CREATE DATABASE college;

USE college;

Creating our First Table:

CREATE TABLE table_name(

column_name1 datatype constraint;

column_name2 datatype constraint;

);

CREATE TABLE student(

Id INT PRIMARY KEY,

Name VARCHAR(50),

AGE INT NOT NULL

);

INSERTING THE VALUES:

INSERT INTO student VALUES(1,”ALMAS”,20);

SELECTION:

SELECT * FROM student

SQL DATATYPES:

CHAR : stores characters of fixed length (0,255)

VARCHAR: stores characters upto given length (0,255)

Difference is that

Col1 is CHAR(50)---and the name is PUNE then the memory takes 50 bits

Col2 is VARCHAR(50)---and the name is PUNE then the memory takes only 4 bits

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

BIT(1)---0,1

BIT(2)---00,01,10,11

DATE---YYYY-MM-DD

SIGNED AND UNSIGNED

Numeric that are positive like salary, age numbers---unsigned

TINYINT(-128 to 127)

TINYINT UNSIGNED(0 to 255) here the range is increased because of the unsigned

Types of SQL Commands:

DDL Data Definition Language: create, alter, rename, truncate and drop

DQL Data Query Language: select

DML Data Manipulation Language: insert update delete

DCL Data Control Language: grant and revoke permissions to users

TCL Transition Control Language: start transaction, commit, rollback

DATABASE QUERIES

CREATE DATABASE COLLEGE;

CREATE DATABASE **IF NOT EXISTS** COLLEGE;-→IF NEW DATABASE IS BEING CREATED AND THE DATABASE IS ALREADY EXISTS THEN THIS WAY ERROR DOESN'T EXISTS

DROP DATABASE COLLEGE;

DROP DATABASE IF EXISTS COLLEGE;

SHOW DATABASES;

USE DATABASE_NAME;

TABLE RELATED QUERIES:

CREATE TABLE TABLE_NAME(

COLUMN_NAME DATA-TYPE CONSTRAINT,

);

SELECT * FROM TABLE_NAME;

INSERT INTO TABLE_NAME(COL_NAME1,COL_NAME2)VALUES("SSSS","SSSS");

KEYS

PRIMARY KEY

- Uniquely identifies each row
- There is only one primary key and not null

FOREIGN KEY:

- This refers to the primary key of the other table
- Can be multiple Foreign Key
- Can have duplicate and null values

CONSTRAINTS

NOT NULL: cant have a null value

UNIQUE: all values are different –no duplicates

PRIMARY KEY: not null and unique

For example: PRIMARY KEY(id, name) is given then id, name alone can be duplicate but both id and name with same values cannot be used

FOREIGN KEY

CREATE stud(

Cust_id INT,

FOREINGN KEY (cust_id) references customer(id)

);

DEFAULT:

Salary INT DEFAULT 25000;

CHECK

It can limit the values allowed in a column

Eg:

CREATE TABLE CITY(

AGE INT;

CONSTRAINT AGE_CHECK CHECK(AGE>18 AND CITY==”DELHI”)

);

OR

CREATE TABLE CITY(

AGE INT CHECK(AGE>18)

);

SELECT COMMAND IN DETAIL:

SELECT

COL_1,COL_2

FROM TABLE_NAME

SELECT **DISTINT *COL_NAME*** FROM TABLE_NAME

SELECT COL_1,COL_2 FROM TABLE WHERE CONDITIONS;

EG: SELECT * FROM STUD WHERE MARKS>10;

WHERE CLAUSE:

OPERATORS

ARITHMETIC: +, -, *, /, %

EXAMPLE: SELECT * FROM STUD WHERE MARKS+10 >100

COMPARISION: = EQUAL TO,

!= NOT EQUAL

>=, <=,<,>, <>(NOT)

