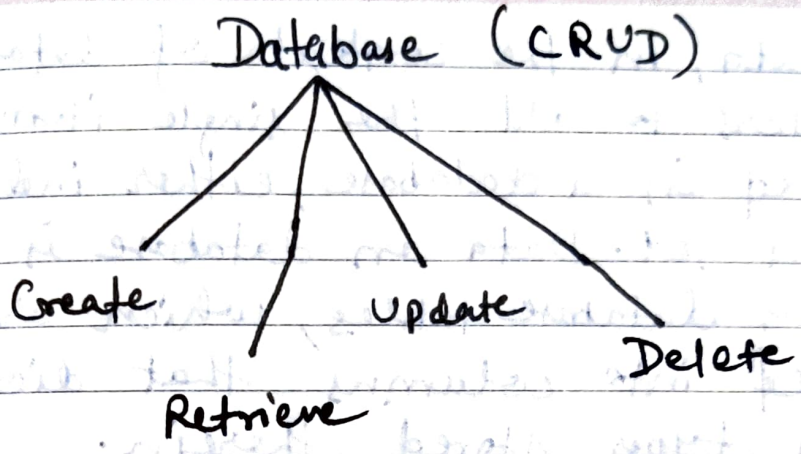
FEBRUARY 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

MYSQL

Friday • JANUARY 03

WK 01 (003-363)



- Whenever we interact with any database, we perform any one from these 4 operations.

Eg :- Create :- eg :- creating a database or a table.

Retrieve :- eg :- login with credentials on any website

Update :- eg :- editing a comment or msg which has already been posted.

Delete :- eg :- deleting a chat (whatsapp chat) so that anyone else can't see.

2020

30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Data :- Data, in the context of databases, refers to all the single items that are stored in a database, either individually or as a set. Data in database is primarily stored in database tables, which are organised into columns that dictate the data types stored therein.

Requirements of the Data :-

- Integrity
- Availability
- Security
- Independent of Application
- Concurrency.

05 Sunday **Dependency of flat files :-**

- Dependency of program on physical structure of data.
- Complex process to retrieve data
- Loss of data on concurrent access.
- Inability to give access based on record.
- Data redundancy.

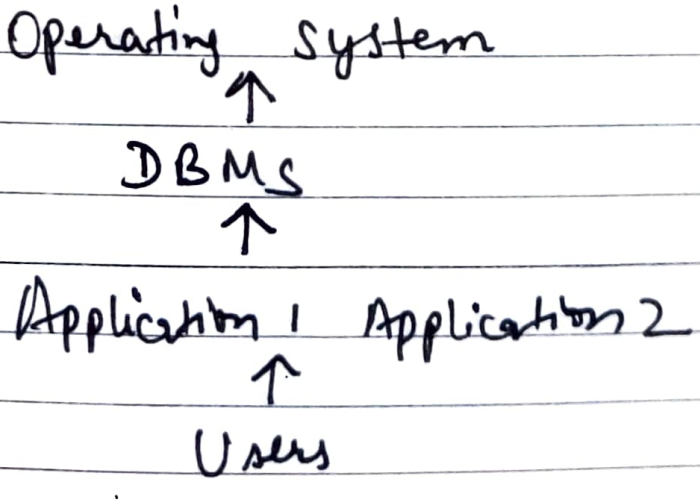
Use-cases for files :-

- On device access (offline access)
- Storing frequent incoming data
- Storing not so important data.

Database :- A database is a shared collection of logically related data and description of these data, designed to meet the information needs of an organization.

DBMS :- A DBMS is a software system that enables users to define, create, maintain & control access to the database. Database systems typically have high cost & they require high end hardware configuration.

An Appl^n program interacts with a database by issuing an appropriate request. (typically a SQL statement)



2020

07

(007-359) WK 02 | JANUARY • Tuesday

M	T	W	T	F	S	S
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

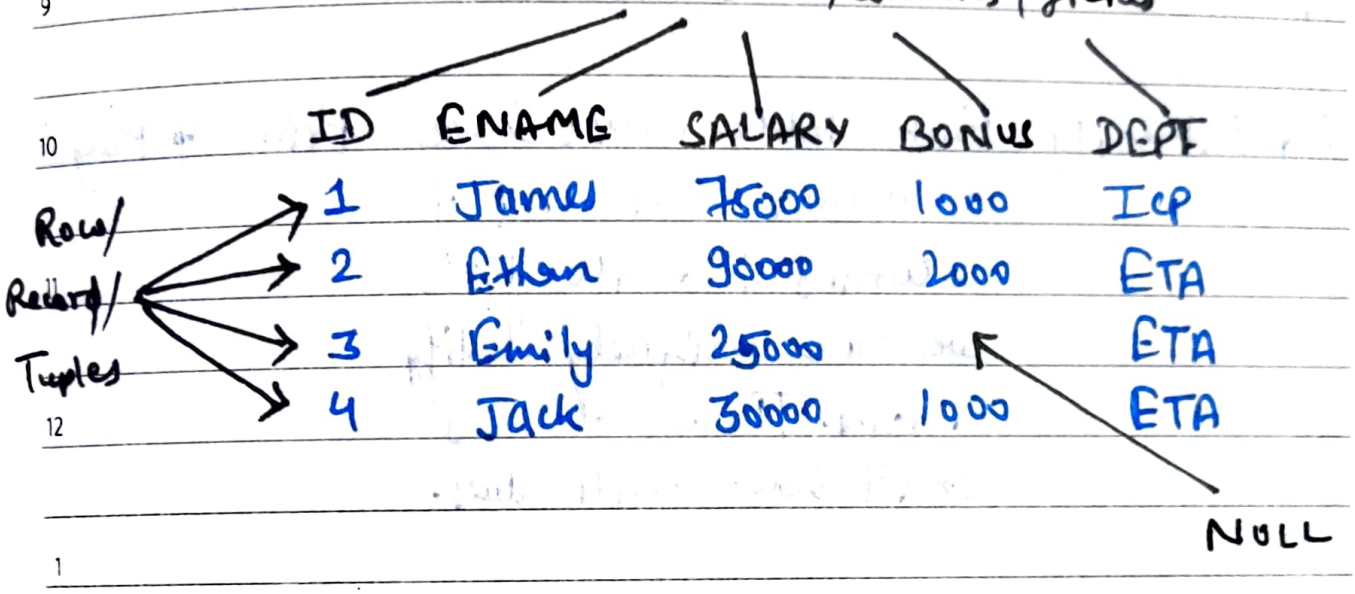
Function of DBMS :-

- Data Management → Store, retrieve & modify data
- Integrity → Maintain accuracy of the data.
- Security → Access to authorised users only.
- Concurrency → Simultaneous data access for multiple users.
- Transaction → Modifications to the database must either be successful or must not happen at all.
- Security → Access to authorised user only.
- Utilities → Data import/Export, user management, backup, logging.

Types of Database :-

- Hierarchical → Tree Structure
- Network → Graph Structure
- Relational → Tabular Structure
- NoSQL → Key-Value pairs Graph Document

Relational Model :- Collection of tables.



Relation is usually represented as :-
Employee (ID, ENAME, SALARY, BONUS, DEPT)

Total no. of rows :- Cardinality of the relation (table).

Total no. of columns :- Degree of the relation (column).

Domain :- {ICP, ETA, IVS} The value of Dept column will be among these three values.

Data Integrity & Constraints :- Data integrity refers to maintaining & assuring the accuracy & consistency of data over its entire life-cycle. DBMS ensures data integrity through constraints which are used to restrict data that can be entered or modified in the database. Database System offers three types of integrity constraints.

