**DBMS**

Database

* A database is a collection of data, typically describing the activities of one or more related organizations.
* For example, a university database might contain information about the following:
* Entities such as students, faculty, courses, and classrooms.
* Relationships between entities, such as students’ enrollment in courses, faculty teaching courses, and the use of rooms for courses.

DBMS

* Database Management System (DBMS) is a software that is designed to assist in maintaining and utilizing large collections of data efficiently.
* The first general purpose DBMS was designed by Charles Bachman in 1960s which was also called as Integral data model.
* In 1960s, IBM developed the Information Management System (IMS) DBMS which was also called as hierarchical data model.
* In 1970, Edgar Codd, at IBM proposed a new model called as relational data model.
* The SQL query language for relational databases, developed as part of IBM’s System R project, is now the standard query language. SQL was standardized in the late 1980s, and the current standard, SQL-92 was adopted by the American National Standards Institute (ANSI) and International Standards Organization (ISO).

Advantages of DBMS

* Data Independence
* Efficient data access
* Data integrity and security
* Data administration
* Concurrent access and data recovery

Data Model

* A data model is a collection of high-level data description constructs that hide many low-level storage details.
* A DBMS allows a user to define the data to be stored in terms of a data model.
* Most database management systems today are based on the relational data model.

The Relational Data Model

* A description of data in terms of a data model is called a schema. In the relational model, the schema
* for a relation specifies its name, the name of each field (or attribute or column), and the type of each field.
* Ex : Students(sid: string, name: string, login: string, age: integer, cgpa: real)

Entity

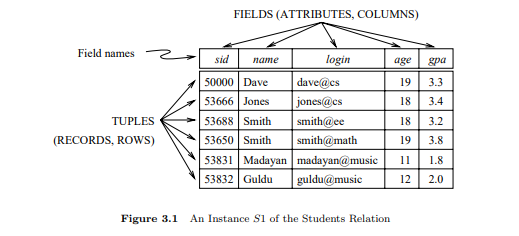
* An entity is an object in the real world that is distinguishable from other objects.
* A collection of similar entities is called as entity set.
* A relationship is an association among two or more entities.

Key Points

* Database design has six steps: requirements analysis, conceptual database design, logical database design, schema refinement, physical database design, and security design.
* In the ER model, a real-world object is represented as an entity.
* An entity set is a collection of structurally identical entities.
* Entities are described using attributes.
* Each entity set has a distinguished set of attributes called a key that can be used to uniquely identify each entity.
* A relationship is an association between two or more entities.
* A relationship set is a collection of relationships that relate entities from the same entity sets.
* A relationship can also have descriptive attributes.

Relational Models

* An instance of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema.
* A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields.
* An instance of the students relation appears in Figure 3.1:



* Note that no two rows are identical. This is a requirement of the relational model—each relation is defined to be a set of unique tuples or rows.
* The degree, also called arity, of a relation is the number of fields.
* The cardinality of a relation instance is the number of tuples in it.
* A relational database is a collection of relations with distinct relation names.
* The relational database schema is the collection of schemas for the relations in the database.

SQL Installation guide

**https://www.youtube.com/watch?v=YSOY\_NyOg40**

SQL Server connection using terminal (command prompt)

* open bin path of MySQL server folder -> copy the path.
* open terminal -> cd C:\Program Files\MySQL\MySQL Server 8.0\bin -> press enter.
* use command MySQL -u root -p ->C:\Program Files\MySQL\MySQL Server 8.0\bin>MySQL -u root -p -> press enter.
* now enter your root password: Enter password: \*\*\*\*\*\*\*\*\*\* -> press enter.
* you are connected to MySQL: start writing queries -> MySQL>

SQL

* It was the query language of the pioneering System-R relational DBMS developed at IBM.
* The first SQL standard was developed in 1986 by the American National Standards Institute (ANSI) and was called SQL-86.
* The SQL-92 language standard uses the word table to denote relation, and we will often follow this convention when discussing SQL.
* The subset of SQL that supports the creation, deletion, and modification of tables is called the Data Definition Language (DDL).
* SQL lets you access and manipulate databases.

