A Project Report
On

DATABASE ON THE FLY

By

Team F5

Name	NUID	Email
Rushabh Nisher	001400817	nisher.r@husky.neu.edu
Rishi Chopra	001825666	chopra.r@husky.neu.edu
Ashu Kapil	001400324	kapil.as@husky.neu.edu
Kavita Patidar	001357868	patidar.ka@husky.neu.edu
Simarjeet Singh Chhabra	001402447	chhabra.si@husky.neu.edu

under the guidance of
Prof. Chaiyaporn Mutsalklisana

For the Subject INFO6210
Data Management and Database Design

Northeastern University
April 19, 2019

Acknowledgement

We wish to express our sincere gratitude to Prof. Chaiyaporn Mutsalklisana, Instructor for course Data Management and Database Design, for providing us an opportunity to do our project work on ‘DATABASE ON THE FLY’

This project bears on imprint of many peoples. We sincerely thank our Teaching Assistant Nilesh Nerkar for his guidance and encouragement in carrying out this project work.

Finally, we would like to thank our colleagues and friends who helped us in completing the Project work successfully

Rushabh Nisher

Rishi Chopra

Ashu Kapil

Kavita Patidar

Simarjeet Singh Chhabra

Abstract

This paper describes a complete understanding of our project Database on the Fly, where we proposed a solution for the problems in existing system to handle data in stock control system and supervision of the demands and supply of products and services. Our motivation is to create a generalized skeleton to help clients in making a well-structured data from unstructured data that would be less time consuming. To create this skeleton in MySQL, we used a sequence of codes that included queries, subqueries, joins, views, triggers, procedures, backups, ERD, functions, grants. To support our solution proposal, we have made three assumptions and explicitly mentioned their pros and cons.

In this project we tried to find out the way to automate the process of database generation and customize it as per the client requirements. Specific care has been taken to ensure that the databases are generated with minimum manual efforts and therefore it emphasizes on reducing any expert to manage it on regular basis.

While were able to answer to answer most of the critical issues with our application, some of automation concepts are yet to be implemented and will be researched in future.

Contents

1. Introduction

The project Database on the Fly is a complete desktop-based application designed on MySQL. The main aim of the project is to develop a common skeleton for all domain that can convert unstructured data to structured data. There will be one login for admin at company level. Two department namely 'Hiring' and 'Finance' will be common for every company and 'Department Others' will be changing according to specific domain. There is a proper relation among company, client and supplier which you can understand with the help of Entity Relationship Diagram.

a. Motivation

To help clients in making a well-structured data from unstructured data that would be less time consuming and helpful in tracking of demand and supply of products and services, we proposed this system called "Database on the Fly".

- Creating a common database skeleton
- Specializing it according to the needs of the client
- Making a function to create departments of companies
- Trying to tackle with unstructured data

b. **Existing System Problems:** As we know manual system are quite tedious, time consuming and less efficient. So, following are some disadvantages of the existing system:

1. Generating any database is quite a tedious and costly task
2. Businesses in similar industry domains tend to have very different type of data management system
3. Lack of a common type of skeleton
4. No set methods to convert a data stream to a database
5. Time consuming
6. Less accurate
7. Lot of manual work
8. Slow data processing
9. Not user-friendly environment

2. Proposed System

We are proposing a system which automates the process of database generation, in such a way that with minimal information from the Client's we'll be able to make a database. The way we're proposing to do this is by creating a common database skeleton. This skeleton can be used by all the companies. This skeleton structure is the one that is present in almost all companies. Once we have this skeleton, all we need to do is specialize according to the needs of the client. This part can be automated by using functions to create tables.

a. Assumptions

i. Clients will provide us with the Domain, Sub-Domain and other basic Information, necessary to make the Database
ii. We are not tackling with each and every intricacy of the department structure of each company (i.e. presence of various departments and relationship between them)
iii. We are just showing departments of one company for representational purpose, the process can be replicated easily
iv. Data Stream will contain all the data of the organization
v. This large file will be a CSV or a JSON file
vi. Functional dependencies of columns not given in the data-stream provided

3. Data Stream

The need of maintaining a database is evident. When a company comes with their needs, we would ask basic questions like the domain and the sub-domain of the company. We would also ask about the other specifics how the company works. This part would make each company unique. But what about the scenarios where we don't have the access to these specifics of the company. What if we only have the access to the data stream of the company. How would we be able to convert this into a SQL Table?

Does a method exist, wherein we can convert the data-stream into MySQL Database? Here we try to answer this question

There are 3 major ways how we can use a data-stream and convert it into a MySql table.

1. Views – We could create various views and essentially use them as our tables, this method can be queried and provides a sense of security. But a major problem in this is that it can't be updated. The data, when once inserted will not change unless there is insertion in the table.
2. Normalization – This, is the best way to convert a huge dataset into small tables. But we need to do this manually, and a method in SQL doesn't exist to normalize the data given a huge table. For Normalization an understanding of the facts is required, which is outside the scope of automated tools. Although a few tools exist, in which we feed the functional dependency and it gives us the normalized form. But still, it is not possible to create a functional dependency by a tool. A possible way to automate is writing queries like,
   ```
   INSERT INTO <new_table>
   SELECT DISTINCT <columns names>
   FROM <massive_table>;
   ```
   But this too would require us to select column names (decide functional dependencies by ourselves)
3. Vertical Partitioning – This is yet another interesting approach, but this feature hasn't yet been added to MySQL and there are no plans to include it anytime soon.

4. DDL

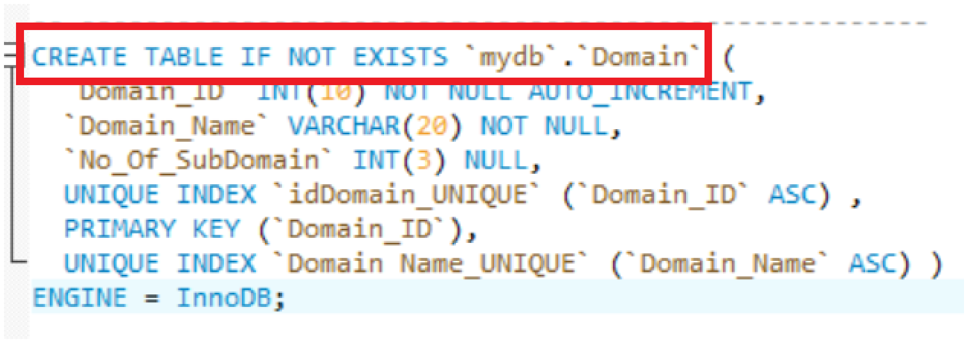
DDL (Data Definition Language)

DDL or Data Definition Language consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER- is used to alter the structure of the database.
- TRUNCATE– is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT – is used to add comments to the data dictionary.
- RENAME – is used to rename an object existing in the database.

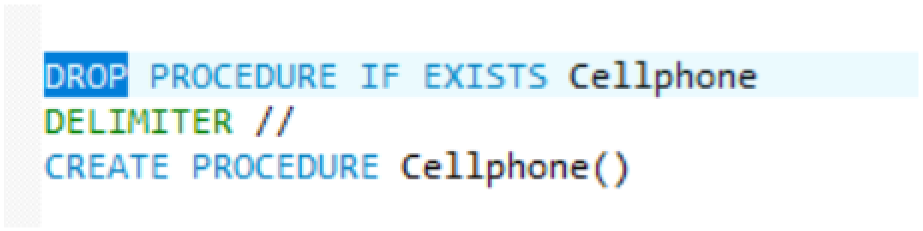
In our Final Project, we have made sufficient use of CREATE commands to create tables in our database.



```
CREATE TABLE IF NOT EXISTS `mydb`.`Domain` (
   `Domain_ID` INT(10) NOT NULL AUTO_INCREMENT,
   `Domain_Name` VARCHAR(20) NOT NULL,
   `No_Of_SubDomain` INT(3) NULL,
  UNIQUE INDEX `idDomain_UNIQUE` (`Domain_ID` ASC) ,
  PRIMARY KEY (`Domain_ID`),
  UNIQUE INDEX `Domain_Name_UNIQUE` (`Domain_Name` ASC) )
ENGINE = InnoDB;
```

In this image we can see the use CREATE command. The marked statement is used to check if the table named 'Domain' exists in the database 'mydb' and if not to create it.

Similarly, we have used DROP command in our Project. This command allows us drop any procedure named 'Cellphone' so that we can create a new procedure called as 'Procedure'. These are some examples of the use of DDL in our project.



```
DROP PROCEDURE IF EXISTS Cellphone
DELIMITER //
CREATE PROCEDURE Cellphone()
```

5. DML

DML (Data Manipulation Language)

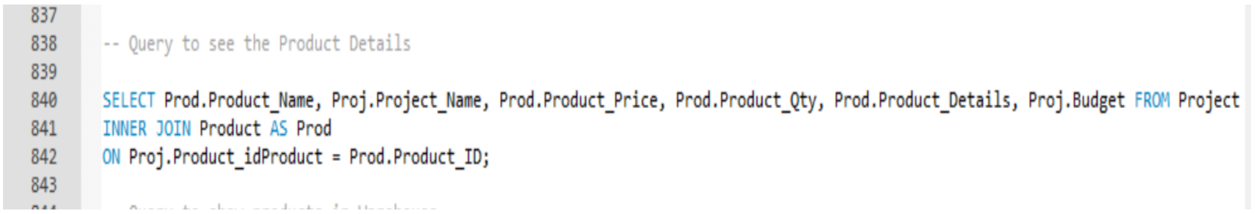
The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements. There are two types of DML: procedural, in which the user specifies what data is needed and how to get it; and nonprocedural, in which the user specifies only what data is needed.

Examples of DML commands:

- SELECT – is used to retrieve data from a database.
- INSERT – is used to insert data into a table.
- UPDATE – is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

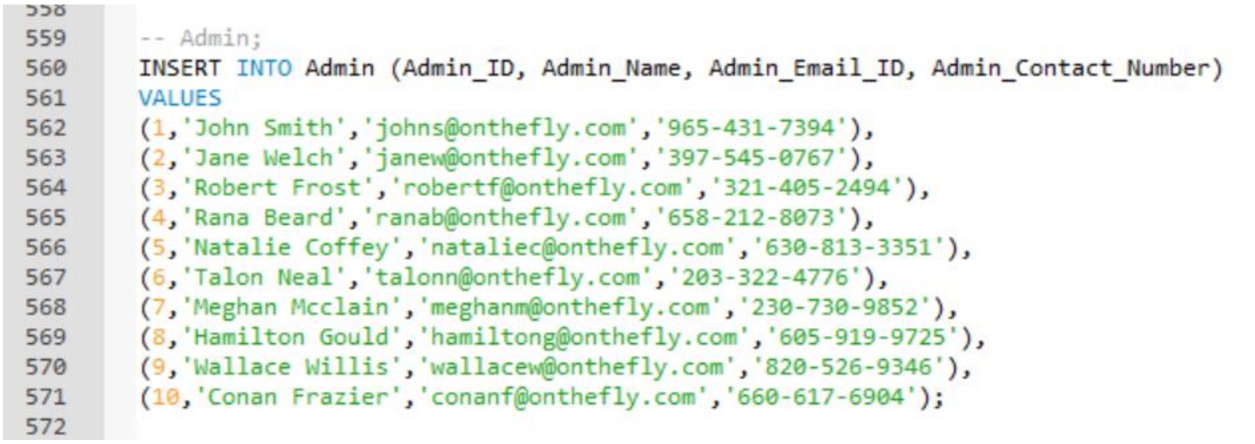
Images attached below are snippets of our code that show the use of SELECT, INSERT. SELECT is used to retrieve data from our database while INSERT is used to add data into our existing tables.

