DATABASE MANAGEMENT SYSTEMS

2 important decisions:

1. Storing the data

2. Querying the data

ENTITIES: About whom are we storing data?

ATTRIBUTES/ FIELDS: What data about entities are you storing?

Ex: Course\_enrollment – entity

Course\_ID, Roll\_no – attributes

Suppose the 3 datasets our university has: Students, Employees, Applicants.

Also, we will be maintaining data under university also under students.

Data at one particular location has some reference to other entity. -> Relation between data.

**Database**- It is storage for any kind of data/information. Ex: Phone directory, Account Book etc

**Relational Database-** Store data/information with predefined structure and relations between data. In this data is stored in form of tables, which is further the combination of column and row matrix. DB structure is designed on Normalization Principle.

Relational database, as the name says has relations between objects/tables (usually defined by E-R diagram) whereas plain database just consists of objects/files with no relations defined between them.

Relational database is an extension of a plain database.

**DBMS:**

* A DBMS is a storage area that persist the data in files.
* There are limitations to store records in a single database file.
* DBMS allows the relations to be established between 2 files.
* Data is stored in flat files with metadata.
* DBMS does not support client / server architecture.
* DBMS does not follow normalization. Only single user can access the data.
* DBMS does not impose integrity constraints.
* ACID properties of database must be implemented by the user or the developer

**RDBMS:**

* RDBMS stores the data in tabular form.
* It has additional condition for supporting tabular structure or data that enforces relationships among tables.
* RDBMS supports client/server architecture.
* RDBMS follows normalization.
* RDBMS allows simultaneous access of users to data tables.
* RDBMS imposes integrity constraints.
* ACID properties of the database are defined in the integrity constraints.

**DBMS:** is a software system that allows Defining, Creation, Querying, Update, and Administration of data stored in data files.

*Features:*

* Normal book keeping system, Flat files, MS Excel, FoxPRO, XML, etc.
* Less or No provision for: Constraints, Security, ACID rules, users, etc.

**RDBMS:** is a DBMS that is based on Relational model that stores data in tabular form.

* SQL Server, Sybase, Oracle, MySQL, IBM DB2, MS Access, etc.

*Features:*

* Database, with Tables having relations maintained by FK
* DDL, DML
* Data Integrity & ACID rules
* Multiple User Access
* Backup & Restore
* Database Administration

SQL – Structured Query Language --- Language to query upon the data that we have stored.

Query – Statement that we execute to fetch the data from a RDBMS

SELECT:

|  |  |  |
| --- | --- | --- |
| ID | First\_Name | Last\_Name |
| 1 | A | B |
| 2 | X | Y |

The very first important function/feature is the SELECT statement.

Ex:

STUDENT TABLE

To retrieve all names ->*Query:* SELECT First\_Name, Last\_Name from student;

To retrieve in specific ->*Query:* SELECT First\_Name, Last\_Name from student where ID = 1;

SUBQUERY/NESTED STATEMENTS:

|  |  |  |
| --- | --- | --- |
| id | Name | Batch |
| 1 | Vijay | 2021 |
| 2 | Gayu | 2021 |

|  |  |  |
| --- | --- | --- |
| id | Name | Batch |
| 1 | Vijay | 2021 |

<- Students <- Failed\_students

We need to give degree who is in batch of 2021 and not in Failed\_students.

*Query*: SELECT id from Students where batch=2020 and id NOT IN (SELECT id from Failed\_students);

Data type of outside statement should be equivalent to the result’s type in the nested statement.

* 2 types of Nested Statements: 1. CORRELATED – Subquery is not independent.

2. NON-CORRELATED – Main and subquery are both independent

=> Above example is a non-correlated nested statement.