Data Types in SQL

**1. string**

**2. number**

**3. date**

**String**

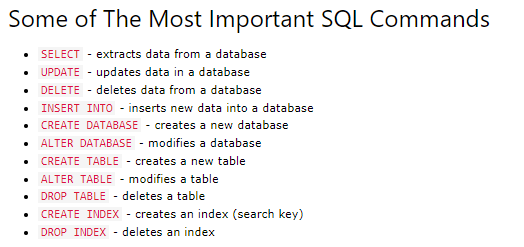
* char (1 - 255)
* varchar (1 - 65535)
* text (1 - 255)
* medium text (1 - 16777215)

**Number**

* bit (1 - 64)
* int / integer (-2147483648 to 2147483647)
* float (size, p) -> p (0 to 24)
* double(size,p) -> p(25 to 35)
* bool -> 0 / 1 => true / false

**Date**

* yyyy-mm-dd format ('1000-01-01' to '9999-12-31')



**CREATE SYNTAX**

CREATE TABLE TABLE\_NAME (ATTRIBUTE\_NAME1 Datatype,

ATTRIBUTE\_NAME2 Datatype…)

**Example**

CREATE TABLE student (name char(25), id int, cgpa float)

**INSERT SYNTAX**

INSERT INTO TABLE\_NAME (ATTRIBUTE\_NAME1, ATTRIBUTE\_NAME2, ATTRIBUTE\_NAME3) VALUES (value1, value2, value3)

**EXAMPLE**

INSERT INTO student (name, id, cgpa) VALUES (‘kalyan’, 30959, 9.46);

**DELETE SYNTAX**

DELETE FROM TABLE\_NAME WHERE attribute\_name = value

**EXAMPLE**

DELETE FROM student WHERE id = 30959;

**UPDATE SYNTAX**

UPDATE TABLE\_NAME SET attribute\_name = value WHERE condition

**EXAMPLE**

UPDATE student SET cgpa = 9.5 WHERE id = 30959;

**SQL SELECT Statement**

It is used to select data from a database and the data returned is stored in a table which is called result-set.

**SELECT SYNTAX TO RETRIVE ALL RECORDS**

Select \* from Table \_Name.

**EXAMPLE**

Select \* from student;

**SELECT SYNTAX TO GET SPECIFIC COLUMNS**

Select column1, column2 from Table\_Name;

**EXAMPLE**

Select name, id from student;

**SELECT SYNTAX TO GET DISTINCT VALUES**

Select distinct column\_name from table\_name

**EXAMPLE**

Select distinct name from student;

**SELECT DISINCT TO GET NAMES WITH EVEN ID NUMBERS**

Select distinct s.name from student s where MOD(s.id,2) =0;

**SQL WHERE Clause**

It is used to filter the records based on a condition.

**SQL WHERE SYNTAX**

Select column\_names from table\_name WHERE condition

**Example**

Select id, name from student where cgpa=9.46;

A screenshot of a computer

Description automatically generated

**SQL BETWEEN EXAMPLE**

Select \* from student where id BETWEEN 30900 and 31000;

**SQL LIKE EXAMPLE**

*It is used in a WHERE clause to search for a specified pattern in a column.*

***Examples***

* Select \* from student where name like ‘a%’;
* This will retrieve all records whose name starts with a.
* Select \* from student where name like ‘%a’;
* This will retrieve all records whose name ends with a.
* Select \* from student where name like ‘%ab%’;
* This will retrieve all records whose name contains ab.

**SQL NOT**

* Select \* from city where not name = ‘kalyan’;
* This will retrieve all the records where the name is not Kalyan.
* Select \* FROM student WHERE id <> 18;
* This will retrieve all the records where the id is not 18.
* The NOT can be used with NOT LIKE, NOT BETWEEN, etc.

