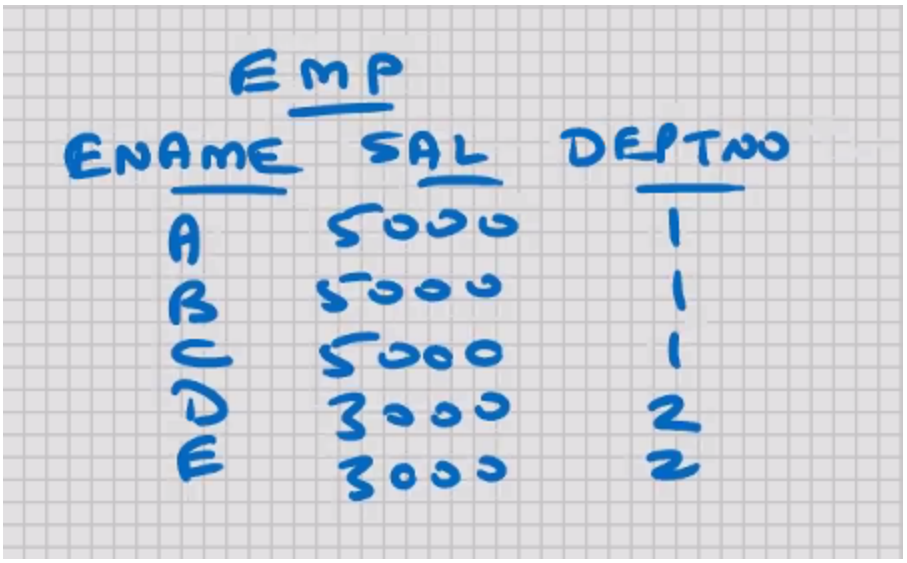
Day_12

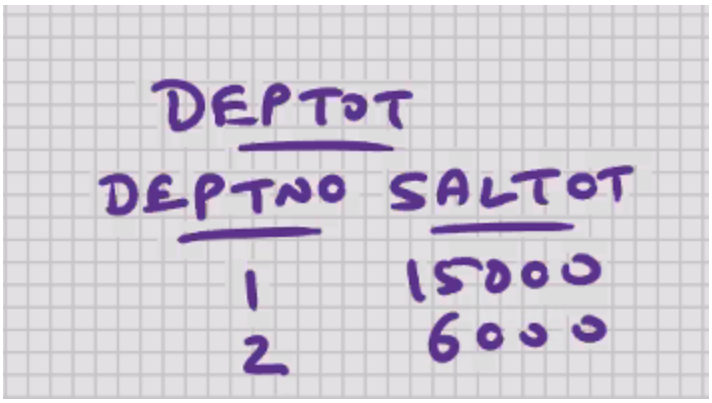
MySQL -STRORED OBJECTS

- objects that are stored in the database
- CREATE .... tables, indexes, views, stored procedures, stored functions
- anything that you do with CREATE command is a stored object

DATABASE TRIGGERS:

- present in some of the RDBMS
- Routine (set of commands) that gets executed automatically whenever some EVENT takes place
- Triggers are written on tables
- Events are:
  - before insert, after insert
  - before delete, after delete
  - before update, after update
- In MySQL, all triggers are at row level(will fire once for each row)
- In MySQL, you can have maximum 6 triggers per table






```
delimiter //
create trigger abc
before insert
on emp for each row
begin
    insert into tempp values(1, 'inserted');
end; //
delimiter ;
```

- if insert operation on table fails, then it will cause the even to fail, and trigger changes are automatically Rolled back
- if trigger fails, it will cause the event to fail, and the INSERT operation on table is automatically Rolled back
- YOUR DATA WILL ALWAYS BE CONSISTENT
- within the trigger, all MySQL-PL statements allowed, e.g. variables, cursors, IF statements, loops, sub-blocks, etc.
- whether you ROLLBACK or COMMIT AFTERWARDS, the data will always be consistent
- Rollback and Commit not allowed inside the trigger
- Rollback or Commit has to be specified AFTERWARDS, the data will always be consistent

---

- uses:
  - maintain logs(audit trails) of insertions
  - automatic data duplication, data replication, data mirroring

    maintain 2 or more copies of table in the event o INSERT
  - maintain SHADOW TABLES in the event of insert
  - maintain standby database in the event of insert
  - AFTER INSERT trigger is recommended