LOGICAL: AND , OR, NOT, IN , BETWEEN, ALL, LIKE, ANY

BITWISE: & BITWISE ANS

| BITWISE OR

AND: WHERE MARKS>90 AND CITY =”MUMBAI”----TWO CONDITIONS MUST SATISFY

OR: WHERE MARKS>90 OR CITY=”MUMBAI”-----ANY ONE CONITION CAN SATISY

BETWEEN: WHERE AGE BETWEEN 18 AND 30 ---INCLUSIVE ANYONE WITH 18 ,30 ALSO DISPLAYED

IN: WHERE CITY IN(“PARIS”,”US”)

NOT: WHERE CITY NOT IN (“PARIS”, “US”)

LIKE: WHERE CITY LIKE “%S”----STARTS WITH S

LIMIT CLAUSE:

SELECT COL1 FROM TABLE LIMIT NUMBER

--- SELECT * FROM STUDENT LIMIT 3

ORDER BY CLAUSE:

TO SORT IN ASC OR DESC

---SELECT * FROM STUDENT ORDER BY CITY ASC

----TO KNOW TOP 3 STUDENT SCORES : SELECT * FROM STUD ORDER BY MARKS DESC LIMIT 3

AGGREGATE FUNCTIONS

PERFORMS A CALCULATION ON A SET OF VALUES AND RETURNS A SINGLE VALUE

- **COUNT()**
- **MAX()**
- **MIN()**
- **SUM()**

- **AVG()**

----SELECT MAX(MARKS) FROM STUD;

GROUP BY CLAUSE:

GROUPS ROWS THAT HAVE SAME VALUE INTO SUMMARY ROWS

**GENERALLY WE USE GROUP BY WITH SOME AGGREGATION FUNCTIONS*

CONDITION: COUNT NO.OF STUDENTS IN EACH CITY

QUERY:

SELECT CITY,COUNT(NAME)

FROM STUDENT

GROUP BY CITY;

CONDITION: QUERY TO FIND AVG MARKS IN EACH CITY IN ASCENDING ORDER

QUERY:

SELECT CITY, AVG(MARKS)

FROM STUD

GROUP BY CITY

ORDER BY AVG(MARKS) ASC

CONDITION: FIND THE TOTAL PAYMENT ACCORDING TO EACH PAYMENT METHOD

QUERY:

SELECT MODE, COUNT(CUST)

FROM PAYMENT

GROUP BY MODE

HAVING CLAUSE

USED WHEN WE WANT TO APLLY ANY CONDITION AFTER GROUPING

CONDITION: COUNT NUMBER OF STUDENTS IN EACH CITY WHERE MAX MARKS CROSS 90

SELECT CITY,COUNT(NAME)

FROM STUDENT

GROUP BY CITY

HAVING MAX(MARKS)>90

WHERE CANT BE USED BECAUSE:

WHERE APPLIES CONDITION ON THE ROWS

HAVING APPPLIES CONDITION ON THE GROUP

*******GENERAL ORDER*******

SELECT COLUMN

FROM TABLE

WHERE COND

GROUP BY COLUMN

HAVING CONDITION

ORDER BY COLUMN ASC/SDESC

TABLE RELATED QUERIES

UPDATE

UPDATE TABLE_NAME

SET COL1=VAL1,COL2=VAL2

WHERE COND

CONDITION: TO CHANGE ALL THE GRADES THAT ARE A TO A+

QUERY :

UPDATE STUDENT_TABLE

GRADE='A+'

WHERE='A'

IN MY_SQL TO REMOVE THE SAFE MODE USE THIS COMMAND:

SET SQL_SAFE_UPDATES=0

DELETE

DELETE FROM TABLE_NAME

WHERE COND

CONDITION: DELETE STUDENT DETAILS WHERE MARKS <20

DELETE FROM STUDENT

WHERE MARKS<20

FORIENGN KEY

CASCADING FOR FK

ON DELETE CASCADE

ON UPDATE CASCADE

WHEN THE ROWS OF PRIMARY KEY OF THE PARENT TABLE IS UPDATED OR DELETED THEN THE ONE THAT REFERENCE TO THE CHILD TABLE ALSO IS CHANGED

SYNTAX:

CREATE TABLE STUDENT{
ID INT PRIMARY KEY;

COURSEID INT;

FOREIGN KEY(COURSEID) REFERENCES DEPT(ID)