In the first image select is used to retrieve the product name, project name product price, product quantity, product details, etc. from the result of the join of project and product tables.



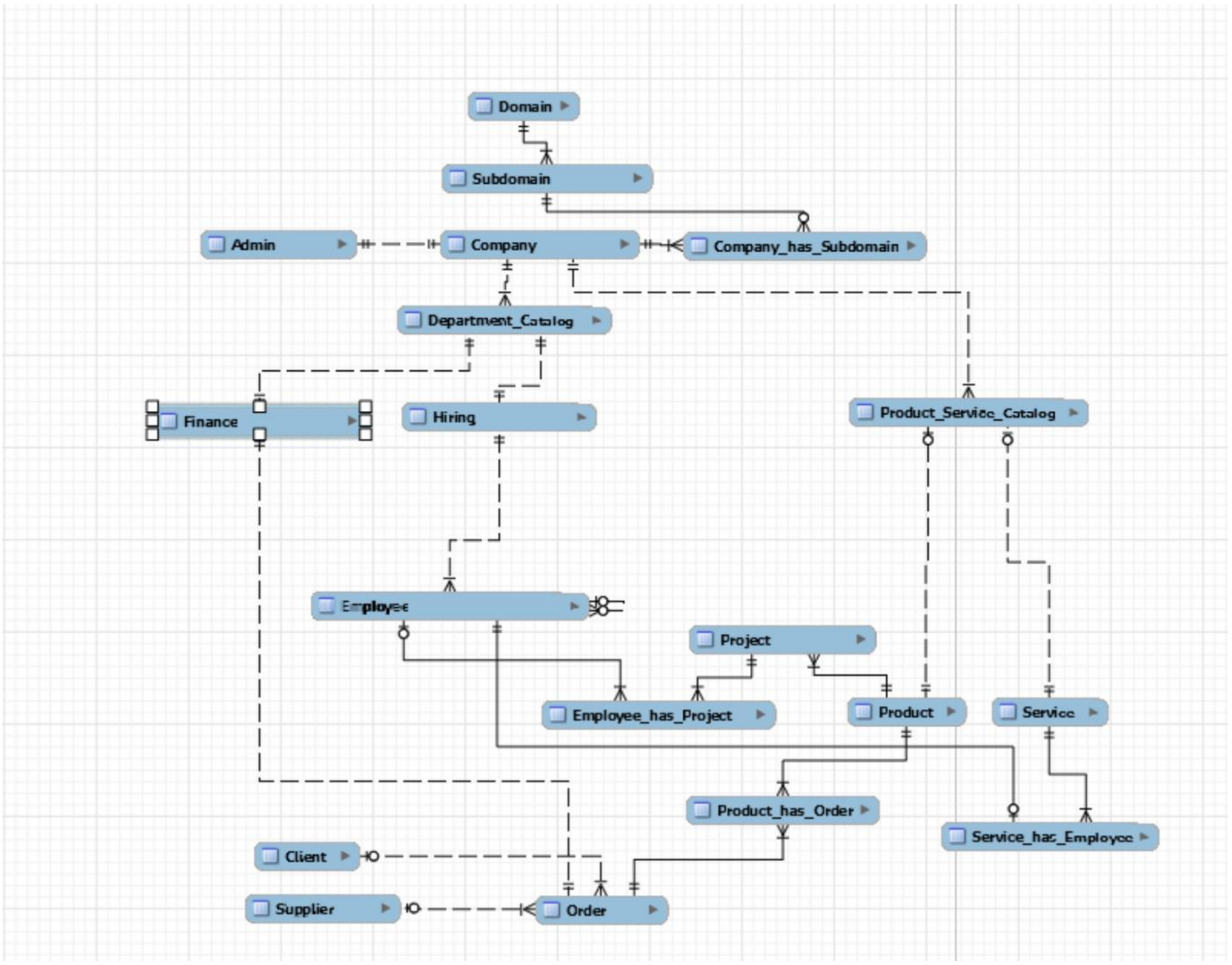
```
-- Query to see the Product Details

SELECT Prod.Product_Name, Proj.Project_Name, Prod.Product_Price, Prod.Product_Qty, Prod.Product_Details, Proj.Budget FROM Project
INNER JOIN Product AS Prod
ON Proj.Product_idProduct = Prod.Product_ID;
```

In the image below, INSERT is used to add data into the admin table.



```
-- Admin;
INSERT INTO Admin (Admin_ID, Admin_Name, Admin_Email_ID, Admin_Contact_Number)
VALUES
(1,'John Smith','johns@onthefly.com','965-431-7394'),
(2,'Jane Welch','janew@onthefly.com','397-545-0767'),
(3,'Robert Frost','robertf@onthefly.com','321-405-2494'),
(4,'Rana Beard','ranab@onthefly.com','658-212-8073'),
(5,'Natalie Coffey','nataliec@onthefly.com','630-813-3351'),
(6,'Talon Neal','talonn@onthefly.com','203-322-4776'),
(7,'Meghan Mcclain','meghanm@onthefly.com','230-730-9852'),
(8,'Hamilton Gould','hamiltong@onthefly.com','605-919-9725'),
(9,'Wallace Willis','wallacew@onthefly.com','820-526-9346'),
(10,'Conan Frazier','conanf@onthefly.com','660-617-6904');
```

6. ERD


Domain
Subdomain
Admin
Company
Company_has_Subdomain
Department_Catalog
Finance
Hiring
Product_Service_Catalog
Employee
Project
Employee_has_Project
Product
Service
Product_has_Order
Service_has_Employee
Client
Supplier
Order

Domain: -

Primary Key - Domain_ID

Attributes – 1. Domain_Name

2. No._Of_SubDomain

This tables is the primary segregation of companies. As we have the goal of creating databases for different companies in an automated way, the idea is to come up with a basic structure of databases. For example, if the client is Toyota, we can enter domain as Automobiles. This allows us a basic idea of what the structure of databases should be. Our assumption is that similar domain companies have similar structure. The main attributes of this domain are Domain_Name and No._Of_SubDomain. No. of subdomain signifies how many categories can be made from a domain. For example, if the domain is Automobiles, subdomain can be two i.e. Four Wheelers and Two wheelers.

Attribute	Primary Key	Foreign Key	Data type
Domain_ID	Yes	No	Varchar(5)
Domain_Name	No	No	Varchar(20)
No._Of_SubDomain	No	No	Int(3)

Sub Domain: -

Primary Key - SubDomain_ID

Foreign Key – Domain_ID

Attributes – 1. SubDomain_Name

2. No._Of_Comapnies

This tables is the second table in our database structure. Once the client has finalized the type of domain it falls in, we need to specify the type of subdomain it has. This means further classifying the domain and breaking down into smaller structure. This is necessary as we need our database to be in 2NF form. Therefore, to avoid redundancy and any anomaly, classification of domain is required.

The primary key in this table is SubDomain_ID which will be unique for a subdomain. There will be a foreign key also in this table for Domain namely Domain_ID. This will establish the relationship between Domain and Subdomain. The other attributes of this table are subdomain name which gives us the naming for subdomain and No. of companies in a subdomain. For example, if Domain is Automobile, subdomain name will be two wheelers and four wheelers and no. of companies can be how many companies are there in the subdomain. Like if we have two companies in 4 wheelers Toyota and Mercedes, the No._Of_comapnies will be 2.

Attribute	Primary Key	Foreign Key	Data type
SubDomain_ID	Yes	No	Int
Domain_ID	No	Yes	Int
Domain_Name	No	No	Varchar(25)
No._Of_SubDomain	No	No	Int(3)

Admin: -

Primary Key - Admin_ID

Foreign Key – NA

Attributes – 1. Admin_Name

2. Admin_Email_ID

3. Admin_Contact_Number

This table belongs to the admin of the database. As per our design concept every company will be given a dedicated admin. Only admin will have all the privileges of the databases. Like create, update, delete etc. This admin will be single point of contact for the client in addressing all his issues related to databases. The admin will have a unique called Admin_ID which will be the primary key for the table. There will be no foreign key admin as admin will be independent entity.

For example if we take the case of Toyota, we will assign a dedicated admin for Toyota for all the issues pertaining to the databases.

Attribute	Primary Key	Foreign Key	Data type
Admin_ID	Yes	No	Varchar(5)
Admin_Name	No	No	Varchar(20)
Admin_Email_ID	No	No	Varchar(45)
Admin_Contact_Number	No	No	Int(10)

Company: -

Primary Key - Company_ID

Foreign Key – Admin_ID

Attributes – 1. Company_Name

2. Address

3. Contact_Number

4. Email_ID

5. No._Of_Department

6. No._Of_Employees

This table gives the detail about the company. Once the client selects his domain and subdomain, his company is assigned an Admin. Admin then creates the databases for that company. The company table has a primary key as Company_ID. It also has a foreign key as Admin_ID. The other attributes for company table are Company_Name, Address, Contact_Number, Email_ID. All the columns can give complete overview of the company.

Attribute	Primary Key	Foreign Key	Data type
Company_ID	Yes	No	Varchar(5)
Admin_ID	No	Yes	Varchar(5)
Company_Name	No	No	Varchar(45)
Address	No	No	Varchar(100)
Email_ID	No	No	Varchar(45)
Contact_Number	No	No	Int(20)
No._Of_Department	No	No	Int(10)
No._Of_Employees	No	No	Int(6)

Company has Subdomain: -

Primary Key: SubDomain_idSubDomain

Primary Key: SubDomain_Domain_idDomain

This table shows the relationship between subdomain and company. The primary key is a composite key with domain id and subdomain id. It says that subdomain has a company.

Attribute	Primary Key	Foreign Key	Data type
SubDomain_idSubDomain	Yes	No	Int
SubDomain_Domain_idDomain	Yes	No	Int

Department Catalog: -

Primary Key - Department_ID

Foreign Key – Company_ID

Attributes – 1. Department_Name

2. Department_Head

3. No._Of_Employees

This table gives the information about the departments of the company. Departments are uniquely created based on domain. Once a domain is selected departments are created with the help of stored procedure. The primary key of this table is Department_ID. It also has a foreign key of Company_ID. The other attributes are Department_Name, Department_Head and No._Of_Employees.As this is the catalog for department it provides the list of departments a company has.

