Relational Database Design

Normalisation: Normalisation is a step-by-step process by which we can convert our database design to standard format. Using normalisation we can reduce the complexity of the database. It is used to reduce/eleminate redundancy inside a database and helps to improve the performance.

→ Normalization is used to remove anomalies such as - insertion, updation and deletion anamoly from the database.

1NF: A table is in 1NF if:

→ there are no repeated rows of data

→ columns only contain a single value

→ the table has a primary key

Student	Year	Subject
Ram	12	Math
Shyam	13	English
Ram	12	English, Science
Mohan	11	Math, English

This table is not in 1NF as it contains repeated rows & multiple values inside a column.

⇒

Student	Year	Sub
Ram	12	Math
Shyam	13	English
Ram	12	English
Ram	12	Science
Mohan	11	Math
Mohan	11	English

This table is in 1NF.

2NF: A table is in 2NF if:

→ it is in 1NF

→ every column that is not a primary key of the table is dependent on the whole of the primary key. Or

We may say the table must be in 1NF and should not contain any partial dependency i.e., any non-prime attribute is dependent on any proper subset of any candidate key of the table.

non-prime attribute → The attribute which does not appear in any of the candidate keys are called non-prime attributes.

Primary Key

Student	Subject	Grade	Age
Natasha W.	Maths	A	15
Natasha W.	English	B	15
Daniel J.	Maths	C	16
Simon B.	Chemistry	A	14
Emma T.	Geography	B	14

In this table Student + Subject column together form a primary key. Also, then all the records inside table are unique and are single-valued. Hence, it is in 1NF. Here, Grade and Age column are non-prime attribute. Now, proper subset of our candidate key can be — Student attribute, Subject attribute or student and subject attribute as a whole. and our non-prime attributes should not be dependent on these attributes in any way.

But when we consider Grade column, it must be dependent on Student + Subject column otherwise there is no sense of keeping Grade column in the table. Hence, it violates 2NF condition.

Similarly, Age column must be related to some student in the Student hence a dependency is also there. thus it violates 2NF. To remove ~~2NF~~ these partial dependencies, we can divide the table into two parts as -

Student	Subject	Grade
Natasha W.	Maths	A
Natasha W.	English	B
Daniel J.	Maths	C
Simon B.	Chemistry	A
Emma T.	Geography	B

Student	Age
Natasha W.	15
Daniel J.	16
Simon B.	14
Emma T.	14

3NF:

For a table to be in 3NF -

→ it must be in 2NF and

→ Every column that is not a primary key is only dependent on the whole of the primary key i.e., there should be no transitive dependencies.

Primary key

Subject	Year	Star Pupil	Star Pupil DOB
Math	2015	Mathew Taylor	1999-03-21
Physics	2015	William	1999-09-15
Chemistry	2015	Georgina	1998-11-04
Math	2016	Benjamin	2000-05-02
Math	2016	William	1999-09-15

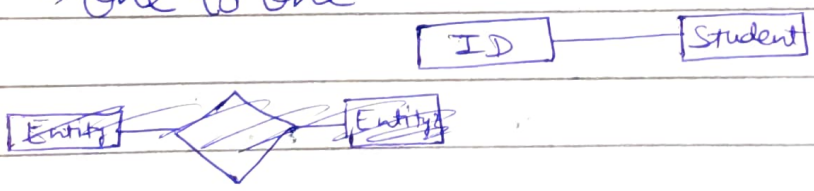
Not in 3NF because star pupil DOB column depends on star pupil which is not Primary key

Subject	Year	Star Pupil
Math	2015	Matthew T.
Physics	2015	William
Chemistry	2015	Georgina
Maths	2016	Benjamin
Math	2016	William

Star pupil	Star DOB
Matthew T.	1999-03-21
William	1999-09-15
Georgina	1998-11-04
Benjamin	2000-05-02
~~William~~	~~1999-09-15~~

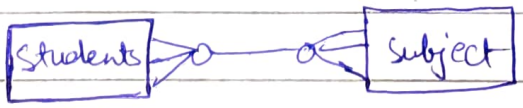
Relationships:

→ One to One



→ One to Many

→ Many to Many



Constraints:

→ UNIQUE →NOT NULL →CHECK
→ DEFAULT

Aggregate Functions: Aggregate function helps us to perform some calculation on data within a column and return one resultant row.

Eg: COUNT, SUM, MIN, MAX, AVERAGE

We can use GROUP BY clauses to group the results by one (or more) columns.

We can even use a HAVING clause in a similar way to a WHERE clause in a SELECT statement to filter the result set.

→ COUNT function → It doesn't count null values.

Eg: select count (col-name) from table-name;
select count (*) from customers where first-name is Null;

→ SUM function → Used to sum the numerical values in a table.

Eg: select Sum (col-name) from table-name;
select Sum (n-rows) from seats;

→ MIN function →
select min (col-name) from table-name;

→ MAX function →
select max (col-name) from table-name;

→ AVERAGE function →
select avg (col-name) from table-name;

Exercise-6

(1) How many bookings did customer id 10 make in Oct. 2017.
select count(*) from bookings where customer_id = 10;

(2) Count the number of screenings for Blade Runner 2049 in Oct. 2017.
select count(*) from ~~bookings where~~ films f join screenings s on s.film_id = f.id where f.name = 'Blade Runner 2049';

(3) Count the number of unique customers who made a bookings for October 2017.
select count(distinct(customer_id)) from bookings;

→ **Group by clause** : The features we're fetching if they are not used as aggregate function must also come under group by clause.
Select customer_id, count(id) from bookings group by customer_id;

→ **Having clause** : It acts as where statements but for groupby clause containing queries.

select customer_id, screening_id, count(id) from booking group by customer_id, screening_id having customer_id = 10;

Exercise-7

(1) Select the customer id and count the number of reserved seats grouped by customers for oct. 2017.
select b.customer_id, count(rs.id) from bookings b join reserved_seat rs on b.id = rs.booking_id group by b.customer_id;

(2) Select the film name and count the number of screenings for each film that is over 2 hours long.
select f.name, f.length_min, count(s.id) from screenings s join films f on s.film_id = f.id group by f.name, f.length_min having f.length_min > 20;