**SQL ORDER BY**

* It is used to order the records.
* Its default order is ascending.
* The values can be ordered in ascending order using ASC.
* The values can be ordered in descending order using DESC.

**Examples**

* Select \* from student ORDER BY id, name;
* This will retrieve all the records by sorting the id and name column values in ascending order.
* Select \* from student ORDER BY id ACS, name DESC;
* This will retrieve all the records by sorting id column values in ascending order and name column values in descending order.

**SQL IS NULL**

* It is used to check if a value is null or not.
* It returns the rows which are null based on condition.

**Example**

* Select \* from student id **is null**;
* This will retrieve all the records from student table where id is null.

**SQL IS NOT NULL**

* It is used to check if a value is not null or not.
* It returns the rows which are not null based on condition.

**Example**

* Select \* from student id **is not null**;
* This will retrieve all the records from student table where id is not null.

**SQL SELECT TOP**

* It is like MySQL limit.
* It retrieves records up to a provided range.

**Example**

* Select Top 3 \* from student; (SQL)
* This will retrieve the first 3 records from the student table.
* Select \* from student **LIMIT** 3; (MySQL)
* This will retrieve the first 3 records from the student table.
* Select \* from student **LIMIT** (select **count**(\*) \* 0.5 from student);
* This will retrieve the first 50 percent of rows data from the table.
* Select \* from student where id > 30959 **LIMIT** 3;
* This will retrieve the top 3 rows from student table where id > 30959.

**SQL MIN & Max**

* The min and max functions in sql are used to get the min and max values in the specified column.
* It works with numeric data types and date time values.

**Examples**

* Select min(id) from student;
* This will return the minimum id number from student table.
* Select max(id) from student;
* This will return the maximum id number from student table.
* Select min(id) as minimum\_id from student table;
* This will return the minimum id along with as alias for minimum\_id.

**SQL COUNT**

* It returns the number of records based on column.

**EXAMPLE**

* Select **count**(id) from student;
* It returns the number of id’s.
* Select count(distinct id) from student;
* It returns the number of unique id’s.
* Select count(\*) as [number of records] from student;
* It returns the number of records with an alias number of records.

**SQL SUM**

* It returns the sum of all records of a specific column.

**Example**

* Select sum(id) from student;
* It will return the sum of all id’s.
* Select sum(id) from student where name = ‘kalyan’;
* It will return the sum of id’s where name is kalyan.

**SQL AVG**

* It returns the average of a column.

**Example**

* Select avg(cgpa) as [Average CGPA] from student;
* It will return the average cgpa from student table;

**SQL LIKE**

* It is used with WHERE clause to search for a specified pattern in a column.
* It uses two wildcard characters to search for patterns.
* The first wildcard character is ‘%’ which represents 0, 1 or multiple characters.
* The second wildcard character is ‘\_’ which represents single character.

**Example**

* Select \* from student where name like ‘%a’;
* Select \* from student were name REGEXP ‘[a]$’;
* This will retrieve all the records where student name ends with ‘a’.
* Select \* from student where name like ‘a%’; (SQL)
* Select \* from student where name REGEXP ‘^[a]’; (MySQL)
* It will retrieve all the records where student name starts with ‘a’.
* Select \* from student where name like ‘%an%’;
* It will retrieve all the records where name contains ‘an’.
* Select \* from student name like ‘k\_l%’;
* It will retrieve all the records where name is kalyan, kali, etc.
* Here the ‘\_’ represents any single character.
* Here the ‘%’ represents all characters after the pattern.

**SQL WildCard Characters**

* A wildcard character is used to substitute one or more characters in a stirng.
* They are used with LIKE operator.

**Examples**

* Select \* from student where name like ‘[rvk]%’; (SQL)
* Select \* from student where name REGEXP ‘[rvk]’; (MySQL)
* This will retrieve all the records where name starts with ‘r’ or ‘v’ or ‘k’.
* Select \* from student where name like ‘[a-c]%’;
* This will retrieve all the records where name starts with ‘a’ or ‘b’ or ‘c’.