Attribute	Primary Key	Foreign Key	Data type
Department_ID	Yes	No	Varchar(6)
Comapny_ID	No	Yes	Varchar(6)
Department_Name	No	No	Varchar(20)
Department_Head	No	No	Varchar(20)
No._Of_Employees	No	No	Int(5)

Finance: -

Primary Key - Transaction_ID

Foreign Key – Department_Catalog_iddepartment

Attributes – 1. Transaction_Type

2. Transaction_Ref

3. Transaction Amount

4. Transaction Date

This table provides the details for finance department. As per structure of database, we have assumed that Finance department will be the common department for any type of company and will be responsible for all the transaction. All the suppliers will be paid through Finance and all the money will received from clients via this department. The primary key is Transaction_ID while department_ID will be foreign key. Other attributes are Transaction type, transaction ref, transaction amount and Transaction date.

Attribute	Primary Key	Foreign Key	Data type
Transaction_ID	Yes	No	Varchar(6)
Department_Catalog_iddepartment	No	Yes	Varchar(6)
Transaction_Type	No	No	Varchar(15)
Transaction_Ref	No	No	Varchar(45)
Transaction Amount	No	No	Int(10)
Transaction Date	No	No	Date

Hiring: -

Primary Key - Employee_ID

Foreign Key – Department_Catalog_iddepartment

Attributes – 1. Joining Date

2. End Date

3. Salary

This table provides the details for Hiring department. As per structure of database, we have assumed that Hiring department will be the common department for any type of company and will be responsible for all the hiring of employees. All the employee database will be managed through hiring department. The primary key is Employee_ID while department_ID will be foreign key. Other attributes are Joining Date, End Date and Salary.

Attribute	Primary Key	Foreign Key	Data type
Employee_ID	Yes	No	Int(10)
Department_Catalog_iddepartment	No	Yes	Int(10)
Joining Date	No	No	Date
End Date	No	No	Date
Salary	No	No	Int(6)

Employee: -

Primary Key - idEmployee

Foreign Key – Department_Hiring_iddepartment

Attributes – 1. Employee_Name

2. Department_ID

3. Employee_Email

4. Employee_Address

5. Employee_Phone_Number

6. Role

This department gives the detail about the employee of company. All the complete database of the employee is mentioned here. The primary key is Employee_ID while department_ID is the foreign key. Other attributes are Employee_Name, Department Name, Employee_Email, Employee_Address, Employe_Phone_Number and Role.

Attribute	Primary Key	Foreign Key	Data type
Employee_ID	Yes	No	Int
Department_Hiring_iddepartment	No	Yes	Int
Employee_Name	No	No	Varchar (45)
Department_ID	No	No	Int
Employee_Email	No	No	Varchar (45)
Employee_Phone_Number	No	No	Int (20)
Employee_Address	No	No	Varchar (100)
Role	No	No	Varchar (10)

Product_Service_Catalog: -

Primary Key – Product_Service_ID

Foreign Key – Company_ID

Attributes – 1. Name

2. Type

This table gives the detail about the products/services that the company has to offer. All the complete database of the products is mentioned here. This can also be referenced to the catalog of a company whose products are at display. For example, if company is Apple then its products are Iphone, Ipad, Macbooks etc while services can be Repair etc. The primary key is Product_Service_ID while Company_ID is the foreign key. Other attributes are Name and Type.

Attribute	Primary Key	Foreign Key	Data type
Product_Service_ID	Yes	No	Int
Company_ID	No	Yes	Int
Name	No	No	Varchar (45)
Type	No	No	Varchar (10)

Product: -

Primary Key – Product_ID

Foreign Key – Product_Service_catalog_ID

Attributes – 1. Product_Name

2. Product_Price

3. Product_Qty

4. Product_Details

This table gives the detail about the products of company. All the complete database of the product is mentioned here. The primary key is Product_ID while Product_Service_Catalog_ID is the foreign key. Other attributes are Product_Name, Product_Price, Product_Qty, Product_Details.

Attribute	Primary Key	Foreign Key	Data type
Product_ID	Yes	No	Varchar (6)
Product_Service_Catalog_ID	No	Yes	Varchar (10)
Product_Name	No	No	Varchar (45)
Product_Price	No	No	Int (10)
Product_Qty	No	No	Int (10)
Product_Details	No	No	Varchar (100)

Service: -

Primary Key – Service_ID

Foreign Key – Product_Service_catalog_ID

Attributes – 1. Service_Name

2. Service_Charge

This table gives the detail about the services of company. All the complete database of the services is mentioned here. The primary key is Service_ID while Product_Service_Catalog_ID is the foreign key. Other attributes are Service_Name, Service_Charge.

Attribute	Primary Key	Foreign Key	Data type
Service_ID	Yes	No	Int
Product_Service_Catalog_ID	No	Yes	Int
Servivce_Name	No	No	Varchar (45)
Service_Charge	No	No	Int (5)

Project: -

Primary Key – Project_ID

Foreign Key –Product_ID

Attributes – 1. Project_Name

2. No._Of_Employees

3. Budget

4. Start_Date

5. End_Date

This table gives the detail about the Projects that are currently going on in company. All the complete database of the services is mentioned here. We are assuming that a single project will lead to a product that will further go for complete production. The primary key is Project_ID while Product_ID is the foreign key. Other attributes are Project_Name, No._Of_Employees, Budget, Start_Date and End_Date.

Attribute	Primary Key	Foreign Key	Data type
Project_ID	Yes	No	Int
Product_ID	No	Yes	Int
Project_Name	No	No	Varchar (45)
No._Of_Employees	No	No	Int (3)
Budget	No	No	Int (10)
Start_Date	No	No	Date
End_Date	No	No	Date

Client: -

Primary Key – Client_ID

Foreign Key –N/A

Attributes – 1. Client_Name

2. Client_Address

3. Client_Email_ID

4. Client_Phone_Number

5. Order_ID

This table gives the detail about the Clients that are currently onboarded in company. All the complete database of the clients is mentioned here. We are assuming that a client can have multiple orders and the order ID should be mentioned in client table.The primary key is Client_ID while there is no foreign key. Other attributes are Client_Name, Client_Address, Client_Email_ID, Client_Phone_Number and Order_ID

Attribute	Primary Key	Foreign Key	Data type
Client_ID	Yes	No	Int
Client_Name	No	No	Varchar (45)
Client_Address	No	No	Varchar (100)
Client_Email_ID	No	No	Varchar (45)
Client_Phone_Number	No	No	Int (10)
Order_ID	No	No	Varchar (10)

Supplier: -

Primary Key – Supplier_ID

Foreign Key –N/A

Attributes – 1. Supplier_Name

2. Supplier_Address

3. Supplier_Email_ID

4. Supplier_Phone_Number

5. Order_ID

This table gives the detail about the Suppliers that are currently onboarded in company. All the complete database of the Suppliers is mentioned here. We are assuming that a supplier can have multiple orders and the order ID should be mentioned in suppler table.The primary key is Supplier_ID while there is no foreign key. Other attributes are Supplier_Name, Supplier_Address, Supplier_Email_ID, Supplier_Supplier_Number and Order_ID

Attribute	Primary Key	Foreign Key	Data type
Supplier_ID	Yes	No	Int
Supplier_Name	No	No	Varchar (45)
Supplier_Address	No	No	Varchar (100)
Supplier_Email_ID	No	No	Varchar (45)
Supplier_Phone_Number	No	No	Int (10)
Order_ID	No	No	Varchar (10)

Order: -

Primary Key – Order_ID

Foreign Key – Department_Finance_Transaction_ID

Attributes – 1. Supplier_ID

2. Client_ID
3. Order_Type
4. Name
5. Order_Value
6. Order_Status

This table gives the detail about the Orders that are processing in company. All the complete database of the orders are mentioned here. We are assuming that an order can be placed by client or can be given to supplier for some raw material needed for production. The primary key is Order_ID while the foreign key is Department_Finance_Transaction_ID. Other attributes are Supplier_Name, Client_ID, Order_Type, Name, Order_Value, Order_Status.

Attribute	Primary Key	Foreign Key	Data type
Order_ID	Yes	No	Varchar (6)
Department_Finance_Transaction_ID	No	Yes	int
Supplier_ID	No	No	Varchar (6)
Client_ID	No	No	Varchar (6)
Order_Type	No	No	Varchar (3)
Name	No	No	Varchar (45)
Order_Value	No	No	Int (7)
Order_Status	No	No	Varchar (10)

7. EERD

EER is a high-level data model that incorporates the extensions to the original ER model.

It is a diagrammatic technique for displaying the following concepts

- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation

These concepts are used when the comes in EER schema and the resulting schema diagrams called as EER Diagrams.

Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with ⓓ symbol.

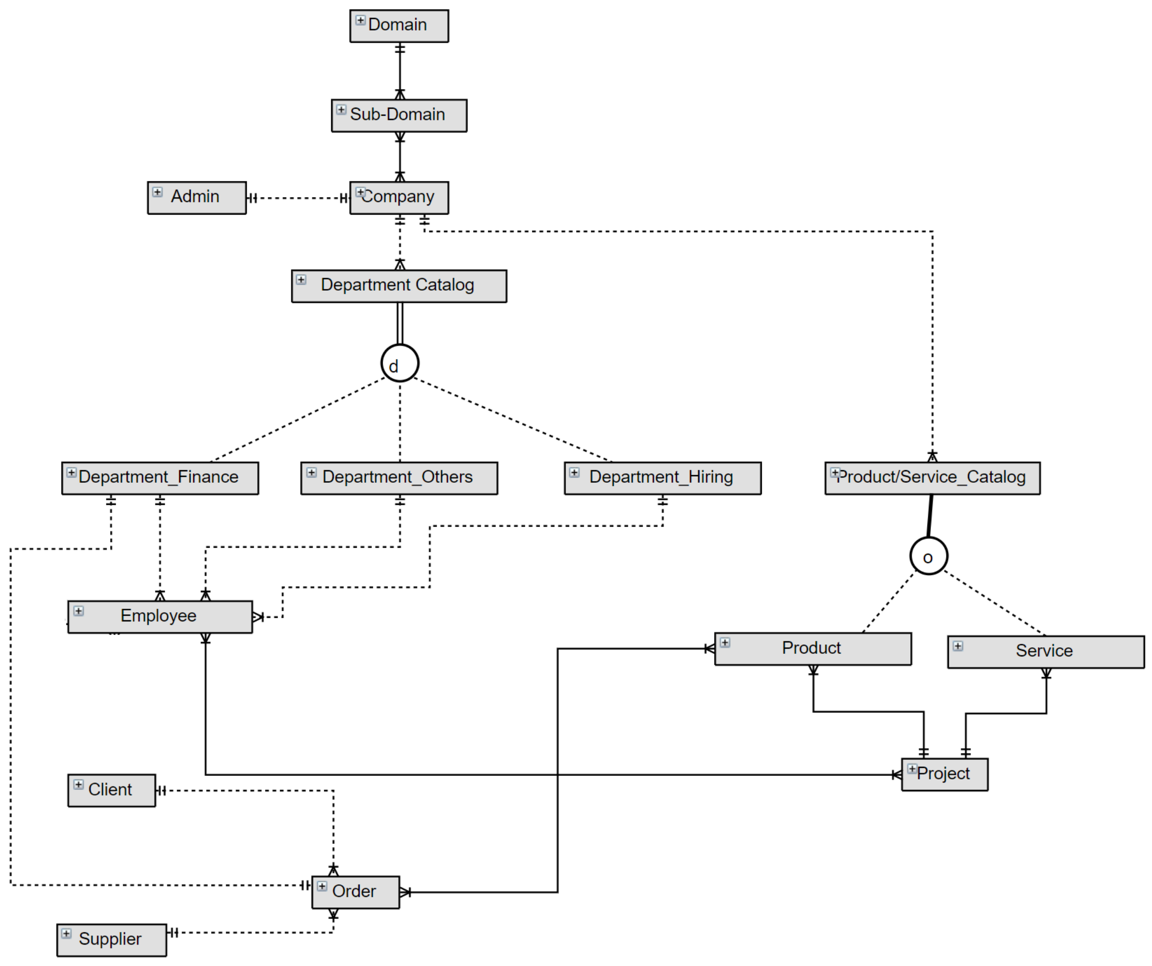
1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.