2020

Integrity Type	Definition	Enforced through
Entity Integrity	Each table must have a column or a set of columns through which we can uniquely identify a row. These columns can't have null values.	Primary Key
Domain Integrity	All attributes in a table must have a defined domain i.e, a finite set of values which have to be used. When we assign a data type to a column we limit the values that it can contain. In addition we can also have value restriction as per business rules eg. Gender must be F or M.	Data types, CHECK CONSTRAINT

Integrity Type	Definition	Enforced through
Referential Integrity	Every value of a column in a table must exist as a value of another column in a different (or the same) table.	FOREIGN KEY

Database Keys!-

A DBMS key is an attribute or set of attributes which helps us uniquely identify a row in a relation | table.

Why do we need keys?

- To uniquely identify a row.
- To enforce Data integrity & constraints.
- To establish r/ps between tables.

Types of Keys!-

- Super key
- Candidate key
- Primary key
- Foreign key
- Alternate key
- Composite key
- Compound key
- Surrogate key.

12 Sunday

Super Key :- Suppose we have these columns in our DB.

emp_id | aadharcard | empname | DOB | Salary | email

unique keys {empid, aadhar, email}

Set of ~~rows~~ Keys which can identify ~~the~~ two rows.

{empid, aadhar, email, emp+aadhar, aadhar+email, email+emp_id, emp_id+aadhar+email}

Candidate Key :- Minimum no. of columns which can uniquely identify the rows.

emp_id, aadhar, email are candidate keys.

emp_id + aadhar
aadhar + email
email + emp_id
emp_id + aadhar + email

} can't be a candidate key because emp_id / aadhar / email are itself a candidate key.

Primary Key :- Primary key is that key which is selected from candidate key.

There can be only one primary key.

- Primary key can't be NULL.
- Primary key should be unique.

emp_id suits all the above 2 cases bcz emp_id is numeric as well as small, so empid will be P. key.

email

2020

DECEMBER 2019

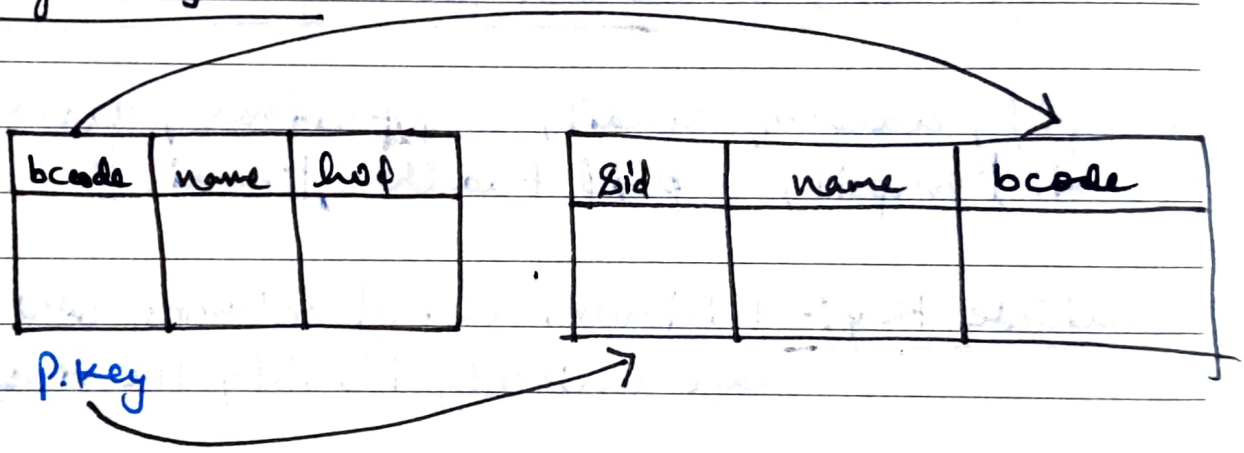
M	T	W	T	F	S	S
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

→ (Candidate key - Primary key.)

Alternate key :- let's suppose if we have

{emp_id, email, aadhar} as candidate key & I have selected emp_id as a primary key then rest 2 key email & aadhar will be the alternate key.

Foreign key :-



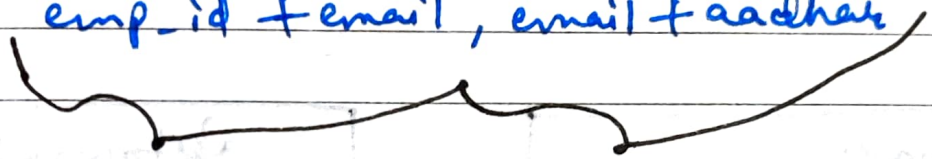
When primary key of one table is used in the other table then that primary key of one table is known as foreign key of the other table.

The whole idea is to maintain Referential Integrity.

- If the branch is removed from the college, then all student of the 2nd table will be removed automatically.
- If the branch changed in future then, branches of all the students will also be changed in the other table.

Composite Key :- Composite key are those super key which has more than 1 keys.

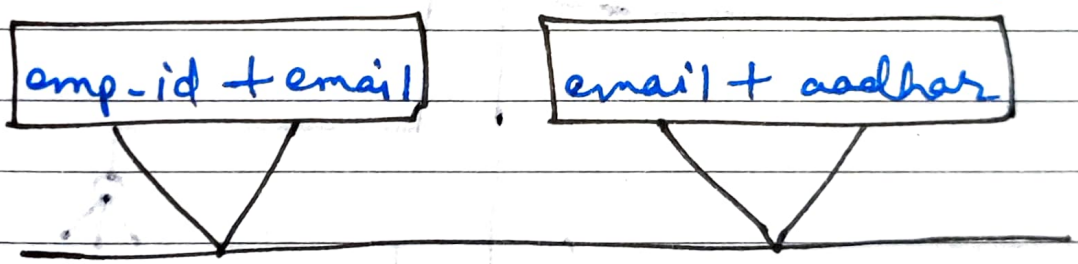
eg :- emp_id + email , email + aadhar



Combination of keys.

Compound Key :- Compound key is that composite key which has atleast 1 foreign key.

eg :- emp_id + email , email + aadhar



Atleast one key should be foreign key in the compound key.

Surrogate Key :- Let's suppose we have name, city & DOB as columns, we can't select any column as ~~primary~~ key because name, city & DOB can't uniquely identify the rows. So, we will make id by own & that will be key.

name	city	DOB

⇒

id	name	city	DOB
			2020

Surrogate key.