**SQL IN**

* It allows us to specify multiple values in a where clause.
* It is a shortcut for or operations.

**Example**

* Select \* from student where cgpa in 9.46;
* This will retrieve all the students having cgpa as 9.46.
* Select \* from student where id in (30959,30976);
* This will return all the student details with id as 30959 and 30976.

**SQL NOT IN**

* It is opposite of IN.

**Example**

* Select \* from student where id NOT IN (30959, 30976);
* This will return all the student details with id except 30959 and 30976.
* Select \* from student where id NOT IN(select sid from certified\_students);
* This will return all the records where student ids are not in certified\_students.

**SQL BETWEEN**

* It is used to retrieve a range of records based on where condition.

**Example**

* Select \* from student where id BETWEEN 30950 and 30960;
* This will retrieve those records where id is between 30950 and 30960.
* Select \* from student where DOB BETWEEN ‘2003-01-01’ and ‘2003-12-31’;
* This will retrieve the records where DOB is between January 2003 and December 2003.

**SQL ALIASES**

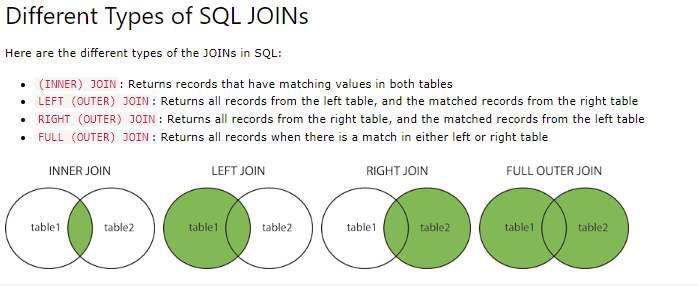
* Aliases are used to give temporary names to a table, column in a table.
* An alias only exists for the duration of that query only.
* An alias is created with **AS** keyword.

**Example**

* Select id as Student\_ID, name as STUDENT\_NAME from student;
* This will retrieve id values and name values from student table by adding column names as STUDENT\_ID and STUDENT\_NAME.

***SQL JOINS***

* *A Join clause is used to combine rows from two or more tables based on a related common column between them.*



**Examples**

* Select student.name, certified\_students.sid from student **INNER JOIN** certified\_students **on** student.id = certified\_students.sid;
* This will return all the records with the ids that are available in both the tables.
* The **INNER JOIN** returns the records that match in both the tables.
* Select student.name, certified\_students.cid from student **LEFT JOIN** certified\_students on student.id = certified\_student.id.
* This will return all the records of student table even if they don’t have cid.
* If they don’t have cid it will return null.
* The **LEFT JOIN** returns the all the records from left table(student), even if there are no matches in the right table(certified\_students).
* Select student.name,certified\_students.cid from certified\_students **RIGHT JOIN** student on student.id = certified\_students.sid;
* This will return all the records from certified\_students even if they don’t have name.
* The **RIGHT JOIN** returns the all the records from rightt table(certified\_students), even if there are no matches in the left table(student).
* Select student.name, certified\_students.sid from student **FULL OUTER JOIN** certified\_students **on** student.id = certified\_students.sid;
* The **FULL OUTER JOIN** will return all records when there is a match in left table or right table records.

**SQL UNION**

* It is used to combine result set of two more.
* Every select statement within **UNION** must have the same number of columns.
* The column must also have similar data types.
* The **UNION** operator returns only distinct values by default.
* If we want to allow duplicate values, then **UNION ALL** can be used.

**Examples**

* Select id from student **UNION** select sid from certified\_students;
* This will return all the distinct IDs from both the tables.
* Select id from student **UNION ALL** select sid from certified\_students;
* This will return all the ids from both the tables including duplicates.