An entity cannot exist in database merely by being member of any super class.
For example: Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.
  For example: Square, Circle, Triangle are the sub class of Shape super class.



In our project we have two Super type and Sub type relationship:

1. Department catalog and Departments
2. Product Service Catalog and Products and Service

1. **Department Catalog**:
   This is supertype class. It has details of all the departments of the company.

   Departments:
   This is the sub type of Department Catalog. This is disjoint relationship as there can be only one department against a department catalog. This also means that this is total participation with supertype.

2. **Product Service Catalog**:
   This is supertype class. It has details of all the products/service of the company.

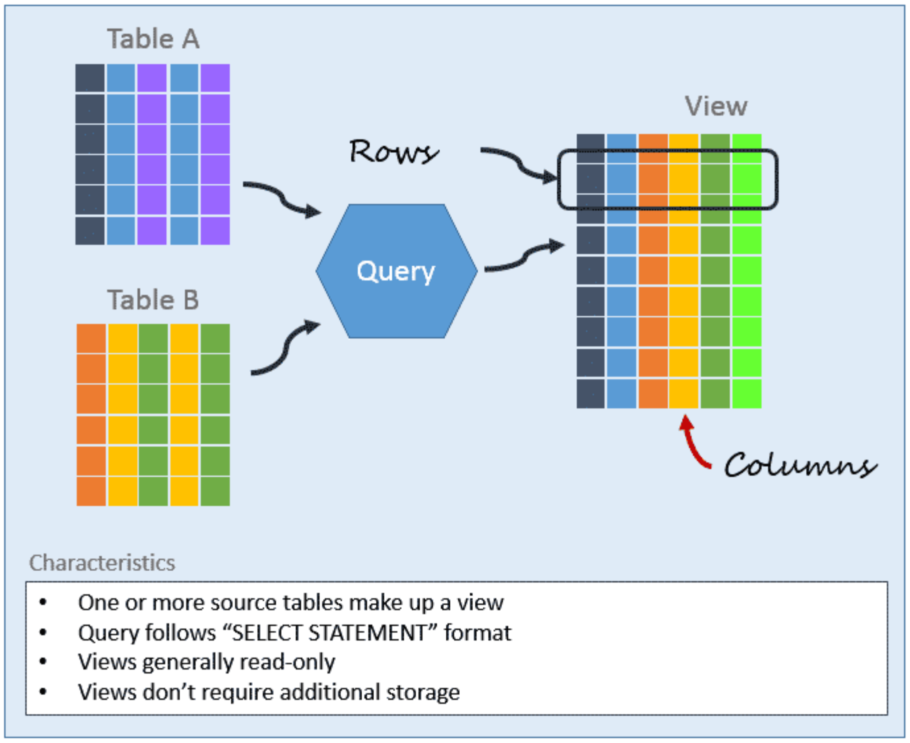
Product and Service:
We assume that either the company will offer a product or a service. Therefore, there will be disjoint rule and the subtype will be in total participation.

8. Views

A database view is a searchable object in a database that is defined by a query. Though a view doesn't store data, some refer to a view as "virtual tables," you can query a view like you can a table. A view can combine data from two or more table using joins and also, just contain a subset of information. This makes them convenient to abstract, or hide, complicated queries.[1] Since the joins are already made and the tables are virtual, it's quicker to run a query on that dataset.

A view can be depicted easily according to the below diagram:

Anatomy of a View

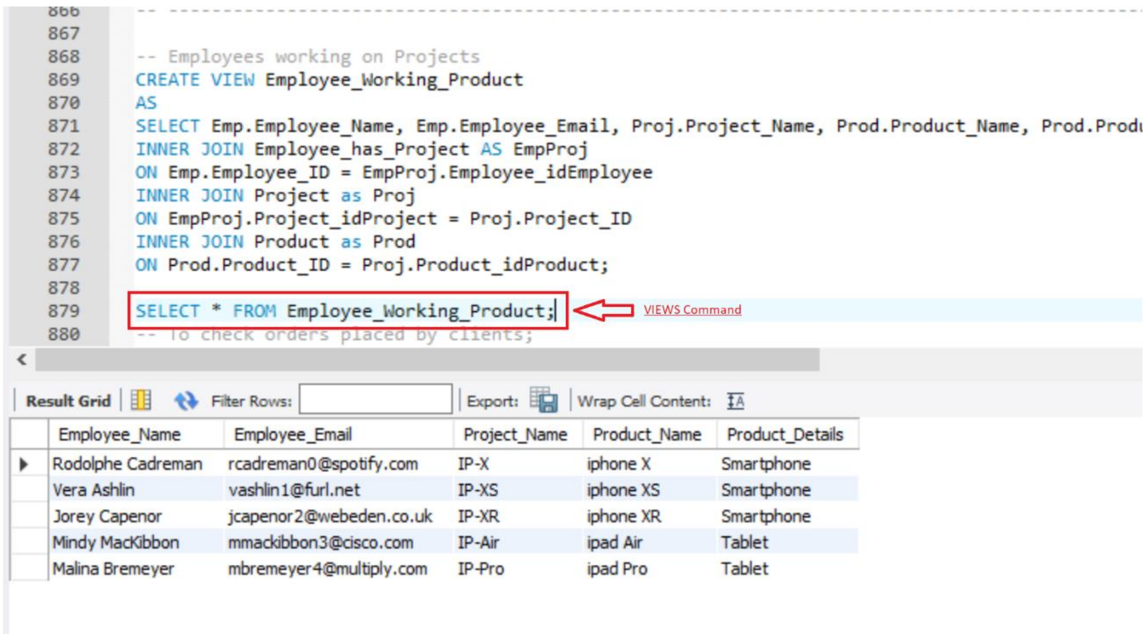


- One or more source tables make up a view
- Query follows "SELECT STATEMENT" format
- Views generally read-only
- Views don't require additional storage

Image 1: Anatomy of a View

Views are used for security purposes in databases, views restrict the user from viewing certain column and rows which means that by using view we can apply the restriction on accessing particular rows and columns for specific user.

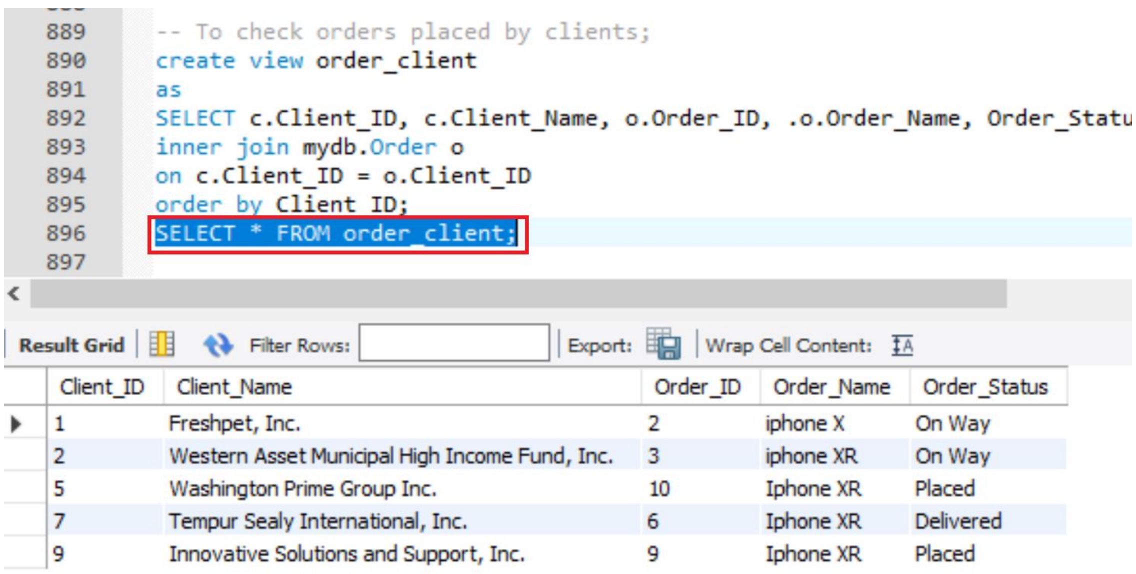
In our project we have used views to represent complex queries in a single line command.



```
-- -----------------------------------------------------------------------

-- Employees working on Projects
CREATE VIEW Employee_Working_Product
AS
SELECT Emp.Employee_Name, Emp.Employee_Email, Proj.Project_Name, Prod.Product_Name, Prod.Produ
INNER JOIN Employee_has_Project AS EmpProj
ON Emp.Employee_ID = EmpProj.Employee_idEmployee
INNER JOIN Project as Proj
ON EmpProj.Project_idProject = Proj.Project_ID
INNER JOIN Product as Prod
ON Prod.Product_ID = Proj.Product_idProduct;

SELECT * FROM Employee_Working_Product;
-- To check orders placed by clients;
```

Employee_Name	Employee_Email	Project_Name	Product_Name	Product_Details
Rodolphe Cadreman	rcadreman0@spotify.com	IP-X	iphone X	Smartphone
Vera Ashlin	vashlin1@furl.net	IP-XS	iphone XS	Smartphone
Jorey Capenor	jcapenor2@webeden.co.uk	IP-XR	iphone XR	Smartphone
Mindy MacKibbon	mmackibbon3@cisco.com	IP-Air	ipad Air	Tablet
Malina Bremeyer	mbremeyer4@multiply.com	IP-Pro	ipad Pro	Tablet