16

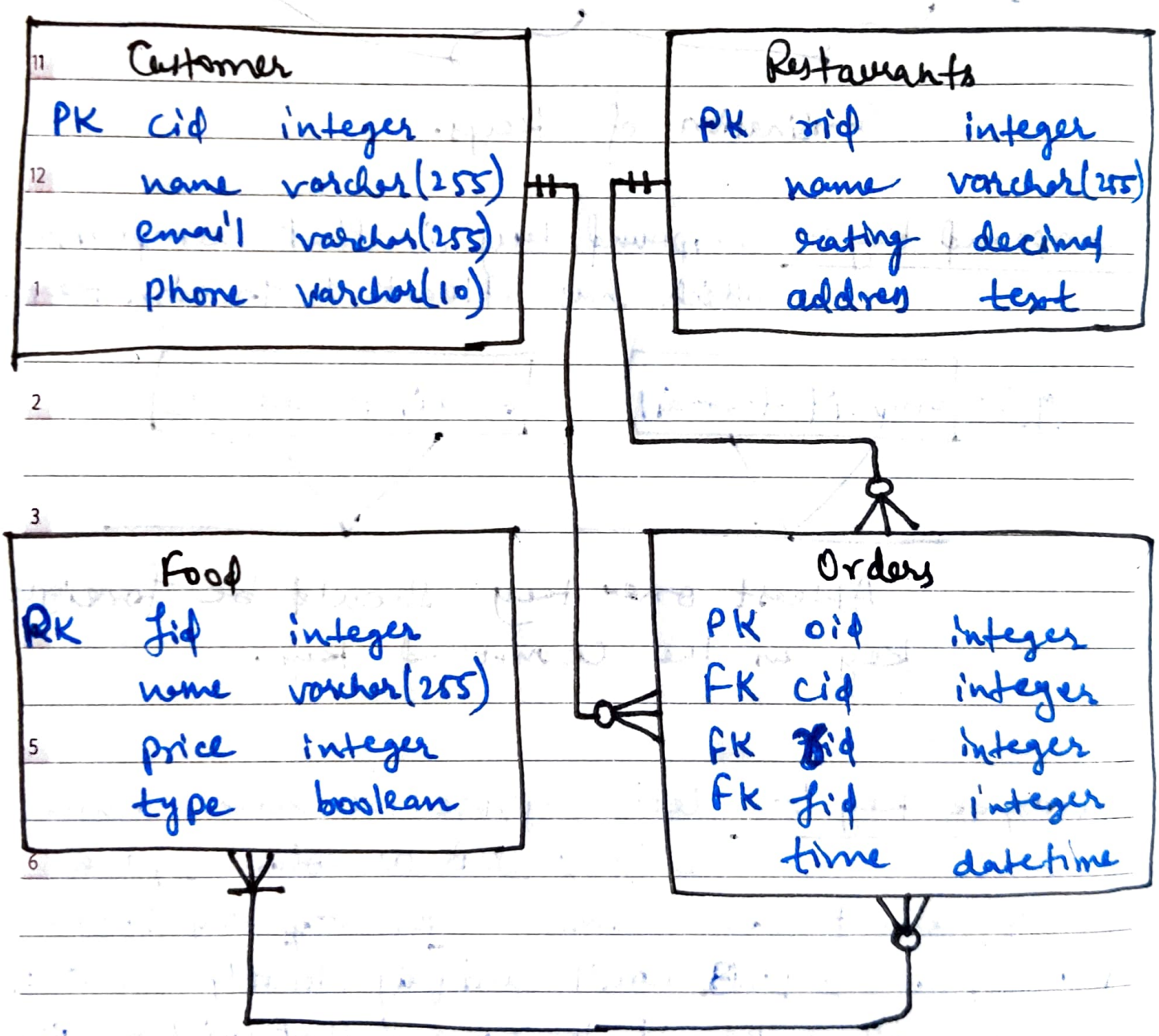
(016-350) WK 03

JANUARY • Thursday

M	T	W	T	F	S	S
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Entity Relationship Model:-

Swiggy Database ER Digram



2020

Customer & order :- one to many

A customer can have multiple orders but an order can be associated with only one customer.

Restraunt & order :- one to many

A restraunt can receive multiple orders but each order can have only one restaurant.

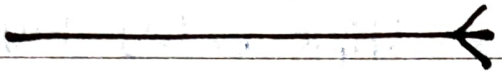
Order & food :- many to many

One order can have multiple food items & one single food item can be a part of multiple orders.

CROW FOOT NOTATION :-

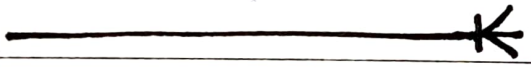
9

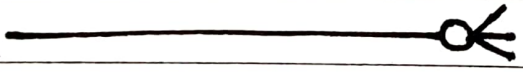
10 One

11  Many

12 One & only one

1 Zero Or One

2  One or many

3  Zero or many

4

5

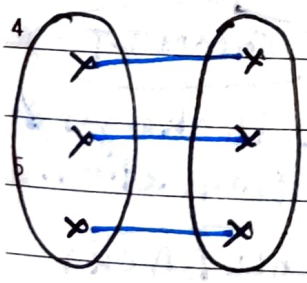
6

19 Sunday

2020

Cardinality of Relationships:- Cardinality of relationships is the number of instances in one entity which is associated to the no. of instances in other.

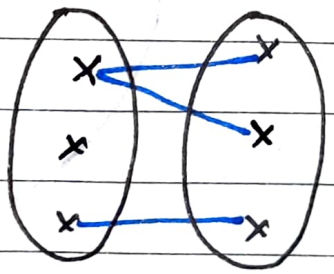
For the relationship b/w Employee & computer it helps us answer questions like how many computers can be allocated to an employee, can computers be shared b/w employees, can employee exists without being allocated a computer etc. eg. if 0 or 1 computer can be allocated to 0 or 1 employees then the cardinality of relationship b/w these two entities will be 1:1.



1:1

one to one

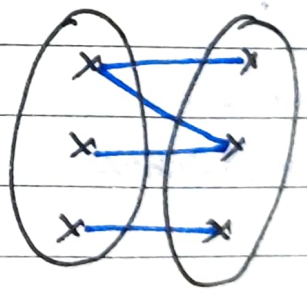
Project partner in college, there is only one option for the next person & the next person also have only one option.



1:N

one to many

In one branch there can be multiple student but on student can have 1 branch.



M:N

many to many.

one student can give multiple exams and 1 exam can give multiple student.