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        insert into tempp values(new.sal, new.ename);
end; //
delimiter ;
```

- new.ename, new.sal, new.deptno are MySQL created variables

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        set new.ename = upper(new.ename);
end; //
delimiter ;
```

```
*       new.ename, new.sal, new.deptno are MySQL created variables

Uses:-
*       Data cleansing BEFORE INSERT
```

```
create trigger abc
before insert
on emp for each row
begin
        if new.deptno = 1 then
                set new.sal = 5000;
        elseif new.deptno = 2 then
                set new.sal = 3000;
        else
                set new.sal = 2500;
        end if;
end; //
delimiter ;

Uses:-
*       Dynamic default values BEFORE INSERT
```

```
delimiter //
create trigger abc
before insert
on emp for each row
begin                                   (6000)    (3000)
        update deptot set saltot = saltot + new.sal
        where deptno = new.deptno;
end; //                  (2)
delimiter ;

Uses:-
*       auto-updation of related tables
        AFTER INSERT trigger is recommended
```

```
show triggers;          <- shows triggers from all schemas

show triggers from [db_name];

show triggers from cdacmumbai;

drop trigger abc

*       if you drop the table, then the indexes and triggers are dropped
        automatically
```

```
select * from information_schema.triggers
where trigger_schema = 'cdacmumbai';
```

```
*       if you share your table with the others, the indexes and triggers
        will be shared automatically
```

```
*       you can call stored procedures and stored functions inside
        the trigger
```

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
        insert into tempp values(1, 'deleted', user(), now(), etc.);
end; //
delimiter ;

Uses:-
*       maintain logs (audit trails) of deletions
        AFTER DELETE trigger is recommended
```

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
        insert into tempp values(old.sal, old.ename);
end; //
delimiter ;

*       old.ename, old.sal, old.deptno are MySQL created variables
```

```
Uses:-
*       maintain HISTORY tables in the event of delete
        AFTER DELETE trigger is recommended
*       ON DELETE CASCADE BEFORE DELETE
*       In the above, Set the child rows to NULL, BEFORE DELETE
```

```
Uses:-
*       auto-updation of related tables
        AFTER DELETE trigger is recommended
```

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
        update deptot set saltot = saltot - old.sal
        where deptno = old.deptno;
end; //
delimiter ;
```

- all triggers are at server level
- perform the DML operations using MySQL CLC, or MySQL Workbench, or any front-end s/w, the triggers will execute.

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        insert into tempp values(1, 'updated');
end; //
delimiter ;
```

```
Uses:-
*       maintain logs (audit trails) of updations
        AFTER UPDATE trigger is recommended
```

```
Uses:-
*       maintain SHADOW and HISTORY tables in the event of update
        AFTER UPDATE trigger is recommended
```

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        insert into tempp values(old.sal,old.ename);
        insert into tempp values(new.sal,new.ename);
end; //
delimiter ;

*       old.ename, old.sal, old.deptno, new.ename, new.sal, new.deptno
        are MySQL created variables
```

Cascading triggers → one trigger causes a second trigger to execute, which in turn a third trigger to execute, and so on.

- When all this cascading triggers are firing, any kind of power failure network failure, etc.; the entire transaction is automatically Rolled back.

Mutating tables error→ if some Cascading trigger causes one of the previous triggers to execute, then it will not go into infinite loop; you will get an error that the table is undergoing Mutation and the entire transaction is automatically Rolled back

```
delimiter //
create trigger xyz
before update
on emp for each row
begin                                (1500)    (5000)    (6000)
        update deptot set saltot = saltot - old.sal + new.sal
        where deptno = old.deptno;
end; //                  (1)
delimiter ;                              I
```

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        if old.sal <> new.sal then
                update deptot set saltot = saltot - old.sal + new.sal
                where deptno = old.deptno;
        end if;
end; //
delimiter ;
```

