the PRIMARY KEY in another table

-ensures referential integrity in the relation between two tables

-table with the foreign key constraint is labelled as child table

-table containing the candidate key is labelled as the referenced or parent table

\*\* Create table with foreign key: Method-1\*\*

CREATE TABLE students (

ID int NOT NULL

Name varchar(255)

LibraryID int

PRIMARY KEY (ID)

FOREIGN KEY (Library\_ID) REFERENCES Library(LibraryID)

)

\*\* Create table with foreign key: Method-2\*\*

CREATE TABLE Students (

ID int NOT NULL PRIMARY KEY

Name VARCHAR(255)

LibraryID int FOREIGN KEY (Library\_ID) REFERENCES Library(LibraryID)

)

\*\* Add a new foreign key \*\*

ALTER TABLE Students

ADD FOREIGN KEY (LibraryID)

REFERENCES Library(LibraryID)

**Join**:

-SQL JOIN clause is used to combine rows from two or more tables based on a related column between the two.

**Types of JOINs**:

* ***INNER JOIN***
  + Retrieves records that have matching values in both the tables involved in the join

SELECT \*

FROM Table\_A

JOIN Table\_B

Or,

SELECT \*

FROM Table\_A

INNER JOIN Table\_B

* ***LEFT OUTER JOIN***
  + Retrieves all the record/rows from the left and the matched records/rows from the right table

SELECT \*

FROM Table\_A A

LEFTJOIN TABLE\_B B

ON A.col = B.col

* ***RIGHT OUTER JOIN***
  + Retrieves all the record/rows from the right and the matched records/rows from the left table

SELECT \*

FROM TABLE\_A A

FULL JOIN TABLE\_B B

ON A.col = B.col

* ***FULL OUTER JOIN***
  + Retrieves all the records where there is a match in either left or right table

SELECT \*

FROM TABLE\_A A

FULL JOIN TABLE\_B B

ON A.col = B.col

**Self-Join**:

-A case of regular join where a table is joined to itself based on some relation between its own column(s).

-Self Join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

SELECT A.emp\_id AS “Emp\_ID”, A.emp\_name AS “Employee”,

B.emp\_id AS “Sup\_ID”, B.emp\_name AS “Supervisor”

FROM employee A, employee B

WHERE A.emp\_id = B.emp\_id

**Cross-Join**:

-Cartesian product of the two tables included in the join.

-The table after join contains the same number of rows as in the cross product of number of rows in the two tables.

-If a WHERE clause is used in cross join, it will work like an INNER JOIN

SELECT stu.name, sub.subject

FROM students AS stu

CROSS JOIN subjects AS sub;

**Index:**

-Index is a data structure that provides quick lookup of data in a column or columns of a table

-It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

\*\*Create Index\*\*

CREATE INDEX index\_name

ON table\_name (column\_1, column\_2)

\*\*Drop Index\*\*

DROP INDEX index\_name

**Unique Index**:

-they maintain data integrity by ensuring that no two rows of data in a table have identical key

-once it’s defined, uniqueness is enforced whenever keys are add/changed within the index

CREATE UNIQUE INDEX myIndex

ON students (eroll\_no);

**Clustered vs Non-clustered Index:**

-With the clustered index the rows are stored physically on the disk in the same order as the index. Therefore, there can only be one clustered index.

-With a non-clustered index there is a second list that has pointers to the physical rows. We can have many non-clustered indices, although each new index will increase time it takes to write new records

-Clustered index is generally faster to read. But writing to a table with a clustered index is slower, if there is a need to rearrange the data.

**Data Integrity**:

-It is the assurance of accuracy and consistency of data over its entire life-cycle.

-It is a critical aspect to the design, implementation and usage of any system which stores, processes and retrieves data.

-It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database

**Query**:

-Query is request for data or information from a database table/s.

-It can either be *select* query of *action* query.

\*\*Select Query\*\*

SELECT fname, lname

FROM myDB.students

WHERE student\_id = 1

\*\*Action Query\*\*

UPDATE myDB.students

SET fname = ‘Captain’, lname= ‘America’

WHERE student\_id = 1;

**Sub-Query**:

-Query within a query, aka nested query or inner query

-is used to restrict or enhance the data to be queried by the main query

SELECT name, email, mob, address

FROM myDB.contacts

WHERE roll\_no IN (

SELECT roll\_no

FROM myDB.students

WHERE subject = ’Math’

)

**Correlated and Non-Correlated Sub-Queries**:

-Correlated sub-queries cannot be considered as an independent query but it can refer the column in a table listed in the FROM of the main query.

**SELECT clause**:

-SELECT operator is used to select data from database.