2020

21

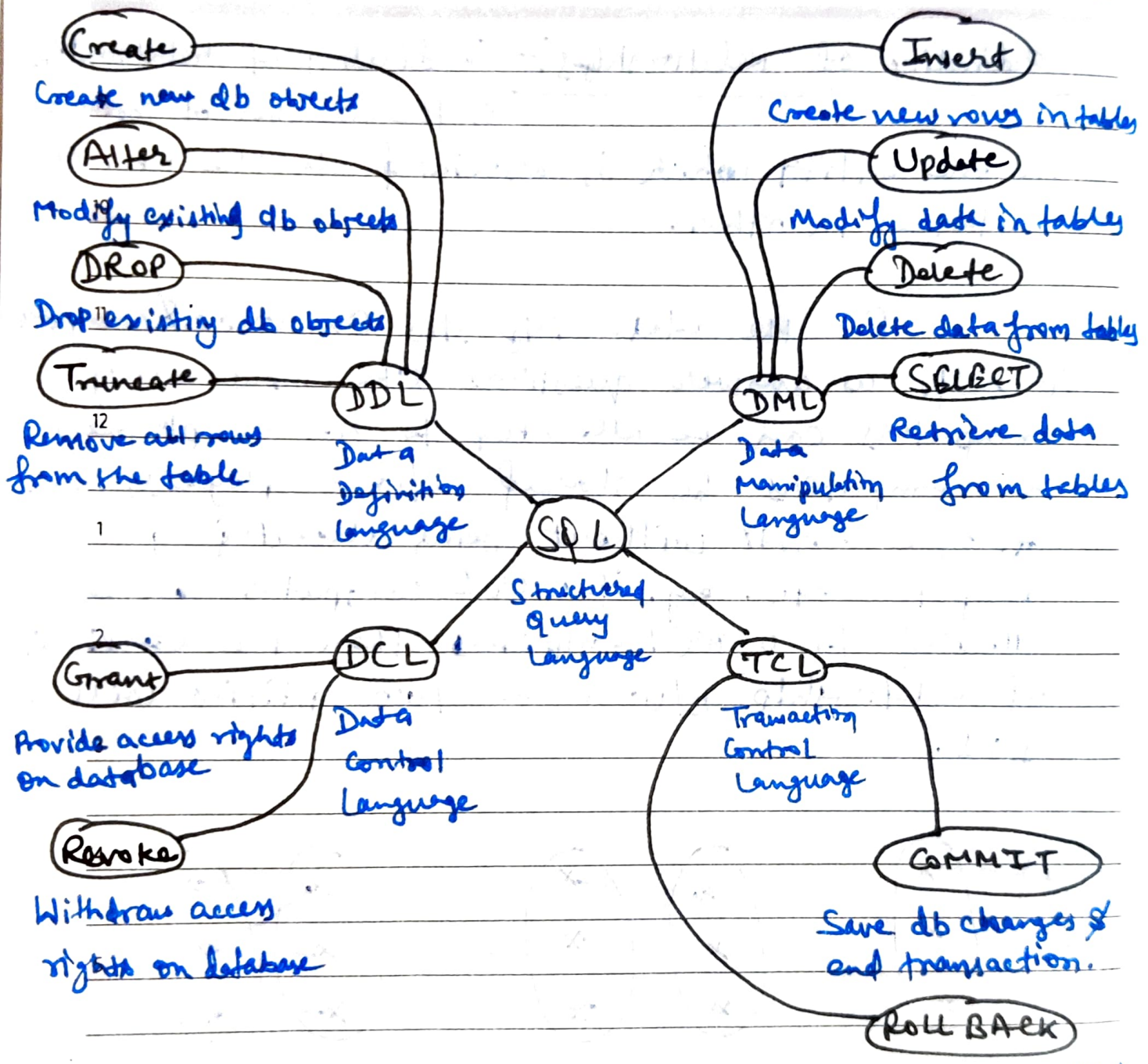
(021-345) WK 04

Types of SQL Commands

JANUARY • Tuesday

DECEMBER 2019

M	T	W	T	F	S	S
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29



DDL - DATA DEFINITION LANGUAGE

Create Database :-

9

CREATE DATABASE IF NOT EXISTS dbname

10

Drop Database :-

11

DROP DATABASE dbname

12

CREATE TABLE :-

1

```
CREATE TABLE IF NOT EXISTS users (
2    id integer,
     name varchar (255),
3    email varchar (255),
     password varchar (255)
4   )
```

DELETE TABLE :- (completely delete the table from the database)

5

DROP TABLE table_name.

6

CONSTRAINTS :-

① NOT NULL :-

```
CREATE TABLE IF NOT EXISTS
table_name (
id integer NOT NULL,
name varchar (255),
)
```

2020

② UNIQUE :-

```
CREATE TABLE IF NOT EXHISTS
table_name (
    id integer NOT NULL,
    name varchar (255),
    email varchar (255),
    CONSTRAINT u_email UNIQUE(email)
)
```

This CONSTRAINT u_email means if we need to change the constraint for email in future, so that it can have duplicate values in future so we have just given the constraint name so, that we can edit the functionality if needed in upcoming times.

UNIQUE can also be written as :-

```
CREATE TABLE IF NOT EXHISTS table_name (
    id integer NOT NULL,
    name varchar (255),
    email varchar (255) UNIQUE
)
```

Two or more columns value shouldn't be repeated at the same time in the table:-

```
CREATE TABLE IF NOT EXISTS table_name(
    id integer NOT NULL,
    name varchar(255),
    email varchar(255),
    UNIQUE(name, email)
```

③ PRIMARY KEY:-

```
CREATE TABLE users(
    id integer NOT NULL PRIMARY KEY,
    name varchar(255),
    email varchar(255) UNIQUE
)
```

Other way:-

```
CREATE TABLE users(
    id integer NOT NULL,
    name varchar(255),
    email varchar(255) UNIQUE,
    PRIMARY KEY(id)
```

If 2 columns are Primary Key of my table:

```
CREATE TABLE users (
    id integer NOT NULL,
    name varchar(255),
    email varchar(255),
    PRIMARY KEY (id, email)
```

(4) FOREIGN KEY :-

Let's suppose we already have users table and we want to make one more table named orders so we will make a foreign key named user_id in Orders table which will reference to id column of users table.

```
CREATE TABLE orders (
    order_id integer,
    user_id integer,
    time_of_order datetime,
    PRIMARY KEY (order_id)
FOREIGN KEY (user_id) REFERENCES users(id)
```

(5) CHECK:- Using CHECK, we can apply some conditions on ~~table~~ columns. If the condition won't satisfy then it will not allow the values to be inserted.

```
CREATE TABLE students (
    sid integer PRIMARY KEY,
    sname varchar (255) NOT NULL,
    email varchar (255) NOT NULL UNIQUE,
    age integer CHECK (age > 6 AND age < 25)
```

(6) DEFAULT:- If no value is entered in that particular column on which default is applied then the default value which is set will be entered as the value in the table.

```
CREATE TABLE passenger (
    pid integer PRIMARY KEY,
    pname varchar(255) NOT NULL,
    gender varchar(255) DEFAULT "Others"
)
```

If no any value is entered into the gender column then the default value which is given (others) will be taken automatically in the table.

```
CREATE TABLE passenger(
    id integer PRIMARY KEY,
    name varchar(255) NOT NULL,
    journey_date datetime DEFAULT CURRENT_
    TIMESTAMP
)
```

If the journey date is not provided by the person who is inserting the data then current time will be inserted as the journey_date inside the table.

(7) AUTO INCREMENT :- If no any value is given to the column on which auto increment constraint is applied then the value will be auto incremented by the last value inside the table.

ALTER TABLE :-

ALTER TABLE is used to :-

1. Add Columns
2. Delete Columns
3. Modify Columns
4. add/remove constraints

① Add columns :-

Let's suppose we have students table and in students table we have sid, sname, email, age columns inside it and now we are going to add one more column inside it that is college

```
ALTER TABLE students ADD COLUMN college
varcher(255) NOT NULL
```

② DELETE COLUMN :-

If we remove age from students column then we need to write a query :-

```
ALTER TABLE students DROP COLUMN age
```

③ Modify column :-

If we have to change sname column whose datatype is varcher(255) & it will be converted to integer then,

```
ALTER TABLE students MODIFY COLUMN
sname integer.
```

2020

④ ADD / REMOVE CONSTRAINTS :-

① ADD Constraint :-

- First we will add a column in passenger ~~column~~ table.

ALTER TABLE passenger ADD COLUMN email Varchar (255) NOT NULL

↑ constraint

email column has been added now to the passenger table, now we need to add constraint on email column (UNIQUE)

ALTER TABLE passenger ADD CONSTRAINT P_email UNIQUE (email)

② REMOVE CONSTRAINT :-

Removing constraint from a particular column of a table.

ALTER TABLE passenger DROP CONSTRAINT P_email

2020 Now we can repeat a single email multiple no. of times in email column because UNIQUE CONSTRAINT has been removed now.

TRUNCATE :- Completely delete all rows of the table.

TRUNCATE TABLE table_name.

DML :- DATA MANIPULATION LANGUAGE

INSERT :-

→ INSERTing into Students table which has sid, sname, email & college as columns.

```
INSERT INTO students (sid, sname, email, college) VALUES (1, "Shivang", "shivang@gmail.com", "NIT NAGALAND")
```

→ Inserting into students table without mentioning COLUMN NAME (This will apply only if we need to insert ~~all values~~ into all columns)

```
INSERT INTO students ~~(sid, sname, email)~~ VALUES (2, "Anushk", "anushk@gmail.com", "Satya Sai University")
```

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Inserting Multiple values simultaneously :-

```
INSERT INTO students VALUES (
    3, "Amit", "amit@gmail.com", "BVU"),
( 4, "Shreyas", "shrey@gmail.com", "BVU"),
( 5, "Sangram", "sang@gmail.com", "ITM")
```

RETRIEVE / SELECT :-

- TO RETRIEVE all the data from the table :-

All Cols :-

```
SELECT * FROM table_name
```

- **Filter Cols :-**

Working on Titanic Dataset. We need to find the rows with Passenger Name, Sex & survived.

```
SELECT Name, Sex, Survived FROM titanic
```

02 Sunday

- **Alias As :-** Giving the column name Alias name so that we can view our table based on whatever we want the column name.