Relational table is a set

SQL CLAUSES: Clause’s Actual Meaning: Those are the things we need to check/ensure

* *where - filter records based upon some conditions*
* *group by - associated with aggregate functions*
  + - *AGGREGATE FUNCTION – compute on a large amt. of data and return a result. Ex: sum, avg, min, max*
    - *WHICH ARE THE PEOPLE WHO DON’T HAVE COUNTRY AS INDIA??*
    - *Query*: Select count(student\_id), country from student where country != ’IN’ group by country;
    - Result: 2 groups one of usa and uk. Final output: number of students from each country where country != INDIA
* *having - filter on results returned by aggregate functions*
  + - *Ex: having count(id) > 1 [ number of students where count of students>1 ]*
* *order by - sorting*

**STUDENT Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Batch | Country |
| 1 | A | 2020 | IN |
| 2 | B | 2021 | USA |
| 3 | C | 2020 | USA |
| 8 | D | 2021 | UK |
| 9 | E | 2021 | IN |
| 11 | F | 2020 | IN |
| 12 | G | 2022 | IN |

*Query:* SELECT \* from student where id>10 and batch=2020;

**WHAT\_TO\_EAT Table**

|  |  |  |
| --- | --- | --- |
| Food\_id | Food\_name | Food\_type |
| 201 | Pastry | 3 |
| 101 | Rice | 1 |
| 120 | Meal | 2 |
| 302 | XYZ | 1 |

* + Rice first and desert in the end. We need to return in the order of what type of food they are. So we need order the food\_id based on food\_type

*Query:* SELECT food\_id from what\_to\_eat ORDER BY food\_type

UNION, MINUS, INTERACT: (more of like sets, not tables)

A = {1,2,3} B = {1,2,4,5}

* UNION – returns all items only once. A U B => {1,2,3,4,5}
* UNION ALL – keeps all the duplicates => {1,1,2,2,3,4,5}
* MINUS – [set A – set B] EX: Instead of NOT IN we can use MINUS.
* INTERSECT – Value present in common i.e., both in set A and set B

Statements are between 2 SELECT statements.

Example=> *Query*: Select name from student MINUS Select name from failed\_students

Select name from students UNION Select name from employees

Select name from students INTERSECT Select name from delhi\_students

|  |  |
| --- | --- |
| Id | Salary |
| 1 | 10000 |
| 2 | 8000 |
| 3 | 900 |

JOINS:

employee employee\_salaries

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Fname | Lname | Role |
| 1 | A | B | CEO |
| 2 | A | C | VP |
| 5 | A | D | VP |

We need to return 1 A B CEO 10000 and 2 A C VP 8000.

We will join the tables by id.

* Inner Join – *Query:* Select \* from table\_A JOIN table\_B;
* Left Join – Select everything from left side and if corresponding values are not there on the right side fill those values with NULL.

We will get 5 A D VP NULL

*Query:* Select \* from table\_A LEFT JOIN table\_B;

* Right Join – Select everything from right side and if corresponding values are not there on the left side fill those values with NULL.

