MySQL | Complete SQL Course | for Data Science

7:13 pm

Data - Raw facts & figures.

Requirement of Data
1. Integrity - Accurate & Consistency
2. Availability
3. Security
4. Independent of Application
5. Concurrency - Multiple Access.

Limitations of flat files
1. Dependency of program on physical structure of data
2. Complex process to retrieve data
3. Loss of data on concurrent access.
4. Inability to give access based on roles (security)
5. Data Redundancy.

A database is a shared collection of logically related data & description of these data, designed to meet the information needs of an organisation.

An application program interacts with a database by issuing an appropriate request (typically a SQL statement)

Types of DBMS
1. Hierarchical - Tree Structure.
2. NoSQL → Key-value pairs - MongoDB
3. Network - Graph Structure (fb) Neo4j
4. Relational - In form of tables. - MySQL, MS Access, PostGre

Row / Records / tuples
Columns / Fields

Domain ⇒ Size, constraints
Integrity

No. of Rows / tuples : Cardinality of the relation
" " fields : Degree of a relation

1. Entity Integrity : Primary Key
2. Domain Integrity : Data Types, Check Constraints
3. Referential Integrity : Foreign Key

DATABASE KEYS.

- Super Key
- Candidate Key
- Primary Key
- Alternate Key

→ Khud se banaya kyunki koi key nahi hain

- Foreign Key
- Composite Key
- Compound Key
- Surrogate Key

Keys { emp_id, aadhar, email }

→ combination of keys

→ Must have atleast one foreign key.

→ Superset / powerset of all Keys - super key

Superkey → { emp_id, aadhar, email, emp_id + aadhar, emp_id + email, aadhar + email, aadhar + email + emp_id }

Min. no. of ~~Key~~ columns to become key - candidate key

Minimal ~~set~~ ~~form~~ superkey.

Primary key : Selected key to fit as a primary key

Candidate Keys - Primary Key = Alternate Key

Here : P.Key → emp_id

Good to have criteria * Numerical
* As small as possible.

Assignment

department 1 : m Instructor

Instructor can take multiple courses. m : n

There are many instructor for a given course

Primary Keys : DID, InstrID, Cid

foreign Keys : RoomID, DeptID, InstrID

(2) Primary Keys : CustID, email, AccType, Tid

foreign Keys : AccType, AccNum, CustID

E-R Model. Entity Relationship Model.

Crow - Foot Notation

|| — Exactly one

|O — zero or one

>O — zero, one or more

>| — One or more

SQL Languages

DDL : Create

Alter

Drop - Delete existing database objects

Truncate - Remove all rows from table.

DCL : Grant, Revoke

DML - Insert, Update, Delete, Select

TCL - Commit, Rollback

-- This is a comment,

/* This is
a comment */

Lucidchart
for ER Diagrams.

```
CREATE DATABASE IF NOT EXISTS MyDB
DROP DATABASE mydb.

CREATE TABLE IF NOT EXISTS users {
    id integer,
    name varchar(255),
    email varchar(255),
    password varchar(255)
};
DROP TABLE users.
```

Empty the table
TRUNCATE TABLE users

Constraints ① NOT NULL

```
Create table person {
    id integer NOT NULL,
    name varchar(255)
}
```

② UNIQUE

```
CREATE TABLE users {
    id integer NOT NULL,
    name varchar(255),
    email varchar(255) UNIQUE
};
```

```
Create table users {
    name varchar(255),
    email varchar(255),
    UNIQUE(email, name)
```

→ OR CONSTRAINT u_email UNIQUE(email

Combination of name & email should be unique.

③ PRIMARY KEY.

```
Create table users {
    id integer PRIMARY KEY
}
```

```
Create table users {
    id integer
    PRIMARY KEY(id)
```

(4) FOREIGN KEY.

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

(5) CHECK.

```
Create table students {
sid integer PRIMARY KEY,
sname ~~VARCHAR~~ varchar(255) NOT NULL,
email varchar(255) NOT NULL UNIQUE,
age integer CHECK (age >6 AND age <25)
```

(6) DEFAULT.

```
create table passengers {
pid integer PRIMARY KEY
pname varchar(255) NOT NULL
gender varchar(255) DEFAULT "others"

journey_date datetime DEFAULT CURRENT_TIMESTAMP
```

(7) AUTO INCREMENT.

```
create table passengers {
id integer PRIMARY KEY AUTO_INCREMENT
pname varchar(255) NOT NULL;
}
```

ALTER TABLE

- add cols
- delete cols
- modify cols
- add/remove constraints

DDL Command.