ON UPDATE CASCADE

ON DELETE CASCADE

};

ALTER

TO CHANGE THE SCHEMA

ADD COLUMN

ALTER TABLE TABLE_NAME

ADD COLUMN COL_NAME DATATYPE CONSTRAINT

DROP COLUMN

ALTER TABLE TABLE_NAME

DROP COLUMN COL_NAME

CHANGE COLUMN

ALTER TABLE TABLE_NAME

CHANGE COLUMN OLD_COL_NAME NEW_NAME NEW_DATATYPE NEW _CONSTRAINT

MODIFY COLUMN

ALTER TABLE TABLE_NAME

MODIFY COL_NAME NEW_COL_NAME NEW_DATA_TYPE NEW_CONSTRAINT

RENAME TABLE

ALTER TABLE TABLE_NAME

RENAME TO NEW_NAME

TRUNCATE

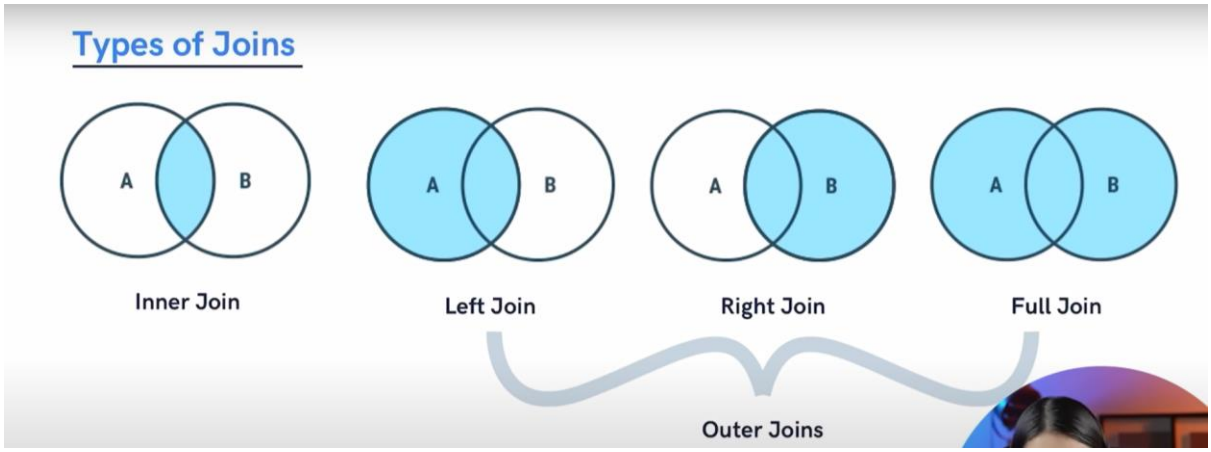
TRUNCATE TABLE TABLE_NAME

DROP-→DELETES THE TABLE

TRUNCATE-→DELETE TABLE DATA

JOIN

Join is used to combine rows from two or more tables, based on a related column between them



INNER JOIN

```
SELECT COL
FROM TABLE_A
INNER JOIN TABLE_B
ON TABLE_A.COL=TABLE_B.COL
```

ALIAS -→USING AS

LEFT JOIN

```
SELECT COLUMN
FROM TABLE_A AS A
LEFT JOIN TABLE_B AS B
ON A.COL=B.COL
```

RIGHT JOIN

```
SELECT COLUMN
FROM TABLE_A AS A
RIGHT JOIN TABLE_B AS B
ON A.COL=B.COL
```

FULL JOIN

```
SELECT * FROM STUDENT AS A
LEFT JOIN COURSE AS B
ON A.ID=B.ID
UNION
SELECT * FROM STUDENT AS A
RIGHT JOIN COURSE AS B
ON A.COL=B.COL
```