**SQL GROUP BY**

* The GROUP BY statement groups rows that have the same values into summary rows like “find the number of customers in each country”.
* The GROUP BY statement is often used with aggregate functions (COUNT( ), MAX( ), MIN(), SUM( ), AVG( ).

**EXAMPLE**

* Select count(customerId), country from customers Group By country;
* This will return all the records with count of each customerId in each country.

**SQL HAVING Clause**

* The SQL HAVING Clause is used because we cannot use where clause with aggregate functions like MAX( ), MIN( ), COUNT( ), etc.

**Example**

* Select count(customerId), country from customers group by country **having** count(customerId) > 5;
* This will return all the countries where count of customerId > 5.

**SQL EXISTS**

* The EXISTS operator is used to check the existence of any record in a subquery.
* The EXISTS operator returns true if the subquery returns one or more records.

**Example**

* Select name from student where **EXISTS** (select cid from certified\_students where certified\_students.id = students.id);
* This will return all the names of certified students.

**SQL ANY**

* The **any** operator allows us to perform a comparison between a single column value and a range of other values.
* It returns a Boolean value as a result.
* It returns true if any of the subquery values meet the condition.

**Example**

* Select name from student where id = **ANY** (select cid from certified\_students where cid = id);
* This will retrieve all the names of students who are certified.

**SQL ALL**

* The **all** operator allows us to perform a comparison between a single column value and a range of other values.
* It returns a Boolean value as a result.
* It returns true if any of the subquery values meet the condition.

**Example**

* Select name from student where id = **ALL**(select cid from certified\_students where cid = id);
* This will retrieve all the names of students who are certified.

**SQL SELECT INTO**

* The SELECT INTO statement copies values from one existing table into another new table.

**Examples**

* **Select** \* **into** certifications from certified\_students;
* This will copy all the details of certified\_students table into certifications table.
* Select name **into** certified\_students from students;
* This will copy the name column values form students table to certified\_students.
* Select \* **into** student\_backup **IN** ‘klu’ from students;
* Here we are using IN clause to point out another database called as ‘klu’ where our student\_backup table is going to be created.
* This will copy all the data from students table into student\_backup table.

**SQL INSERT INTO SELECT**

* The **INSERT INTO SELECT** is used to insert values of one existing table into another existing table;
* The **INSERT INTO SELECT** is only possible if the data types of both source and destination tables matches.

**Example**

* **INSERT INTO** students(name,id,cgpa) **select** cname,cid,score from certified\_students;
* This will copy all the values from certified\_students table to students table because the data types and order of columns in both the tables matches.
* **INSERT INTO** students(name,id,cgpa) select cname,cid,score from certified\_students WHERE cname = ‘AWS-CP’;
* This will copy all the records where cname = ‘AWS-CP’.

**SQL CASE**

* It is like switch case in c or java where it returns a value when a specified first condition is met.

**Examples**

* Select id, name,

**CASE**

**WHEN** cgpa > 9 **THEN ‘**cgpa is above 9’

**WHEN** cgpa > 8.5 **THEN ‘**cgpa is above 8.5’

**ELSE ‘**cgpa is less than 8.5’

**END** AS CGPACriteria FROM student;

* This will retrieve id, name column values from cgpa with an additional alias column as ‘CGPACriteria’ with the message as ‘cgpa is’ based on the student cgpa.
* Select id, name from student ORDER BY (CASE WHEN ID IS NULL THEN named ELSE id);
* This will retrieve id,name column values based on ascending order of id, if id is null then id field will be replaced by name;

**STORED PROCEDURE**

* A stored procedure is a prepared SQL code that we can save and reuse it.

**Examples**

* **CREATE PROCEDURE** selectAll @ID INT **AS** Select \* from students where id = @ID Go;
* **EXEC** selectAll @ID = 30959;
* This will return all the records where id is 30959.
* **CREATE PROCEDURE** selectAll @ID INT, @NAME nvarchar(15) **AS** Select \* from students where id = @ID and name = @NAME;
* **EXEC** selectAll @ID = 30959, @NAME = ‘kalyan’;
* This will return all the records where id is 30959 and name is kalyan.