-The data returned is stored in a result table called result-set

**Common clauses used with SELECT query**:

***WHERE***: It is used to filter records that are necessary, based on specific conditions.

***ORDERBY***: It is used to sort the records based on some fields in ASC or DESC

SELECT \*

FROM myDB.students

WHERE graduation\_year = 2020

ORDER BY studentID DESC

***HAVING***: It is used to filter records in combination with GROUP BY clause

***GROUP*** ***BY***: It is used to group records with identical data and can be used in conjunction with some aggregated functions to produce summarized results from the database.

-It is different from WHERE, since WHERE clause cannot filter aggregated records.

SELECT COUNT(studentID), country

FROM myDB.students

WHERE country != “USA”

GROUP BY country

HAVING COUNT(studentID) > 5;

**UNION, MINUS and INTERSECT commands:**

**-***UNION* operator combines and returns the result-set retrieved by two or more SELECT queries

-*MINUS* operator is used to remove duplicates from the result set obtained by the second SELECT query from that of the first SELECT query and return the filtered results from the first.

-*INTERSECT* combines the result from the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

\*\*Fetch the union of queries\*\*

SELECT name FROM Students

UNION

SELECT name FROM Contacts

\*\*Fetch the union of queries with duplicates\*\*

SELECT name FROM Students

UNION ALL

SELECT name FROM Contacts

\*\*Fetch names from students that are not in contacts\*\*

SELECT name FROM Students

MINUS

SELECT name FROM Contacts

\*\*Fetch names from students that are present in contacts as well\*\*

SELECT name FROM Students

INTERSECT

SELECT name FROM Contacts

**Cursor:**

-A database cursor is a control structure that allows for traversal of records in a database

-Cursors facilitate processing after traversal, such as retrieval, addition, deletion

-They can be viewed as a pointer to row in a set of rows

**Working with SQL Cursor**

1. DECLARE a cursor after any variable declaration. Cursor declaration must always be associated with a SELECT statement
2. Open cursor to initialize the result set. The OPEN statement must be called before fetching rows from the result set
3. FETCH statement to retrieve and move to the next row in the result set
4. Call the CLOSE statement to deactivate the cursor
5. Finally, use the DEALLOCATE statement to delete the cursor definition and release the associated resources

\*\*Declare all Required Variables\*\*

DECLARE @name VARCHAR (50)

\*\*Declare Cursor Name\*\*

DECLARE db\_cursor CURSOR FOR

SELECT name

FROM myDB.students

WHERE parent\_name IN (‘Dad’, ‘Mom’)

\*\*Open cursor and fetch data into @name\*\*

OPEN db\_cursor

FETCH next

FROM db\_cursor

INTO @name

\*\*Close the cursor and deallocate the resources\*\*

CLOSE db\_cursor

DEALLOCATE db\_cursor

**Entities and Relationships**:

-Entity can be a real world object, tangible or intangible, that can be easily identifiable. For example in a college database, students, professors, workers, departments are entities.

-Each entity has some associated properties that provide it an identity

-Relations or links between entities that have something to do with each other. For example the table in company’s database can be associated with the salary table in the same database.

**Types of Relationships in SQL:**

***One-to-one***: each record in one table is associated with max number of one record in the other table.

***One-to-many & Many-to-one:*** A record in a table is associated with multiple records in the other table.

***Many-to-many***: Multiple instances in both sides are needed for defining a relationship

***Self-Referencing relationships***: when a table needs to define a relationship with itself.

**ALIAS**:

-It is a temporary name assigned to the table or table column for the purpose of a particular SQL query.

-Obfuscation technique to secure real names of database fields.

-A table alias is also called **correlation name.**

**-**represented explicitly by AS keyword, sometimes performed without keyword

SELECT A.emp\_name AS “Employee” \*\* ALIAS using AS keyword \*\*

B.emp\_name AS “Supervisor”

FROM employee A, employee B \*\* ALIAS without AS keyword \*\*

WHERE A.emp\_sup = B.emp\_id

**Views**:

-A View is a virtual table based on the result-set of an SQL statement.

-It contains rows and columns, just like a real table.

-The fields in a view are fields from one or more real tables in the database

**Normalization**:

-Normalization represent the way of organizing structured data in the database efficiently.

-It includes creation of tables, establishing relationships between them, and defining rules for those relationships.

-Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

**Denormalization**:

-It is the inverse process of normalization, where the normalized schema is converted into schema which has a redundant information.

-The performance is improved by using redundancy and keeping the redundant data consistent.

-The reason for performing Denormalization is the overheads produced ion query processor by an over-normal structure.

**Forms of Normalization:**

1. ***First Normal Form***

Every attribute in the relation is a single-valued attribute.