LEFT EXCLUSIVE JOIN

```
SELECT * FROM STUDENT AS A
LEFT JOIN COURSE AS B
ON A.ID=B.ID
WHERE B.COL IS NULL
```

RIGHT EXCLUSIVE JOIN

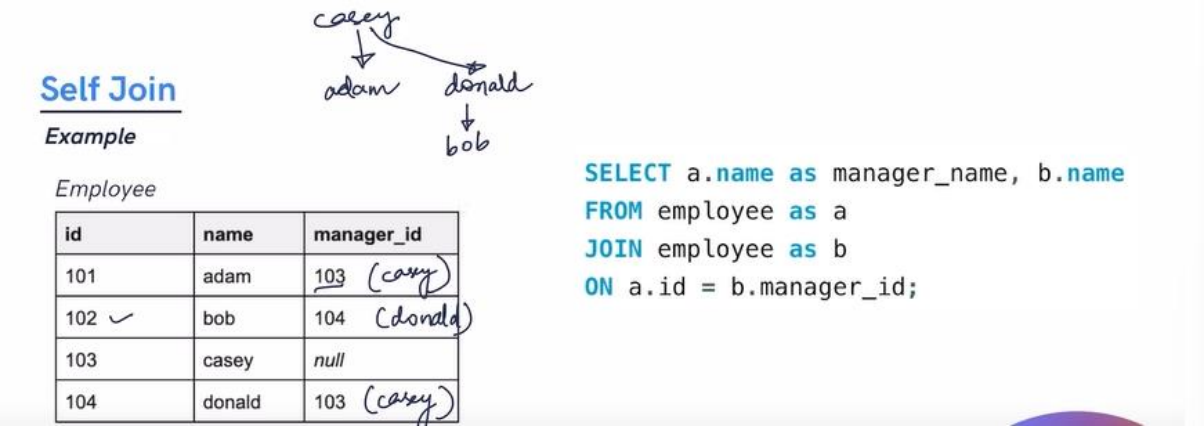
```
SELECT * FROM STUDENT AS A
RIGHT JOIN COURSE AS B
ON A.ID=B.ID
WHERE A.ID IS NULL
```

FULL EXCLUSIVE JOIN

```
SELECT * FROM STUDENT AS A
LEFT JOIN COURSE AS B
ON A.COL=B.COL
WHERE B.COL=NULL
UNION
SELECT * FROM STUDENT AS A
RIGHT JOIN COURSE AS B
ON A.ID=B.ID
WHERE A.ID=NULL
```

SELF JOIN

JOINS SELF TABLE



UNION

SELECT COL FROM TABLE_A

UNION

SELECT COL FROM TABLE_B

MUST:

EVERY SLECT SHOULD HAVE SAME NO.OF COLUMNS

COLUMNS MUST HAVE SIMILAR DATA TYPES

COLUMNS IN EVERY SELECT SHOULD BE IN SAME ORDER

UNION→NO DUPLICATES

UNION ALL→ALLOWS DUPLICATES

SUB QUERIES

A SUB QUERY OR INNER QUERY OR NESTED QUERY IS A QUERY WITHIN ANOTHER SQL QUERY

QUESTION: SELECT NAMES OF STUDENT WHO SCORED MORE THAN THE AVG MARKS

QUERY:

1. TO FIND THE AVERAGE MARKS
2. TO SELECT THE NAME THAT ARE ABOVE THE AVERAGE

```
SELECT NAME, MARKS
FROM STUDENT
WHERE MARKS>(SELECT AVG(MARKS) FROM STUDENT)
```

Q: FIND THE NAMES OF ALL STUDENTS WITH EVEN ROLL NUMBERS

QUERY:

```
SELECT NAME
FROM STUDENT
WHERE ROLL_NUMBER IN (SELECT ROLL_NUMBER FROM STUDENT WHERE ROLL_NUMBER %2==0)
```

Q: FIND THE MAX MARKS FROM STUDENTS OF DELHI

```
SELECT MAX(MARKS)
FROM (SELECT *
FROM STUDENT
WHERE CITY=”DELHI”)AS TMP
```

VIEWS

A VIEW IS VIRTUAL TABLE BASED ON THE RESULT-SET OF SQL

```
CREATE VIEW VIEW1 AS
SELECT ROLLNO, NAME FROM STUDENT;
SELECT * FROM VIEW
DROP VIEW VIEW1
```