2020

```
SELECT Name AS PassengerName, Sex AS Gender, Survived FROM train.
```

- **Expression :-**

Let's suppose, we are checking how many people were travelling with there family & how many members were there in the family of the people who were travelling in titanic.

```
SELECT Name, SibSp + Parch AS Family
FROM titanic.
```

What would be the current age of the people who were onboarding on titanic?

```
SELECT Name, Age + 102 AS CurrentAge
FROM titanic.
```

- **Content :-** Let's suppose the government said whoever was there on titanic will be given as 100000 as compensation.

```
SELECT Name, 100000 As Compensation FROM
titanic.
```

- **Distinct Single Column :-** unique values

If we need to check what is the gender of the people who was on titanic?

```
SELECT DISTINCT Sex FROM titanic
```

- **Distinct multiple Column :-** combination of unique values b/w multiple columns -

```
SELECT DISTINCT Pclass, Embarked FROM train.
```

- **Comparison Operator :-** To check the details of the people who has not survived on titanic.

```
SELECT * FROM titanic WHERE Survived=0
```

To check how many people are more than 50 years old.

```
SELECT * FROM titanic WHERE Age>50
```

• AND / OR / BETWEEN / NOT OPERATOR :-

AND Operator :- Checking the passenger who was in pclass 3 and not survived.

```
SELECT * FROM train WHERE Pclass = 3
AND Survived = 0
```

BETWEEN Operator :- Checking the age of passengers between 10 to 15.

```
SELECT * FROM train WHERE Age
BETWEEN 10 AND 15.
```

OR Operator :- Example not related to titanic.

Fetching records which satisfy any 1 condition.
either city = delhi will show or city = banglore will show.

```
SELECT * FROM Customers WHERE City = delhi
OR CITY = banglore
```

NOT operator :- Show the city except muzaffarpur,

```
SELECT * FROM Customers WHERE NOT city
= muzaffarpur.
```

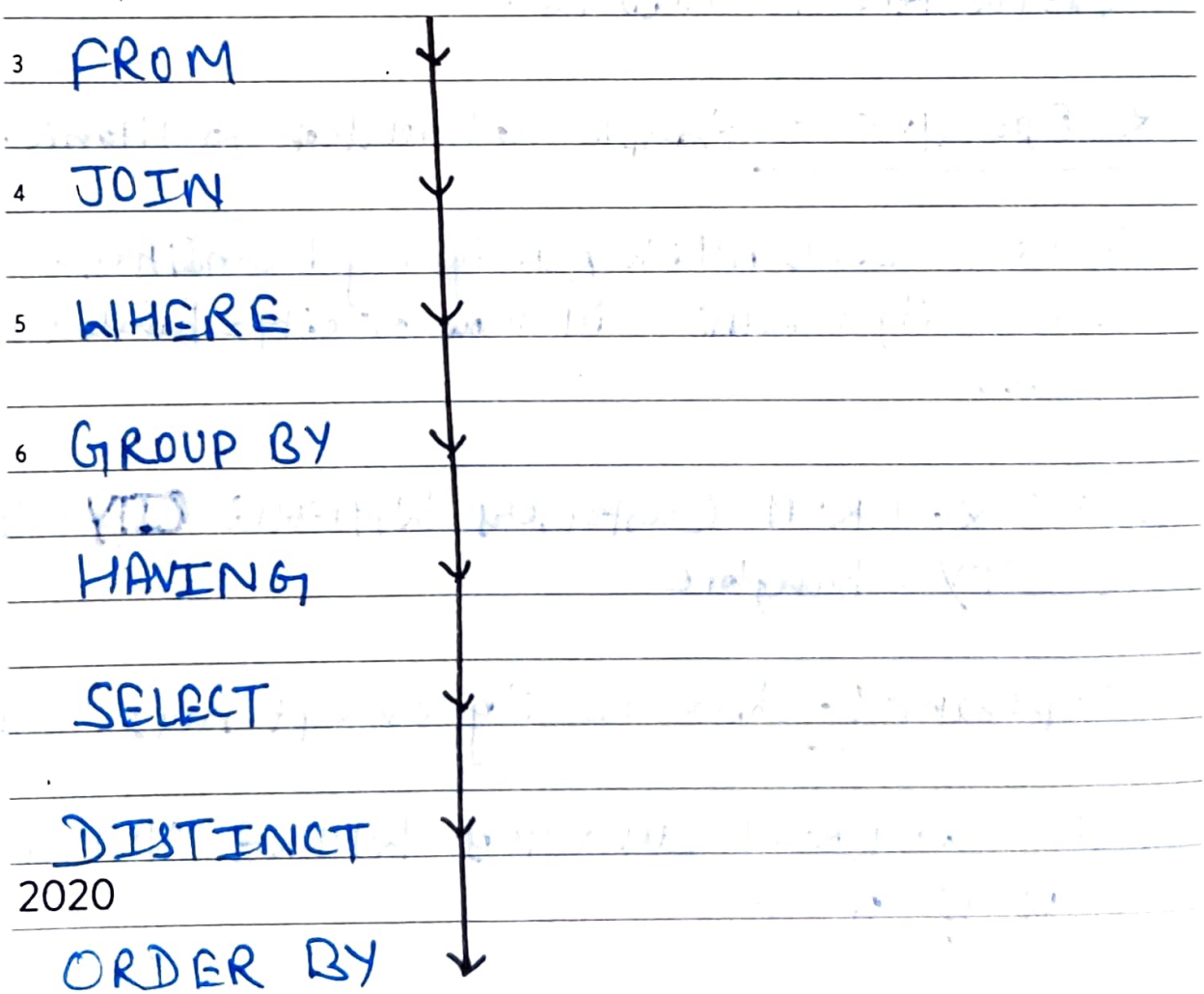
M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Combining AND, OR & NOT :-

```
SELECT * FROM Customers
WHERE Country = Germany AND (City = Berlin
  OR City = NewYORK

SELECT * FROM Customers
WHERE NOT Country = Germany AND NOT
Country = USA
```

ORDER of Query Execution :-



SELECT Id, EName FROM Employee WHERE Salary >30000

FROM will be executed first.
WHERE will be executed next.
SELECT Will be executed at last.

SELECT DISTINCT Designation FROM Employee WHERE Bonus IS NOT NULL

FROM will be executed first.
WHERE will be executed next.
SELECT will be executed next.
DISTINCT will be executed at last.

IN/NOT IN:-

IN:- We use in as an alternative of multiple OR.

Let's suppose we have a movie dataset & we have to populate the title of movies whose genre is like comedy or Action.

Using OR we can write:- SELECT title FROM movies WHERE genre LIKE 'comedy' OR genre LIKE 'Action'

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

IN :- for replacing multiple OR

Let's suppose, if we have to populate the movies whose genre is Comedy, Action or Horror. So we have to write multiple OR in case we want to see the values. So, in order to avoid multiple OR we will use IN.

```
SELECT title, genre FROM movies WHERE genre IN ("Action", "Horror", "Comedy")
```

NOT IN :- It will show the values where genre is not Action, Horror, Comedy. It is inverse of IN.

```
SELECT title, genre FROM movies WHERE genre NOT IN ("Action", "Horror", "Comedy")
```

Wildcards :- Used to search words inside the table column.

- Searching student name starting with word ana.

```
SELECT Name FROM students WHERE Name LIKE 'ana%';
```

• Searching student name having 'ber' in their name.

```
SELECT Name FROM students WHERE Name LIKE '%ber%';
```

• Searching student name having 'Ka' at their last.

```
SELECT Name FROM students WHERE Name LIKE 'Ka%';
```

• Searching the name of student whose name ends with Khan or Kapoor.

```
SELECT Name FROM students WHERE Name LIKE "%Khan" OR ~~actor~~ Name LIKe "%.Kapoor"."
```

→ This will not work because, wildcard query should have LIKE in it.

```
SELECT Name FROM students WHERE Name IN ("%.Khan%.", "%.Kapoor")
```

This code won't work because there should be LIKE in wildcards. It won't work with IN.