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Address | Books | Salutation |
| Sara | Hooks Drive | Until the day I die  Inception | Mrs. |
| Jane | North Oaks | The Alchemist  Inferno | Ms. |

**First normal Form**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Address | Books | Salutation |
| Sara | Hooks Drive | Until the day I die | Mrs.. |
| Sara | Hooks Drive | Inception | Mrs.. |
| Jane | North Oaks | Beautiful Bad | Ms. |
| Jane | North Oaks | Woman 99 | Ms. |

1. ***Second Normal Form***

-A relation is in second normal form if it satisfies the conditions for the first normal form and does not contain partial dependency.

-A relation in 2NF has no partial dependency, i.e., it has non-prime attribute that depends on any proper subset of any candidate key of the table.

2NF Student Table

|  |  |  |  |
| --- | --- | --- | --- |
| Student ID | Student | Address | Salutation |
| 1 | Sara | Hooks Drive | Mrs.. |
| 2 | Jane | North Oaks | Ms. |

2NF Books Table

|  |  |
| --- | --- |
| StudentId | Books Issued |
| 1 | Until the day I die |
| 1 | Inception |
| 2 | Beautiful Bad |
| 2 | Woman 99 |

1. ***Third Normal Form***

-A relation is said to be in third normal form if it satisfies the conditions for the second normal form and there is no transitive dependency between the non-prime attributes, i.e., all non-prime attributes are determined only by the candidate keys of the relation and not by any other non-prime attribute.

3NF Student Table

|  |  |  |  |
| --- | --- | --- | --- |
| StudentID | Student | Address | Salutation\_ID |
| 1 | Sara | Hooks Drive | 1 |
| 2 | Jane | North Oaks | 2 |

3NF Books Table

|  |  |
| --- | --- |
| StudentID | Books Issued |
| 1 | Until the day I die |
| 1 | Inception |
| 2 | Beautiful Bad |
| 2 | Woman 99 |

3NF Salutation Table

|  |  |
| --- | --- |
| Salutation\_ID | Salutation |
| 1 | Mrs. |
| 2 | Ms. |

1. ***Boyce-Codd Normal Form***

-A relation is in Boyce-Codd Normal Form if it satisfies the conditions for third normal form and for every functional dependency, LHS is super key.

-In other words, a relation in BCNF has non-trivial functional dependencies in the form X->Y, such that x is always the super key.

-In the above example, Student table and Salutation table exist in BCNF because they have unique key Student Id and salutation Id which identifies a unique student and salutation respectively. But it cannot be said so for the Books table, because it does not have a super key.

**TRUNCATE, DELETE and DROP statements:**

-DELETE statement is used to delete rows from a table

DELETE FROM Candidates

WHERE CandidateId > 1000;

-TRUNCATE command is used to delete all the rows from the table and free the space containing the table

TRUNCATE TABLE Candidates;

-DROP command is used to remove object from database. If we drop a table, all the rows in the table is deleted and the table structure is removed from the database

DROP TABLE Candidates;

**DROP vs. TRUNCATE:**

-If a table is dropped, everything associated with the tables are dropped as well. This includes-

-> Relationships defined with other tables

-> Integrity checks / constraints

-> Access privileges

-> Other grants

-If a table is truncated, none of the above problems exist and the table retains its original structure

**DELETE vs. TRUNCATE:**

-Truncating deletes all the rows and frees the space containing the table

-Deleting only deletes the rows from the table based on the condition given in the ‘where’ clause or deletes all the rows from the table if no conditions is specified but it does not free the space containing the table.

**Aggregate vs Scalar functions:**

**-**Aggregate performs operations on collection of values to return a single scalar values.

-Often used with GROUP BY and HAVING

-Examples:

**AVG**(): calculates mean/average of the collection of values

**COUNT**(): counts the total number of records in a specific table or view

**MIN**() : calculates the minimum of the collection of values

**MAX**(): calculates the maximum of the collection of values

**SUM**(): calculates the sum of the collection of values

**FIRST**(): fetches the first element in the collection

**LAST**(): fetches the first element in the collection

-A Scalar function returns a single value based on the input value.

-Examples:

**LEN**(): total length of the field

**UCASE**(): converts string to uppercase

**LCASE**(): converts string to uppercase

**MID**(): extracts substrings from a collection of string values in a table

**CONCAT**(): Concatenates two or more strings

**RAND**(): Generates random collection numbers of given length

**ROUND**(): Calculates round off integer value for decimal point values

**NOW**(): returns current date and time

**FORMAT**(): sets the format to display

**User-Defined Functions**

**-**They are functions that accepts parameters, perform complex calculations, and return a value.

-They are written to use the logic repetitively whenever required.

**Types of User-Defined Functions:**

-***Scalar Function:*** returns a single scalar value

-***Table Value Function:*** return a table as output

-***Inline:*** returns a table data type based on a single SELECT statement

-***Multiline-statement:*** multiple SELECT statement can be used inside the function body