We will return 3 NULL NULL NULL 900

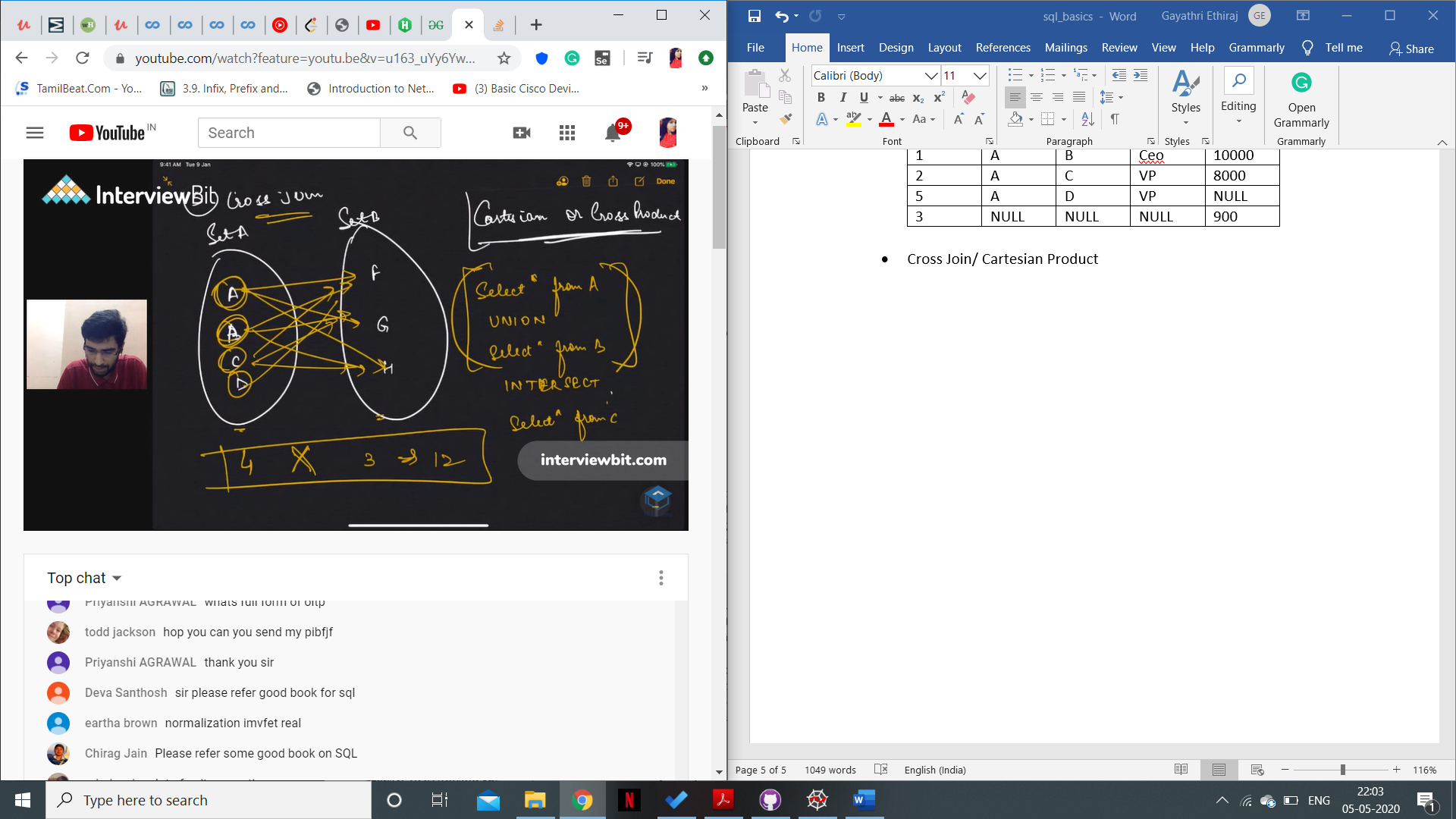
* Full Join – Select everything. Join if a match. Else fill left side or right side by NULL.

Union of left and right join.

We will return

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | A | B | Ceo | 10000 |
| 2 | A | C | VP | 8000 |
| 5 | A | D | VP | NULL |
| 3 | NULL | NULL | NULL | 900 |

* Cross Join/ Cartesian Product



Queries should run fast. If we want to speed up our queries, INDEXING.

|  |  |  |
| --- | --- | --- |
| Id | Name | Score |
| 1 | A | 60 |
| 2 | B | 10 |
| 3 | C | 90 |
| 4 | D | 80 |
| 5 | E | 5 |

Index: It is a data structure(BTree, B+Trees)

Provides quick lookup based upon queries in a column or a set of columns.

CREATE INDEX Name\_score ON students(Name,Score)

Disadvantage: It needs extra space. It can slow down writes.

* Unique indexing: Ensures that no 2 students have the same name
  + unique index on name column of students.
* Non-unique indexing: -> Speed up the queries

CLUSTERED VS. NON-CLUSTERED INDEXES:

Queries fetch student always in order of roll numbers.

If I order the roll numbers in my memory, it’ll speed up my queries.

CLUSTERED INDEXES: They change the way data is stored in the database as per the index.

NON-CLUSTERED INDEXES: Maintain a separate Data Structure to optimize queries.

* + 1. There can be only one clustered index. There can be multiple non-clustered indexes.
    2. Clustered Index affects how data is stored in memory while non-clustered don’t.

Clustered is directly affecting data, non-clustered doesn’t directly affect.

* + 1. Clustered indexes have faster performance w.r.t non-clustered.

VIEWS:

students female students

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Batch | Gender |
| 1 | A | 2020 | M |
| 2 | B | 2020 | F |
| 3 | C | 2018 | F |
| 4 | D | 2019 | M |

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Batch | Gender |
| 2 | B | 2020 | F |
| 3 | C | 2018 | F |

Suppose a new student gets added who is a female.