ALTER TABLE students ADD COLUMN college VARCHAR(255)

ALTER TABLE students DROP COLUMN age;

ALTER TABLE students MODIFY COLUMN sname integer

ALTER TABLE students DROP CONSTRAINT u_email

DML

(1) INSERT.

INSERT INTO STUDENTS VALUES (6, "abc", "ab@gmail.com", "123")
(7, "ab", "a@gmail.com", "456");

(2) RETRIEVE.

SELECT * from train

SELECT Name, Sex, Survived FROM train

SELECT Name As P_Name, Sex AS Gender from train

(Titanic Dataset)

Expression : SELECT Name, Sibsp + Parch As FAMILY FROM train

SELECT Name, Age + 102 As CurrentAge FROM train

Constant : SELECT Name, 100000 As COMPENSATION FROM train

Distinct Single Col : SELECT DISTINCT Embarked FROM TRAIN

SELECT DISTINCT Pclass FROM train

SELECT DISTINCT Pclass, Embarked FROM train (CROSS PRODUCT)

Comparison operator

SELECT * FROM Train WHERE Survived = 0

SELECT * FROM Train WHERE PClass = 3

AND Operator : SELECT* FROM Train WHERE PClass = 3 AND SEX = "M"

SELECT* from Train WHERE Age BETWEEN 10 AND 15 (10 & 15 Inclusive)

ORDER of Query Execution

FJWGHSDO

1 From 2 Join 3 Where 4 GroupBy 5 Having 6 Select 7 Distinct 8 OrderBy

Frank John's Wicked Grave Haunts Reveal Dull Owls

IN/NOT IN

SELECT title FROM movies WHERE genre LIKE "Comedy" OR genre LIKE "Action"

SELECT title, genre from movies WHERE genre IN ("Action", "Horror", "Drama")

SELECT title from movies WHERE genre NOT IN ("Action", "Horror", "Drama")

Wildcards: Used for String searching.

1. %

SELECT title FROM movies WHERE title LIKE "A%" (where title starts with A)

SELECT title from movies where title LIKE "%man%" (The movies title containing 'man' string are displayed)

SELECT title from movies where title LIKE "%a" (ends with a)

SELECT title FROM movies WHERE actor LIKE "%Khan%" OR actor LIKE "%Kapoor%"

2. _ Wildcard

SELECT title FROM movies WHERE title LIKE "A____" (Movies with 5 letters starting with A)

UPDATE

UPDATE passenger SET name = "Rahul" (every name is updated with Rahul.

UPDATE ~~name~~ passenger SET name = "Ankit" email = "ab@gmail.com"
Where email LIKE "%yahoo%"

DELETE Query

DELETE FROM passenger WHERE id = 1

DELETE FROM movies Where id >> AND email LIKE "%yahoo%"

DELETE FROM passenger WHERE 1 (equivalent to truncate).

functions, sorting data, Grouping data, having

① functions : abs, round, ceil, floor, upper/lower, concat, length, substr(string, start(1), length)

Aggregate fn : min/max/sum/avg
count/count with distinct

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

SELECT title, ROUND((runtime/60)) AS hours FROM movies

ROUND ((runtime/60), 2))
↳ Round off to 2 decimal places

CEIL(runtime/60) ↑
FLOOR(runtime/60) ↓

SELECT UPPER(title) FROM movies / LOWER(title)

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

Length | SELECT title, LENGTH(title) as length FROM movies.

SUBSTR | SELECT title, SUBSTR(title, 1, 5) as short FROM movies.
" " (title, 3, 6) as short2 " "

Aggregate fns | select MAX(BUDGET) from movies.
Select MIN(budget) from movies
Select SUM(india_gross) from movies
Select AVG(india_gross) from movies.

Select Count (*) FROM movies

Select ~~Cou~~ ~~actor~~, COUNT(DISTINCT (actor)) as movie_count
FROM movies

We have total unique actors.

Sorting Data

SELECT title, (ww_gross - budget) AS profit
FROM movies ORDER BY profit DESC LIMIT 5
(Print top 5 profitable movies)

SELECT * FROM movies ORDER BY genre, title DESC
↳ Ascending

Grouping Data.

(1) Top 5 actors who have max. no. of movies.