**SQL COMMENTS**

* Single line comments in sql are represented with ‘- - ‘.
* Multi line comments in sql are represented with ‘/\* \*/’.

**SQL CREATE Database**

* It is used to create a database.

**Example**

* CREATE DATABASE klu;

**SQL DROP Database**

* It is used to drop or delete an existing database.

**Example**

* DROP Database klu;

**SQL BACKUP Database**

* It is used in SQL server to create a full backup of an existing database.

**Example**

* **BACKUP Database** klu **TO DISK = “**Filepath **“;**
* This willcreate a full **BACKUP** klu database into the provide file path.
* **BACKUP Database** klu **TO DISK =** “FILE PATH” **WITH DIFFERENTIAL**;
* It will backup only the parts of database that have changed.

**SQL CREATE TABLE**

* It is used to create a table.

**Example**

* **CREATE TABLE** students(id int, name varchar(20), cgpa float);
* This will create a table named as student with id, name and cgpa attributes.

**SQL DROP TABLE**

* It is used to drop or delete an existing table.

**Example**

* **DROP TABLE** students;
* This will delete students table.

**SQL TRUNCATE TABLE**

* It is used to delete data inside a table but it won’t delete the table.

**Example**

* **TRUNCATE TABLE** students;
* This will delete the data inside the students table.

**SQL ALTER TABLE**

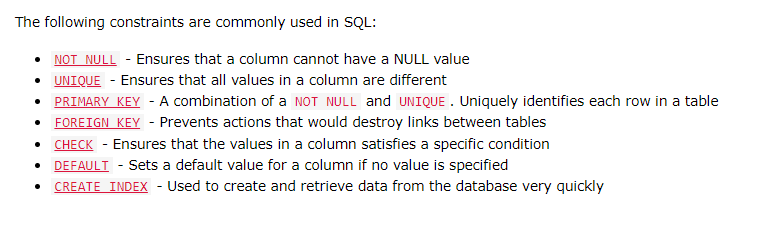
* It is used to ADD, DELETE or MODIFY columns in an existing table.
* It is also used to ADD or DROP various constraints in an existing table.

**Examples**

* **ALTER TABLE** students **ADD** year int;
* This will add an extra column named as year to the existing student table**.**
* **ALTER TABLE** students **DROP COLUMN** year;
* This will drop the year column from the students table.
* **ALTER TABLE** students **RENAME COLUMN** year **TO** study\_year;
* This will change the column name year to study\_year in students table.
* **ALTER TABLE** students **MODIFY COLUMN** id bigint;
* This will change the column datatype of id from int to bigint.

**SQL CONSTRAINTS**

* SQL constraints are used to specify rules for the data in a table.
* They can be used to limit the type of data that go into the table.
* CONSTRAINTS can be table level or column level.
* The CONSTRAINTS are to be defined while creating or while altering a table.



**EXAMPLES**

**NOT NULL**

* CREATE TABLE stu\_details(id int **NOT NULL**, name varchar(50) **NOT NULL**) ;
* This will define a **NOT NULL CONSTRAINT** on id and name attributes i.e. the id and name values must not be **NULL.**
* **ALTER TABLE** stu\_details **ALTER COLUMN** age int **NOT NULL**;
* This will define a **NOT NULL CONSTRAINT** for age attribute in stu\_details table.

**UNIQUE**

* CREATE TABLE employee (eid int, ename varchar(30), esal float, **UNIQUE** (eid));
* This will define a **UNIQUE** constraint on eid i.e. it will not allow duplicate values for eid column.
* ALTER TABLE employee **ADD UNIQUE** (EID);
* This will add an UNIQUE constraint on EID attribute in employee table.
* CREATE TABLE employee (eid int, ename varchar(30), esal float, **CONSTRAINT** empdetails **UNIQUE** (eid,ename));
* This will define a **UNIQUE** constraint on eid and ename i.e. it will not allow duplicate values for eid and ename columns.