- Searching the name of students which has only 5 letters in their name.

```
SELECT Name FROM students WHERE Name LIKE "_____";
```

This query will show the students name having 5 letters.

- Searching the name of students whose name starts with A and having 6 letters in the name.

```
SELECT Name FROM students WHERE Name LIKE "A_____"
```

Charlist Wildcard :-

- The following SQL statement selects all customers with a City starting with "b", "s", or "p".

```
SELECT * FROM Customers WHERE city LIKE '[bsp]%';
```

- The following SQL statement selects all Customers with a city starting with "a", "b", or "c".

```
SELECT * FROM Customers WHERE city LIKE '[a-c]%';
```

(a hyphen c)

- The two following SQL statements selects all customers with a city NOT starting with "b", "s", or "p".

SELECT * FROM Customers WHERE City LIKE '[!bsp]%'

OR

SELECT * FROM Customers WHERE City NOT LIKE '[bsp]%'

UPDATE :-

- Updating all names of the passenger column with the name "Rahul".

UPDATE ~~passenger~~ passenger SET name='Rahul'

- Updating all the passenger name with the name "Rohit" where email is having gmail in it.

UPDATE passenger SET name = "Rohit" WHERE email LIKE '%.gmail%'

- Updating two columns simultaneously.

```
UPDATE passenger SET name = "Ankur",
email = "def@gmail.com WHERE email LIKE
"%.yahoo%."
```

DELETE :- Deleting rows.

- Deleting the 1st ~~column~~ row.

```
DELETE FROM passenger where pid = 1.
```

- Deleting with condition / expression :

```
DELETE FROM passenger WHERE id > 2
AND email LIKE "%.yahoo%"
```

- Deleting the whole table Like truncate.

```
DELETE FROM passenger WHERE 1
```

- WHERE 1 is the condition which is true for all rows.

Functions :-

- ABS
- ROUND
- CEIL
- FLOOR
- UPPER / LOWER
- CONCAT
- LENGTH
- SUBSTR (string, start (1), length)

** Aggregate function **

- MIN / MAX / SUM / AVG
- COUNT / COUNT WITH DISTINCT

ABS :- It gives us the absolute value, It will remove the negative sign & make it positive.

SELECT title, ABS((~~title~~ india_gross - budget)) AS profit FROM movies.

India gross is the amount earned by movies. Budget is the total amount invested on movies. At first india_gross - budget will let us know whether the movies are flop or hit. If it gives minus (-) value the the movie is flop. So, we are putting the absolute function on the india_gross - budget.

2020

M T W T F S
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

ROUND :- It rounds of decimal values & convert it to integer.

Converting the runtime of movies from minutes to hours & apply ROUND Function.

~~SELECT, title, (runtime / 60) AS runtime hrs FROM movies.~~

```
SELECT title, ROUND((runtime/60)) As runtime_hrs FROM movies.
```

Round off with decimal places.

```
SELECT title, ROUND((runtime/60),1) As runtime_hrs FROM movies.
```

Rounding off with 1 decimal places.

CEIL :- It will take the top value like (2.3 = 3, 4.8 = 5, 3.6 = 4, 1.1 = 2)

16 Sunday

2.00 hrs to 2.99 hrs movies will be considered as 3 while using CEIL.

```
SELECT title, CEIL((runtime/60)) AS runtime_hrs FROM movies.
```

2020

FLOOR :- It will take the lowest value like (2.3 = 2, 4.8 = 4, 5.9 = 5)

2:00 hrs to 2.99 hrs movies will be considered as 2 while using FLOOR

```
SELECT title, FLOOR ((runtime / 60)) AS
runtime_hrs FROM movies.
```

UPPER / LOWER :- It will convert the lowercase or uppercase to uppercase in case of UPPER & In case of LOWER it will convert the lowercase or uppercase to lowercase.

```
SELECT UPPER (title) FROM movies

SELECT LOWER (title) FROM movies
```

CONCAT :- Let's suppose we need to see the actor & director in a single column. then we will have to concatinate the columns.

```
SELECT title, CONCAT (actor, ' ', director) AS
Crew FROM movies.
```

LENGTH :- Let's suppose we need to check the length of the name of movies.

```
SELECT title, LENGTH (title) as length
FROM movies.
```

SUBSTR :- (string, start(1), length) :-

Let's suppose if we want the first 5 alphabets of movies name, column.

```
SELECT title, SUBSTR (title, 1, 5) AS
short FROM movies.
```

Aggregate function :- MIN, MAX, SUM, AVG, COUNT, COUNT WITH DISTINCT

- To check Maximum budget of movies.
  `SELECT MAX (budget) FROM movies;`
- To check minumum earnin of movies.
  `SELECT MIN (india_gross) FROM movies;`

• Total income of all the indian movies.

```
SELECT SUM(india_gross) FROM movies;
```

• Average income of all the indian movies.

```
SELECT AVG(india_gross) FROM movies;
```

• Count of all the indian movies.

```
SELECT COUNT(*) FROM movies;
```

• Count With Distinct (counts the unique value/ items inside the table's column)

~~SELECT COUNT WITH DISTINCT~~

```
SELECT COUNT(DISTINCT(actor)) FROM movies;
```

It will give the total no. of actors as an o/p.

SORTING DATA :- Let's suppose we need to sort the movies based on most profitable movies.

```
SELECT title, (worldwide_gross - budget) AS profit FROM movies
```

→ This will show the data according to the profit.

SORT by descending order in terms of profit

```
SELECT title, (worldwide_gross - budget) AS profit
FROM movies ORDER BY profit DESC
```

2020

LIMIT :-

Top 5 profitable movies :-

```
SELECT title, (worldwide_gross - budget)
AS profit FROM movies ORDER BY profit
DESC LIMIT 5;
```

ORDER BY on 2 columns :-

Let's suppose we want the genre of movies in Ascending order as well as the title in Ascending order.

```
SELECT * FROM movies ORDER BY
genre, title;
```

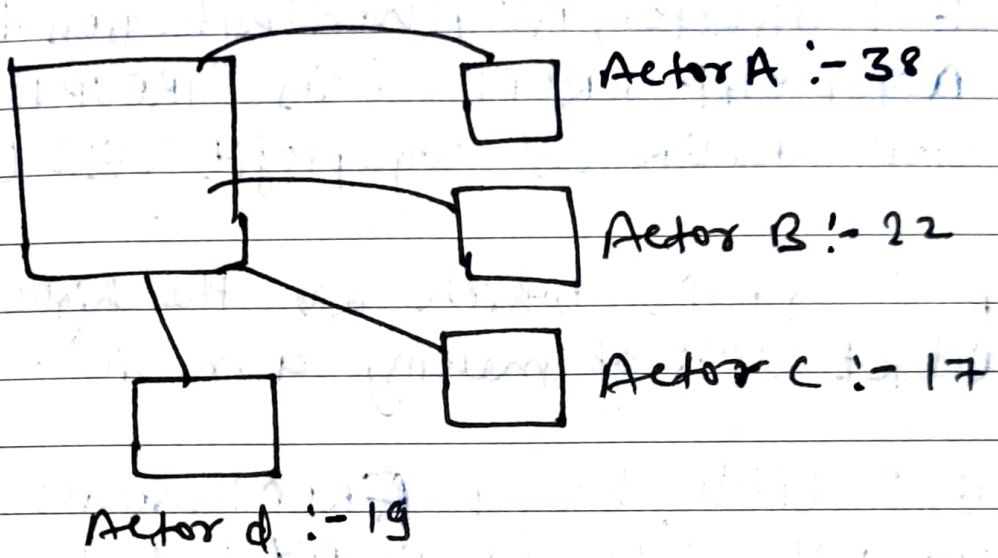
If we want genre in Ascending order & title in descending order.

```
SELECT * FROM movies ORDER BY
genre, title DESC;
```

ORDER BY can be used on multiple columns.