Select actor, COUNT(*) AS num_movies FROM movies
GROUP BY actor
ORDER BY num_movies DES
LIMIT 5

(2) Which genre of movies have made max profit

SUM ~~(count~~ (ww_gross - budget)
total

Select genre, ~~Count~~ (*) As ~~total~~ profit FROM movies
~~(w~~
GROUP BY Genre
ORDER BY ~~total~~ profit DESC
LIMIT 5

Select genre, SUM (ww_gross - budget) as total_profit
FROM movies GROUP BY Genre ORDER BY total_profit
DESC

Select director, AVG(~~SUM~~ ~~COUNT~~ (ww_gross - budget)) as ~~total~~ avg profit

from movies group by ~~total~~ directors ~~and~~

ORDER by ~~total~~ avg profit DESC LIMIT 5

(4) Movie with max. budget

SELECT title, budget FROM movies GROUP BY title

ORDER BY budget DESC / asc (for min)

(5) Which Actor, Director combo. ~~have~~ made max. profit till now? [5]

Select actor, director, SUM(ww_gross - budget) AS profit ~~FROM~~

FROM Movies GROUP BY actor, director ORDER BY profit

DESC LIMIT 5

(6) Top 5 actors with highest profit

Select actor, SUM(ww_gross - budget) As Profit

FROM movies group by actor order by profit desc

limit 5

HAVING - filters on GroupBy.

(avg. of)

All Actors whose movies were released on more than 1000 screens

SELECT actor, AVG(screens) As openings FROM movies

GROUP BY actor HAVING openings > 1000

ORDER BY ~~actor~~ openings DESC

Q. Which of the following aggregate functions can be applied on both numbers & characters typed columns

MIN, MAX, COUNT

SELECT AVG(SALARY) as Avgsal, Dept from employee GROUP BY Dept ORDER BY Avgsal Desc

SELECT Dept, COUNT(*) from employee GROUP BY Dept HAVING COUNT(*) > 1

select DeptId, MIN(salary), MAX(salary)
FROM employee GROUP BY DeptId
HAVING MIN(salary) < 5000
AND MAX(salary) > 15000

CASE (if else for Database)

Select title, (ww_gross - budget) as profit,
CASE
WHEN profit > 100000000 THEN "SUPER HIT"
WHEN profit > 25 cr AND profit < 100 cr
THEN "HIT"
WHEN profit > 0 AND profit < 25 cr THEN 'AVERAGE'
ELSE 'FLOP'
END AS verdict → Name of column when is equivalent to if
FROM movies

JOINS

1. Cartesian Product

Select * from Users (t1) CROSS JOIN groups (t2)

INNER join → default for Inner Join

Select * FROM membership m JOIN users u ON
m.mid = u.uid

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

Select ID, EName, E.compId as ECid, C.compId as CCID, Model
FROM employee E left outer join Computer C on E.compId
= C.CompId

Illustrative Algo

for each row r1 in Employee Table
set matched 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.

A B Right outer join

↳ full outer join

full outer join works through UNION.

SELECT id FROM users
UNION
SELECT gid FROM groups.

both should have same datatype

UNION ALL → allows duplicates

full outer 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

Multiple join

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

Self Join

```
Select u1.name, u2.name from users u1 JOIN users u2
on u1.emergency_contact = u2.id
```

Subquery / Nested Query

if{ if{{{

* Independent
* Correlated

(1) ↳ find movie with max budget

```
Select * from movies where budget = (select MAX(budget)
                                     from MOVIES)
```

(2) Movies with Actor starting with name A

```
Select * from movies where actor IN
    (select actor from movies WHERE actor LIKE 'A
        ↳ DISTINCT
```

(3) find movies with top actors.

```
SELECT * FROM movies WHERE actor IN
(Select actor FROM (select actors, SUM(ww gross - budget) as
                    profit from movies group by actor
                    ORDER by profit DESC) as A )
                                    ↳ this table
```

from me reference diya padhta hain, where me nahi.

CORRELATED Query

find top movie of each genre.

```
Select title, genre, (ww gross - budget) As profit WHERE
(profit = (select MAX (ww gross - budget) FROM movies m2
 where m2.genre = m1.genre)
                                                  from movies m1
```

Equivalence schedules

1. Result equivalence. - If 2 schedules produce same result after execution
2. View equivalence - If transactions in both schedules perform similar action in similar manner.
3. Conflict equivalence -
   - * Both belong to separate transaction.
   - * Both access same data item
   - * Atleast one of them is 'write' operation

   → Two schedules would be conflicting if they have these properties.

They are said to be conflict equivalent iff:

- Both the schedules contain same set of transaction.
- The order of conflicting pair of operations is maintained in both the schedules.

States of Transactions :-