**PRIMARY KEY**

* A primary key constraint is used to uniquely identify each record in a table.
* Primar key values must contain **unique** values and cannot contain **null** values.
* A table can have only one primary key.
* CREATE TABLE employee (eid int, ename varchar(30), esal float, **PRIMARY KEY** (eid));
* This will define a **PRIMARY KEY** constraint on eid i.e. it will not allow duplicate values for eid column and null values.
* ALTER TABLE employee **ADD PRIMARY KEY** (EID);
* This will add an **PRIMARY KEY** constraint on EID attribute in employee table.
* CREATE TABLE employee (eid int, ename varchar(30), esal float, **CONSTRAINT** empdetails **PRIMARY KEY**(eid,ename));
* This will define a **PRIMARY KEY** constraint on eid and ename i.e. it will not allow duplicate values as well as null values for eid and ename columns.

**FOREIGN KEY**

* It is used to prevent actions that would destroy links between tables.
* It is a field or collection of fields in one table that refers to **primary key** in another table.
* The table with the **foreign key** is called **child table** and the table with **primary key** is called **parent table.**

**Examples**

* CREATE TABLE certified\_studnets(cid int, cname varchar(30), **PRIMARY KEY** (cid),

**FOREIGN KEY** (id) **REFERENCES** students(id);

* This will create a table named as certified\_students where cid is PRIMARY KEY and id as FOREIGN KEY which is a PRIMARY KEY in students table.
* ALTER TABLE certified\_students **ADD FOREIGN KEY** (id) **REFERENCES** students(id);
* This will alter a table named as certified\_students where cid is PRIMARY KEY and id as FOREIGN KEY which is a PRIMARY KEY in students table.

**CHECK**

* It is used to **LIMIT** the value range that can be placed in a column.
* If we define a **CHECK** constraint on a column, it will allow only certain values for this column.

**Example**

* **Create table** voters (vid int, name varchar(20), age int **CHECK (AGE >= 18)**);
* This will create a **CHECK** for age stating that the age should be >= 18.
* During insertion if we provide age below 18, it will throw an error and the record will not get inserted.

**DEFAULT**

* It is used to set a default value for a column.
* The specified default value will be added to all new records if no value is specified.

**Examples**

* Create table stu\_det(id int, name varchar(30) **DEFAULT** **‘student’,** cgpa float);
* This will create a table named as stu\_det with id, name attributes where name having DEFAULT value as ‘student’.
* ALTER TABLE stu\_det ALTER id **SET DEFAULT** 21000000;
* This will alter a table named as stu\_det where id is altered with DEFAULT value as 21000000.

**AUTO INCREMENT**

* It generates a unique number automatically when a new record is inserted.
* Often this is the primary key field that we would like to be created automatically whenever a new record is inserted.
* By default the value of AUTO INCREMENT starts from 1.

**Examples**

* Create table stu (id int **AUTO\_INCREMENT**, name varchar (30));
* This will create a table stu where we need not to insert id values.
* Insert Into stu (‘kalyan’);
* This will inserd a records with id as 1 and name as kalyan.
* ALTER TABLE stu **AUTO\_INCREMENT** = 100;
* This will start the **AUTO\_INCREMENT** from 100.