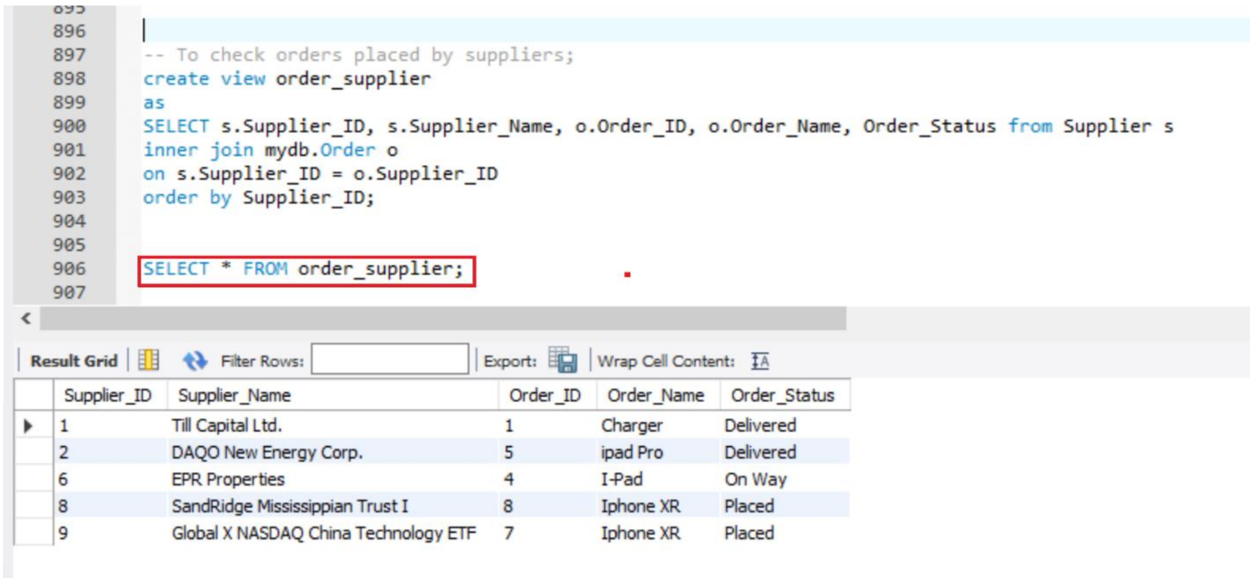
Image 2: VIEW for Employees Working on Products



```
-- To check orders placed by clients;
create view order_client
as
SELECT c.Client_ID, c.Client_Name, o.Order_ID, .o.Order_Name, Order_Statu
inner join mydb.Order o
on c.Client_ID = o.Client_ID
order by Client_ID;
SELECT * FROM order_client;
```

Client_ID	Client_Name	Order_ID	Order_Name	Order_Status
1	Freshpet, Inc.	2	iphone X	On Way
2	Western Asset Municipal High Income Fund, Inc.	3	iphone XR	On Way
5	Washington Prime Group Inc.	10	Iphone XR	Placed
7	Tempur Sealy International, Inc.	6	Iphone XR	Delivered
9	Innovative Solutions and Support, Inc.	9	Iphone XR	Placed

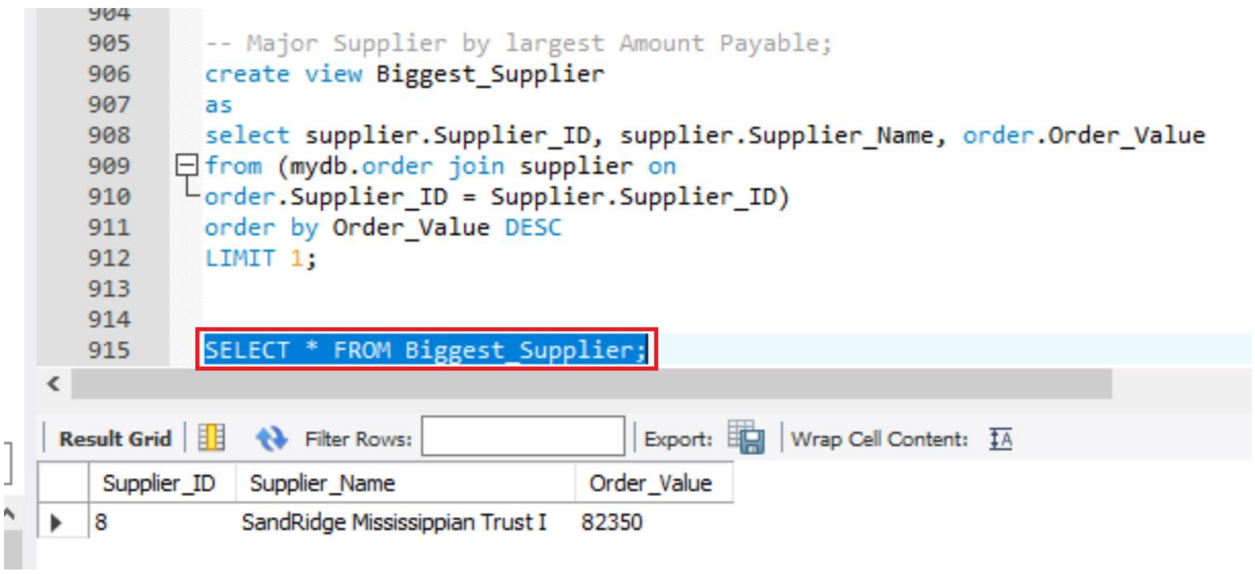
Image 3: VIEW to check orders placed by Clients



```
-- To check orders placed by suppliers;
create view order_supplier
as
SELECT s.Supplier_ID, s.Supplier_Name, o.Order_ID, o.Order_Name, Order_Status from Supplier s
inner join mydb.Order o
on s.Supplier_ID = o.Supplier_ID
order by Supplier_ID;

SELECT * FROM order_supplier;
```

Supplier_ID	Supplier_Name	Order_ID	Order_Name	Order_Status
1	Till Capital Ltd.	1	Charger	Delivered
2	DAQO New Energy Corp.	5	ipad Pro	Delivered
6	EPR Properties	4	I-Pad	On Way
8	SandRidge Mississippian Trust I	8	Iphone XR	Placed
9	Global X NASDAQ China Technology ETF	7	Iphone XR	Placed

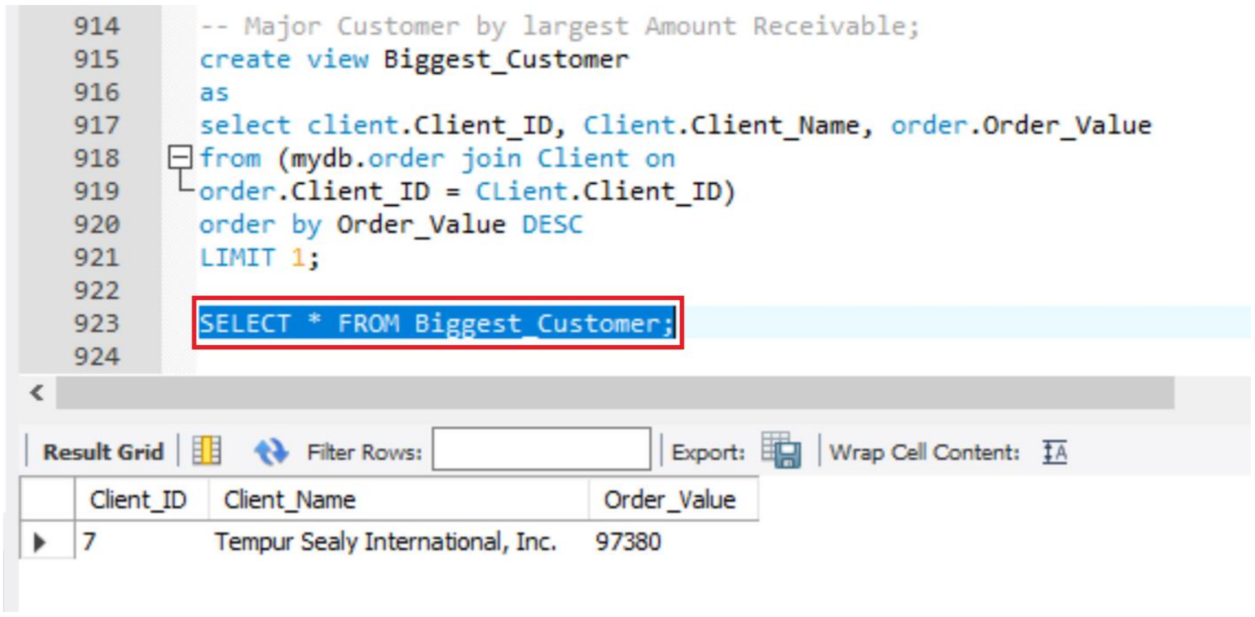
Image 4: VIEW to check orders placed by Suppliers



```
-- Major Supplier by largest Amount Payable;
create view Biggest_Supplier
as
select supplier.Supplier_ID, supplier.Supplier_Name, order.Order_Value
from (mydb.order join supplier on
order.Supplier_ID = Supplier.Supplier_ID)
order by Order_Value DESC
LIMIT 1;


SELECT * FROM Biggest_Supplier;
```

Supplier_ID	Supplier_Name	Order_Value
8	SandRidge Mississippian Trust I	82350

Image 5: VIEW to find the major Supplier by Largest Amount Payable



```
-- Major Customer by largest Amount Receivable;
create view Biggest_Customer
as
select client.Client_ID, Client.Client_Name, order.Order_Value
from (mydb.order join Client on
order.Client_ID = CLient.Client_ID)
order by Order_Value DESC
LIMIT 1;

SELECT * FROM Biggest_Customer;
```

Client_ID	Client_Name	Order_Value
7	Tempur Sealy International, Inc.	97380

Image 6: VIEW to find the Major Customer by Largest Amount Receivable

In the Image 2, the query to make a list of employees working on different products is written. Instead of writing or running the whole query again, the view query, “SELECT * FROM Employee_Working_Product is written” which enables us to get quicker response time from the software to give us the desired results.

Similarly, the queries highlighted in the all the images above run the queries to check orders placed by clients (Image 3), to check orders placed by suppliers (Image 4), to find out the major supplier in terms of largest amount payable (Image 5) and to find out the major customers by terms of largest amount receivables (Image 6).

9. UML Diagrams

a. Use Case Diagram:

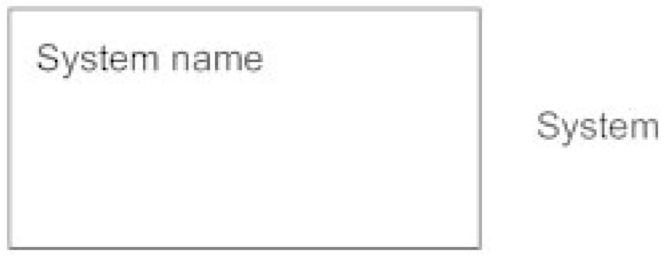
A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.