GROUP BY

GROUPING :- Top 5 actors who have done more number of movies.




```
SELECT actor, COUNT(*) AS num_movies FROM
movies GROUP BY actor ORDER BY num_movies
DESC LIMIT 5
```

Q Which genre movies has the maximum profit (Maximum 5 movies)

```
SELECT genre, SUM(worldwide_gross - budget)
AS profit ~~GROUP BY~~ FROM movies GROUP BY
genre ORDER BY profit DESC LIMIT 5
```

JANUARY 2020

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Q Which top 5 director has the highest profit on an average?

```
SELECT director, AVG(worldwide_gross-budget)
As avg_profit FROM movies GROUP BY
director ORDER BY avg_profit DESC LIMIT 5;
```

Q Top 5 movies which has the highest budget (cost of making a movie).

```
SELECT title, budget FROM movies
GROUP BY title ORDER BY budget DESC
LIMIT 5;
```

Q Which director & actor combination has the maximum profit earned? (5 combination)

```
SELECT actor, director, SUM(worldwide_gross
- budget) As profit FROM movies
GROUP BY actor, director ORDER BY profit
DESC LIMIT 5;
```

23 Sunday

Q Which actor has earned max in which genre? (Top 5)

```
SELECT actor, genre, SUM(worldwide_gross -
budget) As profit FROM movies GROUP BY
actor, genre ORDER BY profit DESC LIMIT 5;
```

Q Which top 10 actors has earn maximum in Bollywood?

```
SELECT actor, SUM(worldwide_gross-budget)
AS total_profit FROM movies GROUP BY
actor ORDER BY total_profit DESC LIMIT 10;
```

HAVING:-

Having is used for checking the condition in GROUP BY. (Just like we have WHERE in SELECT)

Q Show the name of those actors whose movies got more than 1000 openings on screen.

```
SELECT actor, AVG(screens) AS opening
FROM movies GROUP BY actor HAVING
opening > 1000 ORDER BY opening DESC;
```

Assessment:- Consider the following record of the customer table:

LASTNAME
Desouza

What will be the o/p of the following query?

```
SELECT LENGTH(SUBSTR(LASTNAME, 3, 7)) FROM
CUSTOMER
```

Q Which of the following aggregate functions can be applied on both number & character type columns?

9
10 • MIN • MAX • AVG • COUNT • SUM

Q 11 Table T exists with single column A having 4 rows with values -100, 200, NULL, 300.

12 What is the o/p of the following query?

1 SELECT MAX(A) FROM t WHERE a>1000;

2 • 300 • NULL • No. rows • Error is thrown

Q3 Which of the following statements are FALSE about ORDER BY?

4

• ORDER BY can be used with DATE attributes.
5 • ORDER BY default sort is ascending order.
• WHERE Clause can be used after ORDER BY clause
8 • ORDER BY only works on Number columns

Q Consider the table Employee (Ename, Salary, Deptno, Job) and the query,
SELECT COUNT(*) FROM employee ORDER BY Salary WHERE Dept no = 10;

Choose the correct option.

- The above query will lead to error because COUNT(*) is a wrong usage.
- The above query will lead to error because ORDER BY clause should be used after WHERE clause.
- The above query will display the total number of employees belonging to Dept 10 in the ascending order of the Salary.

Sample input for employee table & query output is provided below.

I/p table

Ename	Salary
Ethan McCarthy	70000
Jack Abraham	40000
James Potter	50000

O/p table

Ename	Salary
Jack Abraham	40000
James Potter	50000
Ethan McCarthy	70000

Which of the following query will produce the o/p ?

- SELECT Ename, Salary, FROM employee ORDERBY Ename, DESC, Salary;
- SELECT Ename, Salary FROM employee ORDER BY Ename, Salary;
- SELECT Ename, Salary FROM employee ORDER BY Ename DESC, Salary DESC;
- SELECT Ename, Salary FROM employee ORDER BY Salary, Ename;

Q Consider the following "Quotation" table. How many rows would be displayed when the following query is executed ?

SELECT Item Code, Count(*) FROM Quotation GROUP BY Supplier Id, Item Code.

Quotation Id	Supplier ID	Item Code	Quoted Price
Q1001	S1001	I1012	1500
Q1002	S1002	I1012	1400
Q1003	S1003	I1013	1450
Q1004	S1001	I1012	600
Q1005	S1004	I1013	625

4

Q Given the employee table as input, choose the query that displays department wise average salaries sorted in descending order.

ID	Ename	Dept	Salary
1	James Potter	FSI	78000
2	Ethan McCoy	ETA	90000
3	Emily Rayner	ETA	25000
4	Jack Abraham	ETA	30000

- SELECT AVG(salary) AvgSal, Dept FROM Employee GROUP BY Dept ORDER BY 1;
- SELECT AVG(salary), ~~Dept~~ AvgSal, Dept FROM Employee ORDER BY AVG(salary) GROUP BY Dept;
- ✓ SELECT AVG(salary) AvgSal, Dept FROM Employee GROUP BY Dept ORDER BY AvgSal DESC
- SELECT Dept, AVG(salary) AvgSal, FROM Employee GROUP BY 1 ORDER BY 2 DESC

Q# Given the Employee table as input, which of the following query displays departments that have more than 1 employees?

ID	Name	Dept
1	James Potter	FSI
2	Ethan McCarty	ETA
3	Emily Rayner	ETA

- SELECT Dept, COUNT(*) FROM Employee GROUP BY Dept WHERE COUNT(*) > 1
- SELECT Dept, COUNT(*) FROM Employee GROUP BY Dept, COUNT(*) HAVING COUNT(*) > 1
- ✓ SELECT Dept, COUNT(*) FROM Employee GROUP BY Dept, HAVING COUNT(*) > 1
- SELECT Dept, COUNT(*). FROM Employee GROUP BY COUNT(*) HAVING COUNT(Dept) > 1

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Q Which statement shows the Department ID, minimum salary, & maximum salary paid in the department only if the minimum salary is less than 5000 & maximum salary is more than 15000 ?

⇒
```
SELECT Dept Id, MIN(salary), MAX(salary)
FROM EMPLOYee
GROUP BY Dept Id
HAVING MIN(salary) < 5000
AND MAX(salary) > 15000;
```

Q Consider the following record of Customer table:

First Name	Last Name
Martin	Grove

What will be the O/P of the below query ?

```
SELECT CONCAT(LOWER(Last Name), UPPER
(First Name)) "FULLNAME" FROM Customer.
```

01 Sunday
- martin GROVE
- GROVEmartin
- groveMARTIN
- martin, GROVE

2020

CASE Statements :- If else for db.

Ex :- Let's suppose we need to calculate the profit of all movies from the movies table and we need to check the movies are hit or superhit, flop or average.

```
SELECT title, (worldwide_gross - budget) profit,
CASE
    WHEN (worldwide_gross - budget) > 1000000000 THEN
    "SUPER HIT"
    WHEN (worldwide_gross - budget) > 250000000 AND
    (worldwide_gross - budget) < 1000000000 THEN
    "HIT"
    WHEN (worldwide_gross - budget) > 0 AND
    (worldwide_gross - budget) < 250000000 THEN
    "AVERAGE"
    ELSE "FLOP"
END AS verdict
FROM MOVIES
```

This will generate 3 columns, profit, title & verdict. Verdict will show the movies which is flop, Average, Hit & Superhit.