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        if old.sal <> new.sal or old.deptno <> new.deptno then
                if old.deptno <> new.deptno then
                     update deptot set satot = saltot - old.sal
                     where deptno = old.deptno;
                     update deptot set satot = saltot + new.sal
                     where deptno = new.deptno;
                else
                     /* if you are UPDATING THE SAL COLUMN ONLY */
                     update deptot set saltot = saltot - old.sal + new.sal
                     where deptno = old.deptno;
                end if;
        end if;
end; //
```

NORMALIZATION

- applicable for RDBMS (e.g. MySQL) and ORDBMS(e.g. Oracle)
- concept of the table design
- RDBMS → 1st to 4th Normal Form
- ORDBMS → 1st to 9th Normal Form
- what tables to create, structures, columns, datatypes, widths, constraints
- based on USER requirements

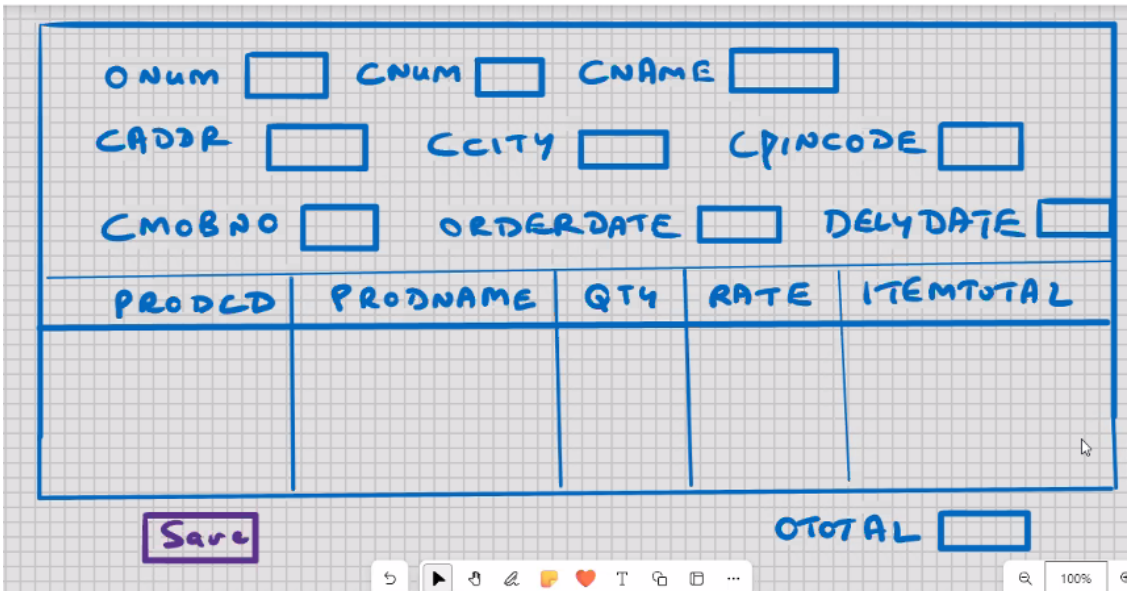
- part of Design phase(min 1/6), Coding(25%-33%),
- aim of Normalization is to have an ‘efficient’ table structure
- aim o Normalization is to avoid the data redundancy(avoid the unnecessary duplication of data)
- secondary aim of Normalization is to reduce the problems of insert, update, and delete
- Normalization is done from input perspective
- Normalization is done from Forms perspective
- VIEW THE ENTIRE APPLICATION ON A PER-TRANSACTION BASIS, AND YOU NORMALISE EACH TRANSACTION SEPARATELY

  e.g. CUSTOMER_PLACES_AN_ORDER, CUSTOMERS_CANCELS_THE-ORDER,

  GOODS_ARE-DELIVERS,

CUSTOMER_PLACES_AN_ORDER

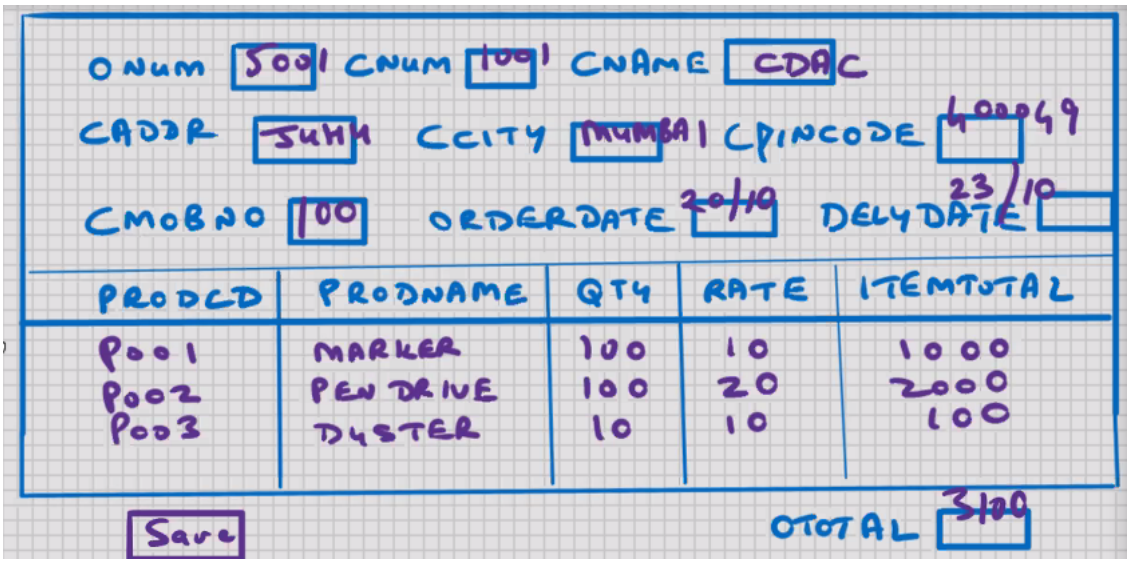
Getting ready for Normalization:



a. for a given transaction, make a list of fields

Onum
Cnum
Cname
Caddr
Ccity
Cpincode
Cmobno
Orderdate
Delydate
Prodcd
Prodname
Qty
Rate