**SQL VIEWS**

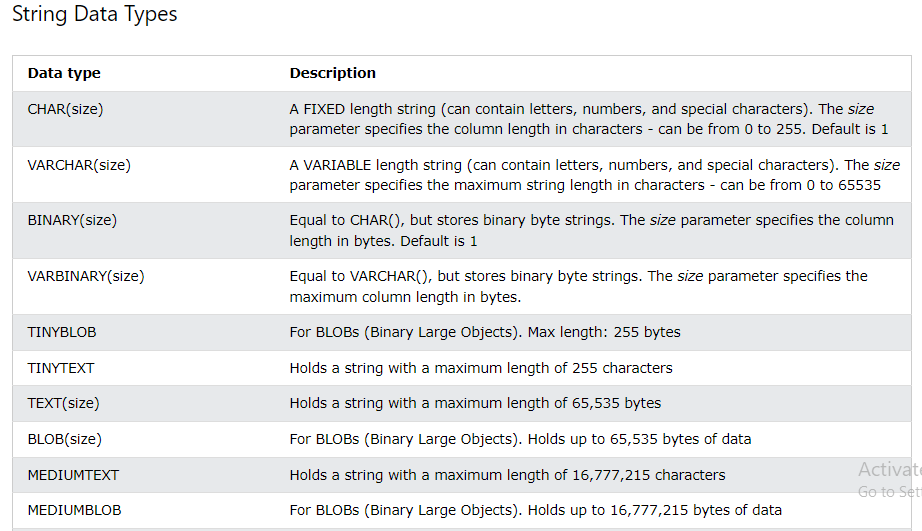
* A view is a virtual table based on the result-set of an SQL statement.
* A view contains columns and rows just like a real table.

**Example**

* **CREATE VIEW [TOP STUDENTS DETAILS] AS** SELECT name, id from students where cgpa > 9;
* This will create a virtual table or **view** AS TOP STUDENTS DETAILS with name, id as attributes consisting records of students where cgpa > 9.
* Select \* from [TOP STUDENTS DETAILS];
* This will retrieve all the records from TOP STUDENTS DETAILS **view**.

**SQL DATA TYPES**

* The data type of column defines what type of value it should hold.
* Each column should have a name and a data type.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**STANDARD EXAMPLES**

**INNER JOIN**

* Given the **CITY** and **COUNTRY** tables, query the sum of the populations of all cities where the CONTINENT is 'Asia'.
* CITY. CountryCode and COUNTRY. Code are matching key columns

.

* **select sum (CITY. Population) from COUNTRY inner join CITY on country. Code = city. CountryCode where country. Continent = 'Asia';**

**CONCAT**

Query the following two values from the **STATION** table:

1. The sum of all values in *LAT\_N* rounded to a scale of 2 decimal places.
2. The sum of all values in *LONG\_W* rounded to a scale of 2 decimal places.

* **SELECT CONCAT(ROUND(SUM(LAT\_n), 2), ' ', ROUND(SUM(LONG\_W), 2)) AS ConcatenatedResult FROM Station;**

**STRING LENGTH**

* Query the two cities in **STATION** with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.
* **SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) ASC, CITY LIMIT 1;**
* **SELECT CITY, LENGTH(CITY) FROM STATION ORDER BY LENGTH(CITY) DESC, CITY LIMIT 1;**

**NESTED SELECTS WITH ROUND**

* Query the Western Longitude (LONG\_W) for the largest Northern Latitude (LAT\_N) in **STATION** that is less than 137.2345. Round your answer to  decimal places.
* **select round(LONG\_W,4) from STATION where LAT\_N = (select max(LAT\_N) from station where LAT\_N < 137.2345);**

**MULTIPLE ORDER BY**

* Query the Name of any student in **STUDENTS** who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.
* Select name from students where marks > 75 ORDER BY RIGHT(name,3), ID ASC;

**MAX WITH MULIPLICATION**

* We define an employee's total earnings to be their monthly salary \* months worked, and the maximum total earnings to be the maximum total earnings for any employee in the **Employee** table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.
* SELECT MAX(months \* salary), COUNT(employee\_id) FROM EMPLOYEE WHERE

(months \* salary) = (SELECT MAX(months \* salary) FROM EMPLOYEE);

**CEIL AND REPLCE FUNCTIONS**

* Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.:  average monthly salaries), and round it up to the next integer.

* SELECT CEIL(AVG(Salary) – AVG(REPLACE(Salary,0,””))) from EMPLOYEES;