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | E | 2019 | F |

Now this data should not only be added to students table. It should be added to female students table as well. If by mistake we forget to add the record to both the tables, operations dependent on the female\_students data would go wrong.

Hence, we would give the view of students to the female\_students table.( a view/subset of students data where gender = female).

A view is nothing but a SELECT statement that gives an intuition of a different table while in reality, data is present at only on location. It’s like a camera viewing the database but with the filter.

If u add something to students, it’ll be updated to view also.

WHERE clause pattern matching

* 1. Wild Card Patterns:

Q) Find all students whose name begins with ‘kh’

-> *Query:* Select \* from student where name LIKE ‘kh%’;

Q) Find all students that have ‘ama’ in their name

-> *Query:* Select \* from student where name LIKE ‘%ama%’;

***% - any sequence of 0 or more characters***.

Q) Find all students that do not have ‘ama’ in their name

-> *Query:* Select \* from students where name NOT LIKE ‘%ama%’;

2. Wildcard for specific position( \_ ):

-> *Query:*  Select \* from students where name LIKE ‘\_ \_my’;

Q) Return all students whose name has more than 4 characters

-> *Query:* Select \* from students where name LIKE ‘\_ \_ \_ \_ %’

1. OLTP Database - Transactional

2.OLAP Database - Analytics

|  |  |
| --- | --- |
| OLTP – Transactional  Online Transactional Processing | OLAP – Analytics  Online Analytics Processing |
| A large number of queries but each query is very small. (insert a value, update a value, ..) | Useful for Business Analytics & Intelligence |
| Fast response time | Smaller number of queries, but each query can be quite complex |
| Distributed | Visualization – common feature |
| Ex: Updating who liked a post in FaceBook.  Serve a lot of parallel requests. | Ex: Students who attended more than 5 classes/ who solved more than X contests. |

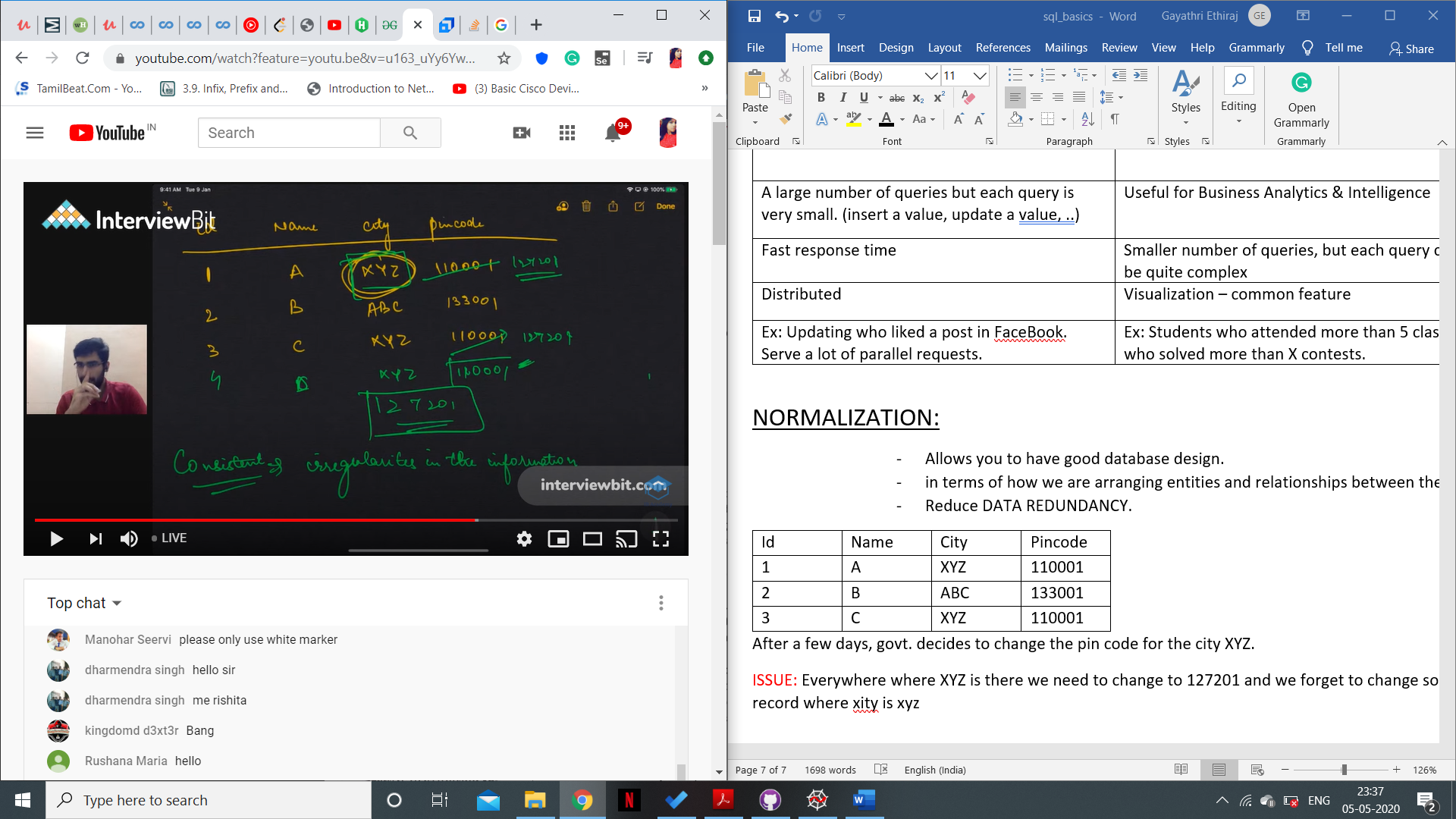
NORMALIZATION:

* + - Allows you to have good database design.
    - in terms of how we are arranging entities and relationships between them.
    - Reduce DATA REDUNDANCY.

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | City | Pincode |
| 1 | A | XYZ | 110001 |
| 2 | B | ABC | 133001 |
| 3 | C | XYZ | 110001 |

Student table ------- After a few days, govt. decides to change the pin code for the city XYZ.

ISSUE: Everywhere where XYZ is there we need to change to 127201 and we forget to change some record where city is XYZ -> IRREGULARITIES IN THE INFORMATION. CONSISTENCY ISSUES.



|  |  |  |
| --- | --- | --- |
| Id | Name | City id |
| 1 | A | 1 |
| 2 | B | 2 |
| 3 | C | 1 |
| 4 | D | 1 |

|  |  |  |
| --- | --- | --- |
| City id | Name | Pincode |
| 1 | XYZ | 110001 |
| 2 | ABC | 133001 |

student table city table

So, if we want to make changes, it’s sufficient if we change it in the CITY table ALONE.

I can change XYZ to new pin code 127201 easily.

1 NF

1 NF : There are no multivalued attributes.

|  |  |
| --- | --- |
| Stud\_id | Book\_id |
| 1 | 101 |
| 1 | 120 |
| 1 | 113 |
| 1 | 121 |
| 1 | 180 |
| 2 | 11 |
| 2 | 113 |
| 2 | 21 |

Allocation table

|  |  |
| --- | --- |
| Stud\_id | Book\_id |
| 1 | [101,120,113,121,180] |
| 2 | [11,113,21] |

Rows are having multi value attribute

So, we split it like this:

2 NF:

PRIMARY KEY: attribute that has to be unique for any record in the table

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Gender | Salutation |
| 1 | A | M | Mr |
| 2 | B | F | Ms |
| 3 | A | M | Mr |

If we decide to change Mr -> Master. Iterate throughout the column and update the value

If gender and salutation are the 2 columns that can be uniquely identified by something other than the primary key => we split these set of columns to a different table

|  |  |  |
| --- | --- | --- |
| Id | Name | Gender\_id |
| 1 | A | 1 |
| 2 | B | 2 |
| 3 | A | 1 |

|  |  |  |
| --- | --- | --- |
| Gender\_id | Gender | Salutation |
| 1 | M | Mr. |
| 2 | F | Ms. |

For any changes, it is sufficient if we make changes to the gender table alone.

SQL Statement Types:

ex: How we build a house?