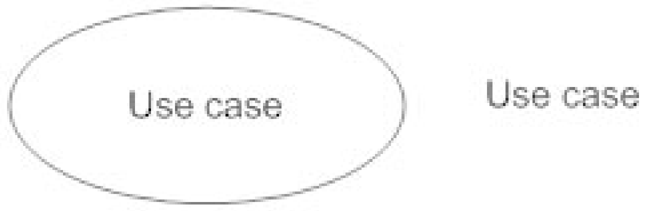
They also help identify any internal or external factors that may influence the system and should be taken into consideration.

They provide a good high-level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented.

System
System's boundaries are drawn using a rectangle that contains use cases. We place actors outside the system's boundaries.



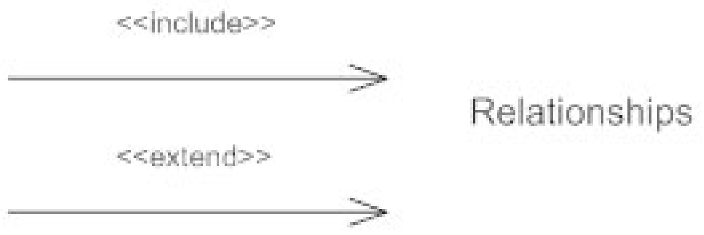
Use Case
Use cases are drawn using ovals. We label the ovals with verbs that represent the system's functions.



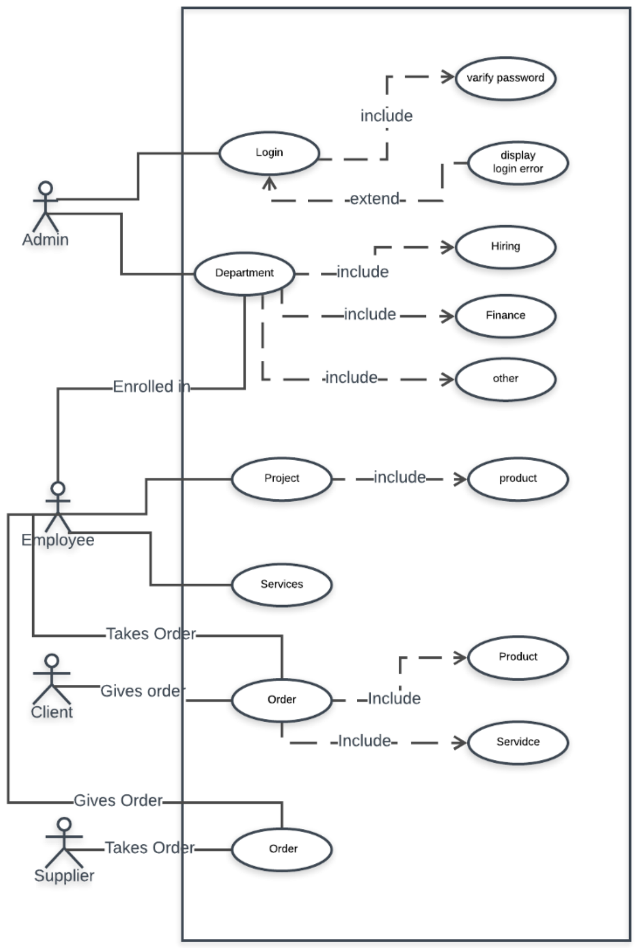
Actors
Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

Relationships

We illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, we use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



Use Case Diagram for Database On The Fly:



In the above use case diagram the actors are Admin, Employee, Client and Supplier and they are linked to a system and their functionality within the system is defined with use cases.

The admin first logs in, include relationships, first the password will be verified, and the user will logged in. Extends show that is a possibility if the admin types in wrong password , he will be shown display login error.

We have created an actor called Employee for the company, and every employee works in a certain department so that's why there is a relationship that the several departments can include Hiring, Finance or any other department in company depending upon the company background or requirements.

When employee works in a department, he either can work in product team or a service team. If he is assigned project, he works in Product team else he works in Service team providing customer service, getting feedback and providing solution to customers.

The employee takes order from client, works on that order and gives the order back to client after it gets completed. The supplier takes order, after it gets completed.

This is the flow of the use case diagram which can be applied to any business organization or company.

a. Activity Diagram:

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

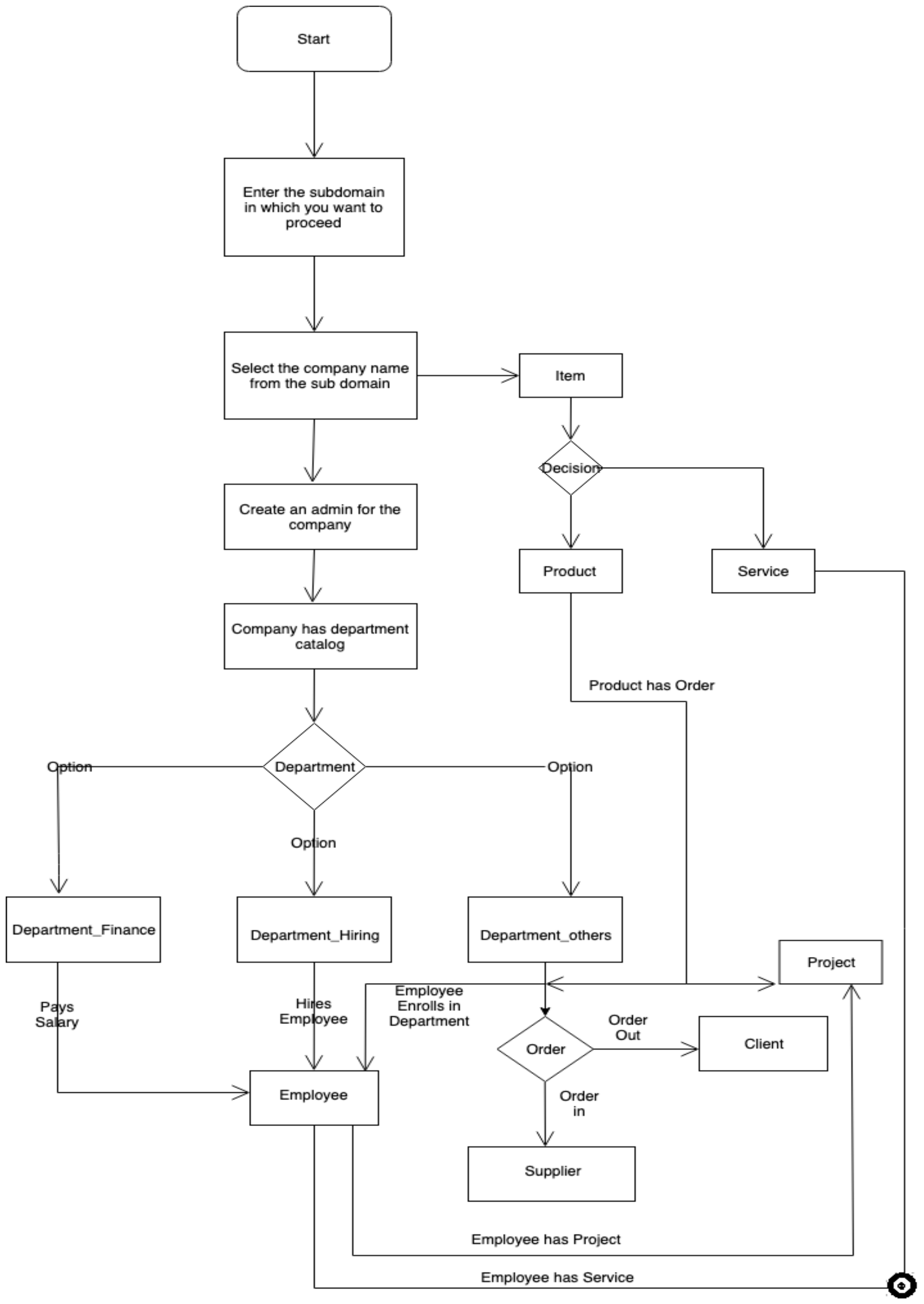
The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

Activity is a operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as −

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.



In this activity diagram, the process starts with the company or organization entering the subdomain in which they want to proceed. Then after selecting the subdomain entering the company name in sub domain.

As each company has different departments , so the company has department catalog in which they can manage all the different departments. The given company can also create an admin for their company who can manage all the database organization. After selecting the company name in subdomain, the given company can pivot into either product or service. As every company can provide either product or service.

The department catalog has different departments like department finance, department hiring and department others which varies upon the company.

The department hiring hires employee. Employee works on project on either service. In the other department, the client gives order and supplier takes order.

And this is the whole process of activity diagram for the project database on the fly where common things in company or organization is taken and schema is developed so it becomes easy for the company to build database without putting more time and efforts.

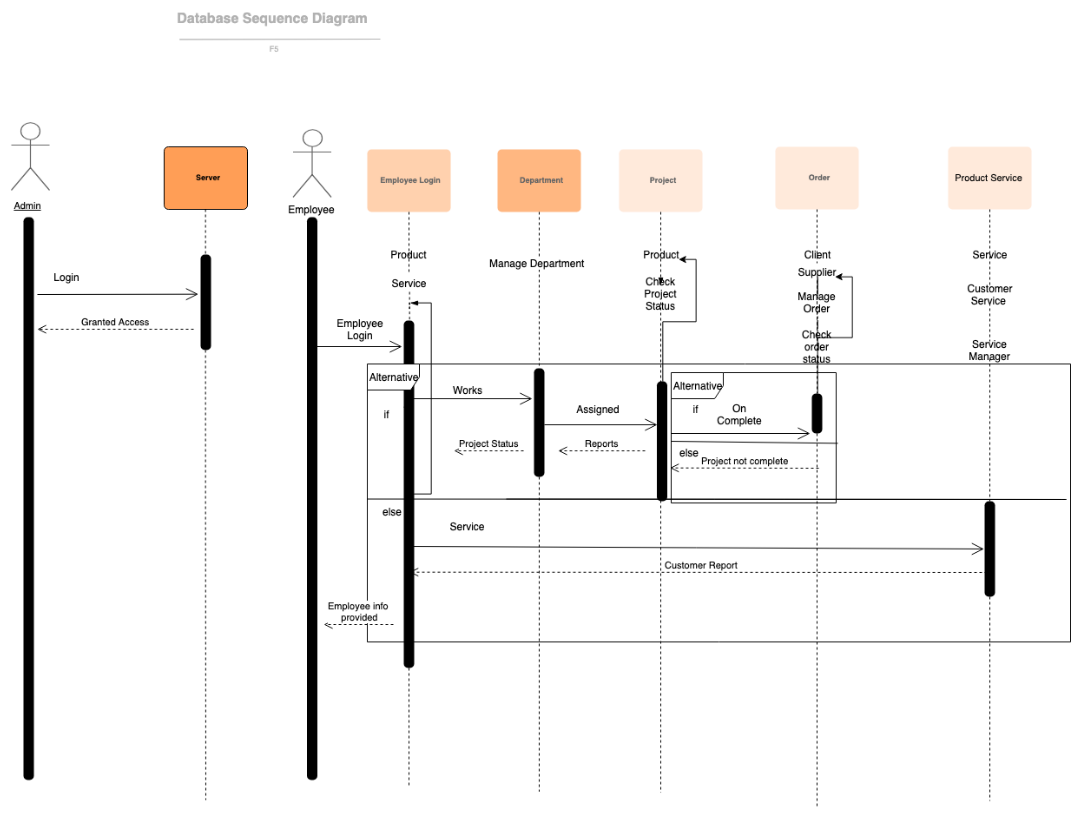
c. Sequence Diagram:

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

Lifelines – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically, each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name: Class Name

Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.