Itemtotal
Ototal

b. Ask the client for some sample data



c. With the permission and involvement of Client, strive for atomicity

(Column is divided into sub-columns, and sub-columns are divided into sub-sub_columns)

d. For every column make a list of column properties

e. Get client sign-off

f. End of client interaction

g. Assign the datatype for each column

h. Assign the not null, unique, and check constraints

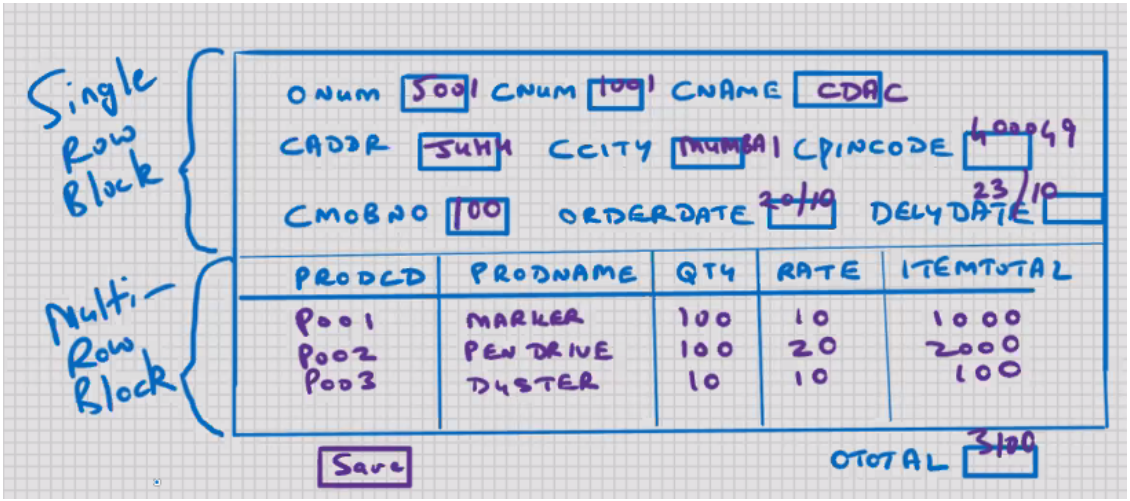
i. For all practical purposes, you can have a single table with all these columns

j. Remove the computed columns(e.g. itemtotal, ototal)

k. Key element will be the Primary key of the this table

- At this point, the data is in Un-Normalized Form (UNF)
- Un-Normalized Form→ starting point Normalisation

NORMALIZATION:



1. Remove the repeating group into a new table

Onum						
Cnum						Prodcd
Cname						Prodname
Caddr						Qty
Ccity						Rate
Cpincode						
Cmobno						
Orderdate						
Delydate						

2. the key element will be the primary key of the new table
3. (This step may or may not be required) Add the Primary key of the original table to the new table to give you a composite Primary key

Onum	Onum
Cnum	Prodcd
Cname	Prodname
Caddr	Qty
Ccity	Rate
Cpincode	
Cmobno	
Orderdate	
Delydate	

- above 3 steps are to be repeated infinitely till you cannot Normalise any further

FIRST NORMAL FORM(FNF) (SINGLE NORMAL FORM) (1NF)→ Repeating groups are removed from table design

- 1 : many is always encountered here
- DEPT and EMP tables are in First Normal Form

25%

---

4. Only the tables with Composite Primary key are examined
5. Those non-key elements that are not dependent on the entire Composite Primary key, they are to be removed into a new table

Onum	Onum	
Cnum	Prodcd	Prodname
Cname	Qty	Rate
Caddr		
Ccity		