1. First decide on how my house should look like. DEFINITION
2. Who will live in the house? QUERY
3. You change/modify the design of the house. MANIPULATING
4. Decide who will control/manage it. CONTROL

DATA DEFINITION LANGUAGE: create, drop

QUERY: Select

DATA MANIPULATION LANGUAGE: insert, update

CONTROL: Grant, revoke(read writes, or write writes)

PROCEDURES:

Each procedure is a SELECT statement identified by the name or these can be modification of students.

change salutation (gender, new salutation)

Q) Create a new table with exact data structure of another table

Students table (id, name, ...) We know only table’s name

We need to create students’ copy, without knowing attributes how do we do that.

I WANT ONLY THE STRUCTURE OF THE TABLE

*Query:* Select \* into student\_copy from students where (3=4);

*I need to put something in where clause is false so that nothing in students table gets carried to the copy table.*

*Because where clause is never satisfied, => no values will get carried into the copy.*

3=4 is always false. We can also put[ where 0 or where false]

Aggregate functions:

* AVG() SUM()
* COUNT() -> Select count(\*) of students;

[Number of records]

* MIN() MAX()
* FIRST() – first element in the order which select statement would have returned the values. LAST() – last “ ”
* null values are counted too.

Q) Return avg of males and females respectively

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Gender | Score |

Select avg(score) from students group by(gender)

Q) Scalar Functions: Takes in one value and returns a value

* UCASE() => Uppercase version of the string
* LCASE() => Lowercase version of the string
* RAND() => Random
* ROUND()
* TO\_TIMESTAMP()
* NOW() => Current date and time

KEYS:

Primary keys and Foreign Keys:

Candidate Keys: A set of columns which can uniquely identify every other value in a record.

Candidate keys can have a single column or a set of columns

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Score | Gender |
| 1 | A | 20 | M |
| 2 | B | 30 | F |
| 3 | C | 31 | M |

In this table, Name can also uniquely identify records

But gender isn’t enough because there are multiple males and females having different name and scores.

|  |  |
| --- | --- |
| Student\_id | Book\_id |
| 1 | 101 |
| 1 | 120 |
| 1 | 113 |
| 1 | 121 |
| 1 | 180 |
| 2 | 11 |
| 2 | 113 |
| 2 | 21 |

Student\_id or book\_id, both don’t seem to be the primary key.

We can’t uniquely identify with student\_id or book\_id alone.

For this particular table, (student\_id, book\_id) together constitute the primary key.

To create primary key,

1. While creating table,

CREATE TABLE students (

id int,

name varchar(255),

score int,

primary key(id)

);

or

CREATE TABLE students (

id int primary key,

name varchar(255),

score int,

);

1. Alter table students

add primary key(id);

CONSTRAINTS:

1. UNIQUE – value in a column/group of columns should never be duplicated.
2. NOT NULL – should never be NULL/ empty.
3. PRIMARY KEY – Unique and NOT NULL
4. DEFAULT – set the field to a default value

Ex: gender char

gender should be only M/F salary table

|  |  |
| --- | --- |
| Id | Salary |
| 1 | 2000 |
| 2 | 3000 |
| 5 | 6000 |
| 3 | 4000 |
| 4 | 2000 |

1. FOREIGN KEY –

employee table

|  |  |  |
| --- | --- | --- |
| Id | Dept | Title |
| 1 | Tech | CTO |
| 2 | PM | PM-1 |
| 3 | PM | PM-2 |
| 4 | Tech | SWE-1 |

when I create employee\_salary table,

Create table employee\_salary(

id int,

salary int,

foreign key(id) references employee);

Alter table employee\_salary

add foreign key(id)

references employee;

|  |  |
| --- | --- |
| Id | Score |
| 1 | 50 |
| 2 | 60 |
| 3 | 20 |
| 4 | 10 |
| 5 | 70 |

DROP vs DELETE vs TRUNCATE

delete: removes a set of record in a table

Ex: Delete from students where score <= 40;

|  |  |
| --- | --- |
| Id | Score |
| 1 | 50 |
| 2 | 60 |
| 5 | 70 |

drop: Drops the entire table, removes complete existence of the table.

Ex: drop table students;

Truncate: Deletes all the rows. But table and it’s definition/constraints are still there.

|  |  |
| --- | --- |
| Id | score |

Ex: truncate table students;