03

(063-303) WK 10

MARCH • Tuesday

FEBRUARY 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

JOINS :-

- Cartesian Product / CROSS JOIN
- Inner Join
- Outer Joins
  - → left outer
  - → Right outer
  - → Full outer
- Self Joins

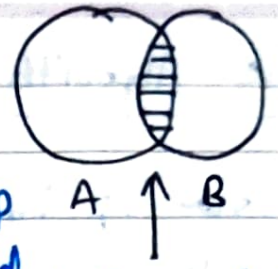
Let's suppose we have a database named Whatsapp and inside whatsapp db we have 3 tables users, groups and membership. In users we have 6 rows & in groups we have 4 rows. & in membership we have 10 rows. Users & groups have many to many relationship, A group can have multiple users. A user can be a part of multiple groups. & If we have many to many relationship then we need to make 3 tables. Membership table will show the relationship b/w users & group. If we apply cartesian product on users & groups then we will have 24 rows in the resultant ~~group~~ table.

```
Select * FROM users CROSS JOIN groups ;
```

2020

CARTESIAN Product/CROSS JOIN is not much useful.

INNER JOINS:-



SELECT * FROM membership m JOIN users u ON m.uid = u.id

The common uid of user table & membership table will be displayed using INNER JOIN.

SELECT ID, E.NAME, E.COMPID AS ECID, C.COMPID AS CCID, MODEL FROM EMPLOYEE E INNER JOIN Computer C ON E.COMPID = C.COMPID

Employee table (ALIAS E)

ID	Ename	Dept	Compid
1	James Potter	IEP	1001
2	Ethan McCarty	ETA	NULL
3	Emily Rayner	ETA	1002

Computer table (ALIAS C)

CompID	Make	Model	MYear
1001	Dell	Vostro	2013
1002	Dell	Precision	2014

FEBRUARY 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

Illustrative Algorithm :-

for each row r1 in Employee table
  for each row r2 in computer table
    if r1.compId == r2.compid
      add combined row to result.

Result :-

ID	ENAME	ECID	CCID	Model
1	James Potter	1001	1001	Vostro
3	Emily Rayner	1002	1002	Precision

CARTESIAN PRODUCT / CROSS JOIN :-

```
SELECT * FROM users CROSS JOIN groups;
```

LEFT JOIN outer :- (LEFT JOIN)

9
SELECT * FROM membership m LEFT OUTER
10 JOIN users u ON m.uid = u.id

11 This will give us the o/p as all the data
from membership table & the data which
12 is matching in users table with membership
table.

1
SELECT ID, ENAME, E.COMPID AS ECID,
2 C.COMPID AS CCID, MODEL FROM Employee E
LEFT OUTER JOIN Computer C ON E.COMPID=
3 C.COMPID

4 Employee table (Alias E)

ID	ENAME	DEPT	COMPID
5 1	James Potter	ICP	1001
2	Ethan McCarty	ETA	NULL
6 3	Emily Rayner	ETA	NULL

07 Sunday
Computer Table (Alias C)

CompID	MAKE	MODEL	MYEAR
1001	DELL	Vostro	2013
1002	Dell	Precision	2014

Illustrative Algorithm :-

```
for each row r1 in Employee table
    set matched_in_comp to false
    for each row r2 in Computer table
        if r1.COMPID == r2.COMPID
            add combined row to Result
            set matched_in comp to true
    if matched_in_comp is false
        add Employee row to result.
```

Result :-

ID	ENAME	ECID	CCID	MODEL
1	James Potter	1001	1001	Vostro
2	Ethan McCarty	NULL	NULL	NULL
3	Emily Rayner	NULL	NULL	NULL

Right Outer JOin :- Right JOIN

```
SELECT * FROM membership m FULL OUTER
JOIN users u ON m.uid = u.id
```

This will give the output as all the data from users table & the data which is matching in users table with membership table.

UNION :-

```
SELECT id FROM users
UNION
SELECT gid FROM groups
```

- No. of columns of id & gid should be same.
- The data type of both the id & gid should be same.
- UNION will always give the unique results.

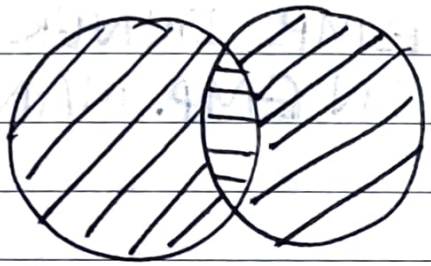
UNION ALL :-

```
SELECT id FROM users
UNION ALL
SELECT gid FROM groups
```

UNION will repeat the records. It will not give the records like UNIQUE. Rest will all same as UNION

FULL JOIN :- RIGHT & LEFT JOIN

```
SELECT * FROM membership m LEFT OUTER
JOIN users u ON m.uid = u.id
UNION
SELECT * FROM membership m RIGHT OUTER
JOIN users u ON m.uid = u.id
```



Q We have 3 tables membership, groups & users & we need name of the person associated with ~~how many~~ which no. of groups?

```
SELECT name, gname FROM membership m
JOIN users u ON m.uid = u.id JOIN
groups g ON m.gid = g.gid ;
```

FEBRUARY 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

SELF JOIN :- Joining a table with itself.

```
SELECT u1.name, u2.name FROM users
U1 JOIN users U2 ON U1.emergency-
contact = u2.id;

SELECT Emp.ID EID, EMP.ENAME EMPNAME,
MGR.ID MID, MGR.ENAME MGRNAME
FROM EMPLOYEE EMP INNER JOIN
EMPLOYEE MGR ON EMP.MANAGER =
MGR.ID
```

Employee Table :- (ALIAS EMP)

ID	Ename	Salary	Dept	Manager
1	James Potter	75000	ICP	3
2	Ethan McCarty	90000	ETA	NULL
3	Emily Rayner	25000	ETA	2

Employee Table :- (ALIAS MGR)

ID	ENAME	SALARY	DEPT	MANAGER
1	James Potter	75000	Icp	3
2	Ethan McCarty	90000	ETA	NULL
3	Emily Rayner	25000	ETA	2

15 Sunday

Illustrative Algorithm :-

```
for each row r1 in Employee (EMP)
    for each row r2 in Employee (MGR)
        if r1.MANAGER = r2.ID
            add combined row to Result
```

Result :-

EID	EMP NAME	MID	MGR NAME
1	James Potter	3	Emily Rayner
3	Emily Rayner	2	Ethan McCarty

17

(077-289) WK 12

MARCH • Tuesday

M T W T F S S
3 4 5 6 7 1 2
10 11 12 13 14 9 9
17 18 19 20 21 15 16
24 25 26 27 28 22 23
29

Subquery :- Query inside query. The query which is inside will be solved first then the outer query will get solved.

Independent Subquery :- Query which is inside has nothing to do with the query outside in terms of execution. Both queries will be independent to each other. There is no colliding factor w.r.t execution.

Correlated query :- Query which is inside and the query which is outside are correlated to each other

Independent Subquery Examples :-

Q Find movies with maximum budget?

```
SELECT * FROM movies WHERE budget =
(SELECT MAX(budget) FROM movies)
```

The query which is inside will give a single result after execution became = is used here

2020

Q Write a query to show the columns of those movies whose actor name starts with letter "A".

SELECT * FROM movies WHERE actor IN (SELECT DISTINCT(actor) FROM movies WHERE actor LIKE "A%")

The query which is inside will give a multiple results because IN is used here

Q Write a query to show all the movies of top 5 actors who has earned maximum profit.

SELECT * FROM movies WHERE actor IN (SELECT actor FROM (SELECT actor, SUM(worldwide_gross-budget) AS profit FROM movies GROUP BY actor ORDER BY profit DESC LIMIT 5) A)

→ When we use FROM in the ~~third~~ second subquery then we will have to use reference name or alias for table so we have used A as reference & WHEN we use WHERE in subquery like first query then we don't need any reference because where doesn't require any reference.

2020

FEBRUARY 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

Correlated query example :-

Q Find top movie of each genre.

```
SELECT title, genre, (worldwide_gross - budget)
As profit FROM movies m1 WHERE
(worldwide_gross - budget) = (SELECT MAX(
worldwide_gross - budget) FROM movies m2
WHERE m2.genre = m1.genre
```