In this sequence diagram first the admin logs into the database and upon valid credentials, the admin he is logged in and can manage the company database.

After that employer logs in the company and checks for the work assigned. The employee either works in Product or service. If the employee works in Product, he is assigned some project and upon completion of the project the order gets completed else the order is incomplete and project needs to work upon. The employee can also work in service where he will get customer report and feedback and use the feedback to improve the service of the company.

This is the sequence or flow of what happens when an employee logs into company portal to working under either product or service upon the work assigned to employee.

10. DCL

DCL (Data Control Language)

DCL It is used to control user access in a database and includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system. These commands are related to the security issues.

Examples of DCL commands:

- GRANT- gives user's access privileges to database.
- REVOKE- withdraw user's access privileges given by using the GRANT command.

Grant - DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owners of the database object can provide/remove privileges on a database object.

Syntax for Grant:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

- privilege_name is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- object_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- user_name is the name of the user to whom an access right is being granted.
- user_name is the name of the user to whom an access right is being granted.
- PUBLIC is used to grant access rights to all users.
- ROLES are a set of privileges grouped together.
- WITH GRANT OPTION - allows a user to grant access rights to other users.

```
-- -----------------------------------------------------------------------------------------
-- Admin User
create user if not exists Admin identified by 'password';
grant all
on mydb.*
to Admin;
show grants for Admin;

-- Owner User
create user if not exists Owners identified by 'password';
grant SELECT
on mydb.*
```

```
to Owners;
show grants for Owners;

-- Employee User
create user if not exists Employee identified by 'password';
grant SELECT(Employee_Name, Employee_Email, Employee_Phone_Number, Role)
on mydb.Employee
to Employee;
```

This command grants a SELECT permission to provide access or privileges on the database objects on employee table to Employee.

```
grant SELECT(Product_Name, Product_Details)
on mydb.Product
to Employee;
```

This command grants a SELECT permission to provide access or privileges on the database objects on Product table to Employee.

```
grant SELECT(Project_Name, Start_Date, End_Date)
on mydb.Project
to Employee;
```

This command grants a SELECT permission to provide access or privileges on the database objects on Project table to Employee.

```
grant SELECT(Supplier_ID, Client_ID, Order_Type, Order_Name, Order_Status)
on mydb.Order
to Employee;
```

This command grants a SELECT permission to provide access or privileges on the database objects on Order table to Employee.

11. Queries

Query to see the Product Details - Within our database, we have different tables containing the data we want to work with. We need to find out which fields are in the tables. Fields are the specific pieces of data that we can pull from our database. For example, if we want to pull product details from table Product, we will write a query. In this query we used join to pull information from two tables namely 'Product' and 'Project'. As we want some specific fields from both the tables, so we used Inner join.
With 'Select' statement, we have written all the fields that we want in out resulting table that included (Prod.Product_Name, Proj.Project_Name, Prod.Product_Price, Prod.Product_Qty, Prod.Product_Details, Proj.Budget). Here Prod= Product and Proj = Project. As we are joining fields from two tables, name of second table 'Product' is written after 'INNER JOIN'. While using join, there must be any common field among both the table. Here Product ID is the reference column for second table, which is shown after 'ON'.

```
-- Query to see the Product Details

SELECT Prod.Product_Name, Proj.Project_Name, Prod.Product_Price, Prod.Product_Qty, Prod.Product_Details, Proj.Budget
FROM Project AS Proj
INNER JOIN Product AS Prod
ON Proj.Product_idProduct = Prod.Product_ID;
```

Query to show products in Warehouse- To pull product information from product table and showing it in Warehouse table, we used Inner join. With 'Select' statement, we have written all the fields that we want in out resulting table that included (warehouse.Store_ID, product.Product_ID, product.Product_Name, product.product_Price, product.Product_Qty, warehouse.Location, warehouse.Storage_Capacity, warehouse.Remaining_Capacity).
As we are joining fields from two tables, name of second table 'warehouse' is written after 'inner join'. While using join, there must be any common field among both the table. Here Product ID is the reference column for second table, which is shown after 'ON'.
To read the information with quickly and easily, output in table is ordered by Store ID.

```
-- Query to show products in Warehouse

SELECT warehouse.Store_ID, product.Product_ID, product.Product_Name, product.product_Price,
product.Product_Qty, warehouse.Location, warehouse.Storage_Capacity, warehouse.Remaining_Capacity
From product
inner join warehouse
on warehouse.Product_ID = product.Product_ID
order by Store_ID;
```

12. Triggers

Trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.

Example –
Suppose, we are adding a tupple to the 'Donors' table that is some person has donated blood. So, we can design a trigger that will automatically add the value of donated blood to the 'Blood_record' table.

Types of Triggers –

We can define 6 types of triggers for each table:

1. **AFTER INSERT** activated after data is inserted into the table.
2. **AFTER UPDATE:** activated after data in the table is modified.
3. **AFTER DELETE:** activated after data is deleted/removed from the table.
4. **BEFORE INSERT:** activated before data is inserted into the table.
5. **BEFORE UPDATE:** activated before data in the table is modified.
6. **BEFORE DELETE:** activated before data is deleted/removed from the table.

Examples showing implementation of Triggers in our project:

1. -- Trigger to Insert into Product or Service after Insert in Product Service Catalog

```
DELIMITER $$

CREATE TRIGGER `mydb`.`Product_Service_Catalog_AFTER_INSERT`

AFTER INSERT

ON `Product_Service_Catalog`

FOR EACH ROW

BEGIN

IF NEW.PS_Type='P' THEN

INSERT INTO Product (`Product_ID`, `Product_Service_Catalog_idProduct_Service_Catalog` , `Product_Name`, `Product_Price`, `Product_Qty`, `Product_Details`)

VALUES (null, NEW.Product_Service_ID, NEW.PS_Name, null, null, null);

ELSE

INSERT INTO Service (`Service_ID`, `Product_Service_Catalog_idProduct_Service_Catalog` , `Service_Name`, `Service_Charge`)

VALUES (null, NEW.Product_Service_ID, NEW.PS_Name, null);

END IF;

END;

$$
```

- Here, we have created a trigger that will Insert into the Table Product or Service when there is insertion in the table Product_Service_Catalog
- The trigger will insert values into the tables Product or Service depending on the condition whether the type of insertion in Product_Service_Catalog table is 'P' or 'S'

- If it is P then it will insert the value in Product Table, else it will Insert the value in Service Table
- The Delimiter '$$' is used to change the delimiter to '$$' so that we can use ';' in our trigger without SQL thinking it is the end of the line.
- Create Trigger line is used to Create a Trigger with the name followed by it, here the name of our trigger is *`mydb`.`Product_Service_Catalog_AFTER_INSERT`*
- After Insert specifies the condition of the trigger which is pre-defined to be one of the 6 as shown above
- On specifies on which table we have to perform trigger, so after the condition mentioned above occurs we would perform our trigger task. So here, after insertion occurs '*ON*' Product_Service_Catalog the condition of trigger will occur.
- FOR EACH ROW specifies that we need the trigger to run for all the rows and not selected rows.
- BEGIN, here our logic of trigger is written.
- We have applied an If loop to check if the PS_Type is P or not, so if it is P then we will specify which table it must insert the values into (i.e. Product Table)
- Else it will insert values into the other table (i.e. Service Table)
- END statements end the if loop and the Begin of trigger

2-- Trigger to Insert Employee after Insert in Hiring

```
DELIMITER $$

CREATE TRIGGER `mydb`.`Employee_AFTER_INSERT`

AFTER INSERT

ON `Hiring`

FOR EACH ROW

BEGIN

INSERT INTO Employee (Employee_ID, Department_Hiring_idDepartment_Hiring, Employee_Name, Department_ID,
Employee_Email, Employee_Address, Employee_Phone_Number, Role)

VALUES (null, NEW.Hiring_ID, null, 5, null, null, null, null);

END;

$$
```

- Here, we have created a trigger that will Insert into the Table Employee when there is insertion in the table Hiring.
- The trigger will check whether there is insertion in Hiring, if there is it will insert the same Employee in Employee Table.

13. Procedures

PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements.

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc.
All the statements of a block are passed to Oracle engine all at once which increases processing speed and decreases the traffic.

A stored procedure is a prepared SQL code that you can save, so the code can be reused repeatedly. So, if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Advantages:

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:

- Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.
- MySQL does not provide the functionality of debugging the stored procedures.

Examples showing implementation of Stored Procedures in our project:

```
DROP PROCEDURE IF EXISTS Cellphone

DELIMITER //

CREATE PROCEDURE Cellphone()

BEGIN

CREATE TABLE Research_And_Development(

Development_ID INT(10) NOT NULL PRIMARY KEY AUTO_INCREMENT,

Department_Catalog_idDepartment_Catalog INT(10) NOT NULL DEFAULT 1,

Project_ID INT(10) NOT NULL ,

Start_Date DATE ,

Details TEXT ,

Expected_Duration_Months INT(3),

FOREIGN KEY (Project_ID) REFERENCES Project(Project_ID),

FOREIGN KEY (Department_Catalog_idDepartment_Catalog) REFERENCES mydb.Department_Catalog (Department_ID) );
```

```
…
ENGINE=InnoDB DEFAULT CHARSET=utf8;
END;
//
```

- Here we have created a procedure to generate tables in the database.
- We will call these procedures when we need to add those tables in the database.
- We have defined procedures in such a way that they contain tables for different sub-domains.
- So a procedure *'Cellphone()'* will add tables that are related to the mobile-phones in our database and so on.
- Drop Procedure if exists is pretty much self-explanatory line, it states to drop a procedure named cellphone if it already exists
- The Delimiter '//' is used to change the delimiter to '//' so that we can use ';' in our procedure without SQL thinking it is the end of the line.
- Create Procedure creates a procedure with the name 'Cellphone()'
- Begin marks the beginning of the logic of our procedure
- We have then written create table syntax for creating 3 distinct departments in our database viz. Research_And_Development, Warehouse and Production
- End marks the end of the logic of our procedure.