Cpincode		
Cmobno		
Orderdate		
Delydate		

6. Key element on which originally dependent, it is to be added to the new table, and it will be the primary key of the new table

Onum						Onum			Prodcd
Cnum						Prodcd			Prodname
Cname						Qty			Rate
Caddr									
Ccity									
Cpincode									
Cmobno									
Orderdate									
Delydate									

SECOND NORMAL FORM (SNF) (DOUBLE NORMAL FORM) (2NF)→ every column is functionally dependent on primary key

FUNCTIONAL DEPENDENCY→ without Primary key, that column cannot function

67%

25% + 67% → 92%

---

7. Only the no-key elements are examined for inter-dependencies
8. Inter-dependent columns are to be removed into a new table

Onum			Cnum			Onum			Prodcd
Orderdate			Cname			Prodcd			Prodname
Delydate			Caddr			Qty			Rate
			Ccity						
			Cpincode						
			Cmobno						

9. the key element it will be the primary key of the new table, and the primary key of new table, that column, it is to be retained in the original table for relationship purposes.

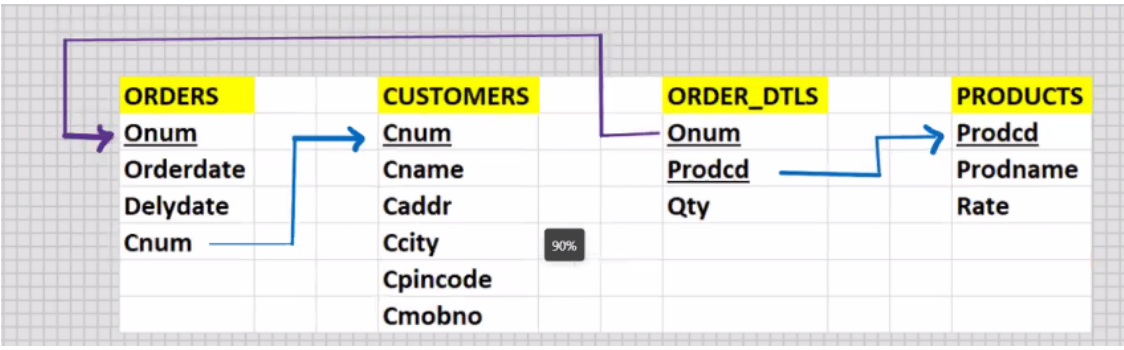
Onum	Cnum	Onum	Prodcd
Orderdate	Cname	Prodcd	Prodname
Delydate	Caddr	Qty	Rate
Cnum	Ccity		
	Cpincode		
	Cmobno		

- above 3 steps are to be repeated infinitely till you cannot Normalize any further

THIRD NORMAL FORM(TNF) (TRIPLE NORMAL FORM) (3NF)→ Transitive dependencies(inter-dependencies) are removed from table design

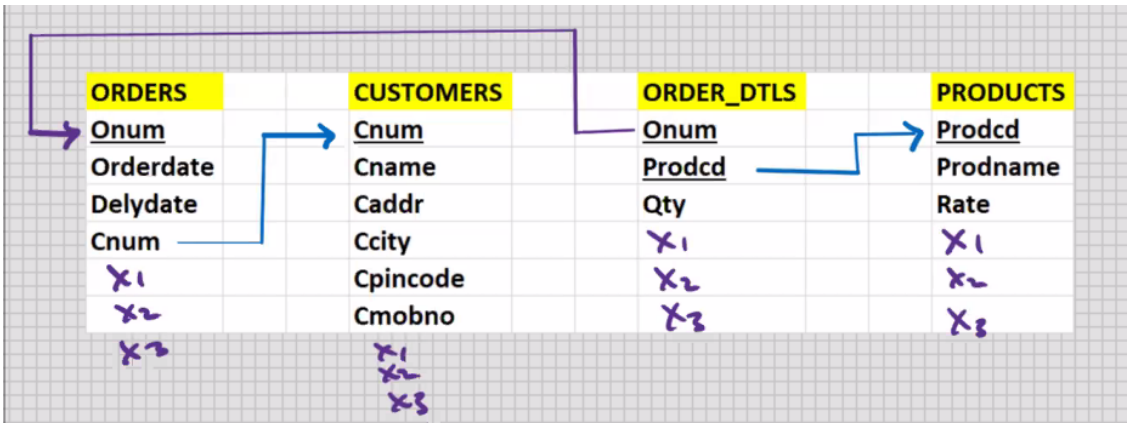
ORDERS	CUSTOMERS	ORDER_DTLS	PRODUCTS
Onum	Cnum	Onum	Prodcd
Orderdate	Cname	Prodcd	Prodname
Delydate	Caddr	Qty	Rate
Cnum	Ccity		
	Cpincode		
	Cmobno		

- Primary Key is a by-product of Normalisation



POST - NORMALISATION

- implement Extension columns



- reserve some columns for logs

Sameer Dehadrai

samdehadrai@yahoo.com
contact@sameerdehadrai.com

Linkedin.com

98201-34740