14. Database Backups

A **backup** is a copy of data from your database that can be used to reconstruct that data.

Three common types of database backups can be run on a desired system: **normal (full), incremental and differential**. Each type has advantages and disadvantages, but multiple database backup approaches can be used together to design a comprehensive server backup and recovery strategy. A customized backup plan can **minimize downtime** and **maximize efficiency**.

Normal or Full Backups

When a normal or full backup runs on a selected drive, all the files on that drive are backed up. This, of course, includes system files, application files, user data — everything. Those files are then copied to the selected destination (backup tapes, a secondary drive or the cloud), and all the archive bits are then cleared.

Normal backups are the fastest source to restore lost data because all the data on a drive is saved in one location. The downside of normal backups is that they take a very long time to run, and in some cases this is more time than a company can allow. Drives that hold a lot of data may not be capable of a full backup, even if they run overnight. In these cases, incremental and differential backups can be added to the backup schedule to save time.

Incremental Backups

A common way to deal with the long running times required for full backups is to run them only on weekends. Many businesses then run incremental backups throughout the week since they take far less time. An incremental backup will grab **only the files that have been updated since the last normal backup**. Once the incremental backup has run, that file will not be backed up again unless it changes or during the next full backup.

While incremental database backups do run faster, the recovery process is a bit more complicated. If the normal backup runs on Saturday and a file is then updated Monday morning, should something happen to that file on Tuesday, one would need to access the Monday night backup to restore it.

Differential Backups

An alternative to incremental database backups that has a less complicated restore process is a differential backup. Differential backups and recovery are similar to incremental in that these backups grab only files that have been updated since the last normal backup. However, differential backups do not clear the archive bit. So a file that is updated after a normal backup will be archived every time a differential backup is run until the next normal backup runs and clears the archive bit.

Similar to our last example, if a normal backup runs on Saturday night and a file gets changed on Monday, that file would then be backed up when the differential backup runs Monday night. Since the archive bit will not be cleared, even with no changes, that file will continue to be copied on the Tuesday night differential backup and the Wednesday night differential backup and every additional night until a normal backup runs again capturing all the drive's files and resetting the archive bit.

Backup schemes for Small and Large Companies

Small Company:

For a small company with small team size with a peak time on weekdays at 8:00 am to 10 :00 pm, and on weekends at 11 :00 am to 8 :00 pm.

From this we can come to know that company has 10 non-peak hours on weekdays and 15 non peak hours on weekends.

Day	Backup Type	Time	Hours
Monday	Increment From Sat	1:00 am	1
Tuesday	Increment From Sat	1:00 am	1.5
Wednesday	Full Backup	12:00 am	3
Thursday	Increment From Wed	1:00 am	0.5
Friday	Increment From Wed	1:00 am	1
Saturday	Full Backup	10:00 pm	3
Sunday	Increment From Sat	12:00 am	0.5

Large Company:

For a large company with 5k plus staff members and has multiple branches.

Peak hours on weekday at 6:00 am to 11: 00 pm (7 non peak hours)

Peak hours on weekend at 9:00 am to 10: 00 pm (11 non peak hours)

Then we can assume full backup takes 10 hrs, incremental backup takes 1 hrs.

Day	Backup Type	Time	Hours
Monday	Increment From Sat	1:00 am	2
Tuesday	Increment From Sat	1:00 am	3
Wednesday	Full Backup	1:00 am	4
Thursday	Increment From Wed	1:00 am	1
Friday	Increment From Wed	1:00 am	2
Saturday	Full Backup	10:00 pm	10
Sunday	Increment From Sat	12:00 am	1

We implemented the backup with the help of triggers wherein if any row gets deleted , its copy gets generated in backup table and person can retrieve the data from the backup table without data getting lost , thus with the use of trigger one can generate backup and if case data gets deleted one can restore the data and continue on with work .Thus reducing efforts and saving time with saving data in backup table.

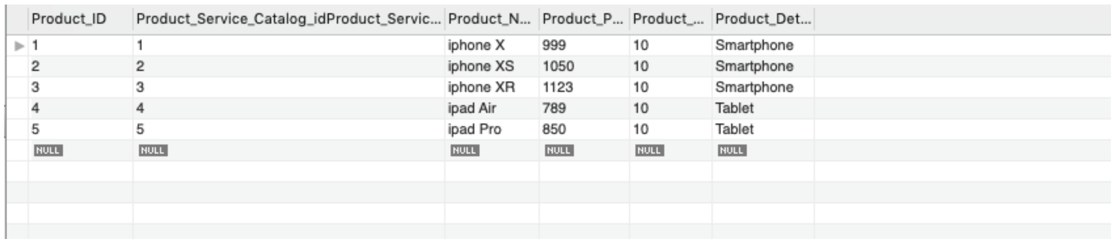
SQL Code:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Product_Backup` (
  `Product_ID` INT(10) NOT NULL AUTO_INCREMENT,
  `Product_Service_Catalog_idProduct_Service_Catalog` INT(10) NULL,
  `Product_Name` VARCHAR(45) NULL,
  `Product_Price` INT(10) NULL,
  `Product_Qty` INT(10) NULL,
  `Product_Details` VARCHAR(100) NULL,
  `Timestamp` datetime(6) Null,
  PRIMARY KEY (Product_ID));
________________________________
DELIMITER $$
CREATE TRIGGER tr_products_backup
After DELETE
ON Product
FOR EACH ROW BEGIN
INSERT INTO Product_Backup
 VALUES (OLD.Product_ID, OLD.Product_Service_Catalog_idProduct_Service_Catalog,
 OLD.Product_Name, OLD.Product_Price, OLD.Product_Qty, OLD.Product_Details,SYSDATE());
END;
```

delete from Product where Product_ID = 1 ;

select * from Product_Backup;

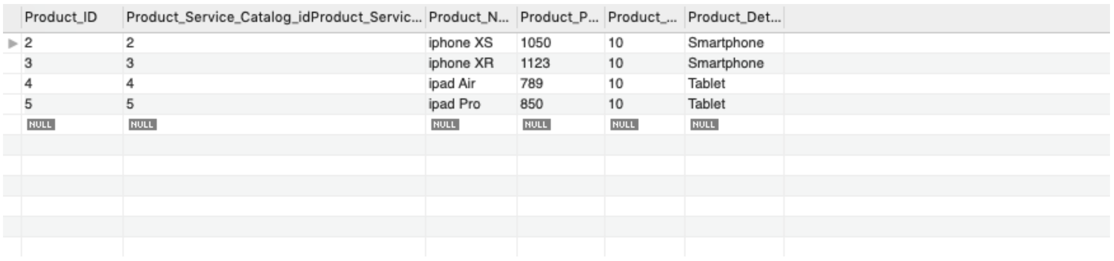
Before Backup:



	Product_ID	Product_Service_Catalog_idProduct_Servic...	Product_N...	Product_P...	Product_...	Product_Det...
▶	1	1	iphone X	999	10	Smartphone
	2	2	iphone XS	1050	10	Smartphone
	3	3	iphone XR	1123	10	Smartphone
	4	4	ipad Air	789	10	Tablet
	5	5	ipad Pro	850	10	Tablet
	NULL	NULL	NULL	NULL	NULL	NULL

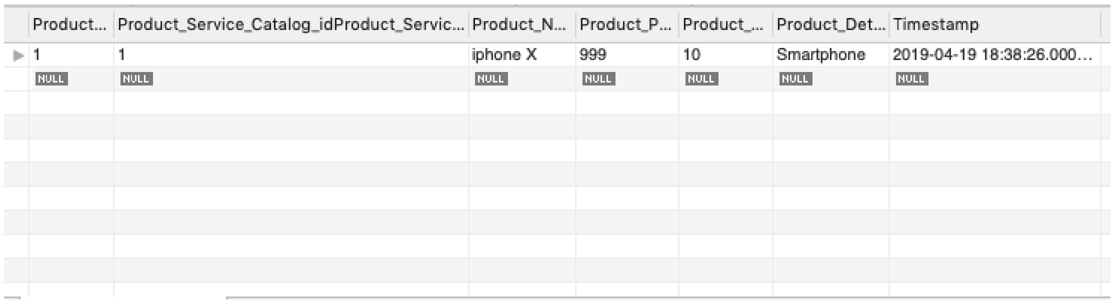
After Backup:

select * from Product



	Product_ID	Product_Service_Catalog_idProduct_Servic...	Product_N...	Product_P...	Product_...	Product_Det...
▶	2	2	iphone XS	1050	10	Smartphone
	3	3	iphone XR	1123	10	Smartphone
	4	4	ipad Air	789	10	Tablet
	5	5	ipad Pro	850	10	Tablet
	NULL	NULL	NULL	NULL	NULL	NULL

Select * from Product_Backup;



	Product...	Product_Service_Catalog_idProduct_Servic...	Product_N...	Product_P...	Product_...	Product_Det...	Timestamp
▶	1	1	iphone X	999	10	Smartphone	2019-04-19 18:38:26.000...
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

15. Conclusions and Future Scope

Our aim for the project Database on The Fly was to provide schema and existing database to companies or organization looking to build data, saving time, efforts and money into building new database from scratch. And we were successful in building the project, though there is some limitations and assumptions as how this project can be implemented and used. But its usage will surely reduce the time efforts for large companies, or any small-scale companies who is just starting and do not want to spend that much for database. By providing the skeleton or existing schema part, the company can leverage on that and can build more complex database, thus helping organization build database faster and easier.

There are some advantages to using our Database project for building like no pre requisite knowledge of database for building database. No expert domain is needed, existing schema will provide all the functionalities to what the client needs. And the database schema is automated, the client just have to input the values , rest the structure and schema for the building of database will be provided. Thus with providing minimum inputs , the client can achieve a fully functionally working database from scratch without putting in any efforts and time into building database.

The project has got wide scope and we will plan to build more features and add extra functionality, so it becomes whole lot easier for any company or organization to use our database schema for project. But with existing features provided, it’s a seamless experience for client or user to just provide input and build database on the fly.

16. Reference Links

- https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html
- https://www.mockaroo.com/
- https://stackoverflow.com/questions/440308/tsql-returning-a-table-from-a-function-or-store-procedure
- https://stackoverflow.com/questions/23382499/run-python-script-on-database-event
- http://stratosprovatopoulos.com/web-development/mysql/pivot-a-table-in-mysql/
- https://data.stackexchange.com/stackoverflow/query/497432
- https://stackoverflow.com/questions/15745042/efficiently-convert-rows-to-columns-in-sql-server
- https://www.w3schools.com/sql/sql_stored_procedures.asp
- https://www.geeksforgeeks.org/sql-trigger-student-database/
- **https://www.essentialsql.com/what-is-a-relational-database-view/**