Application Development

Course Content

- GIT & GitHub
  - ➢ Version Control System
  - ➢ Git & Azure
  - ➢ Key Concepts
  - ➢ Git Commands
- Database Management System (DBMS)
  - ➢ DBMS Definition
  - ➢ Relational Data Model
- Learning C#/ASP.Net Core

Azure & Git Repository

Version Control

Version Control System

Version Control System (VCS) A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes, document development (for other developers and yourself, not for users)

Version Control System

Why Version Control?

- Combine work of multiple collaborators
- Understand changes
- Support incremental development
- Compare and revert to earlier versions
- Backup
- Parallel versions
- Document development (for other developers and yourself, not for users)

Version Control System

Types of VCS

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project.

Version Control System

Distributed/Decentralized version control system (DVCS) clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the sever goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository.

Azure/Git repository does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.

Azure & Git Repository

Git

Azure

Git & Azure

Git is a free and open-source distributed version control system, it is used for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

GIT

Git & Azure

Azure Repository is a set of version control tools that we can use to manage our code. There are so many software that is available in the market to enable version control on our code. We can use the version control system to keep track of each change done by each developer, safely merge them, test the changes, and publish the change into production.

There are two types of version control in Azure Repos.

- **Git:** It is a distributed version control.
- **Team Foundation Version Control:** It is a centralized version control.

Azure

Key Concepts

Key Concepts – Snapshots

- The way git keeps track of your code history.
- Essentially records what all your files look like at a given point in time.
- You decide when to take a snapshot, and of what files.
- Have the ability to go back to visit any snapshot.
- Your snapshots from later on will stay around, too.

Key Concepts – Repositories

A repository is a location for our code, which is managed by version control. It supports Git and TFVC so we can create multiple repositories in a single project and various branches for each repository.

- Often shortened to ‘repo’
- A collection of all the files and the history of those files
- Consists of all your commits
- Place where all your work is stored
- Can live on a local machine or on a remote server (GitHub!)

Key Concepts – Repositories

A repository is a location for our code, which is managed by version control. It supports Git and TFVC so we can create multiple repositories in a single project and various branches for each repository.

- Often shortened to ‘repo’
- A collection of all the files and the history of those files
- Consists of all your commits
- Place where all your work is stored
- Can live on a local machine or on a remote server (GitHub!)
- The act of copying a repository from a remote server is called cloning
- Cloning from a remote server allows teams to work together
- The process of downloading commits that don’t exist on your machine from a remote repository is called pulling changes.
- The process of adding your local changes to the remote repository is called pushing changes

Key Concepts – Commit

A commit is a group of change saved to our local repository. We can share these changes to the remote repository by pushing.

- The act of creating a snapshot
- Essentially, a project is made up of a bunch of commits
- Commits contain three pieces of information:
    1. Information about how the files changed from previously
    2. A reference to the commit that came before it, called the “parent commit”
    3. A hash code name. Will look something like: f2d2ec5069fc6776c80b3ad6b7cbde3cade4e

Key Concepts – Branches

A branch is a lightweight reference that keeps a history of commits and provides a way to isolate changes for a feature or a bug fix from our master branch and other work.

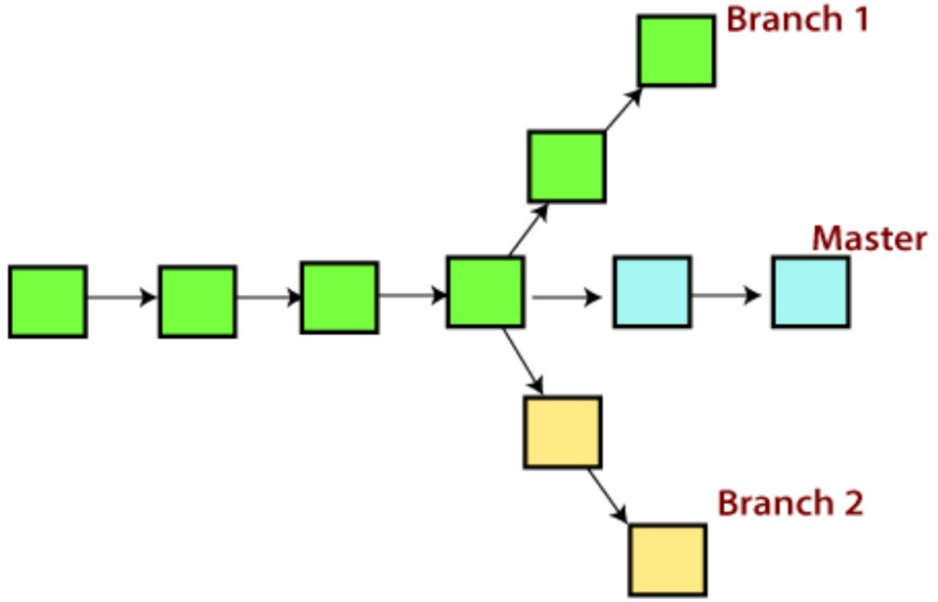
- All commits in git live on some branch.
- But there can be many, many branches.
- The main branch in a project is called the master branch.

So, what is MASTER?

- The main branch in your project.
- Doesn’t have to be called master, but almost always is!

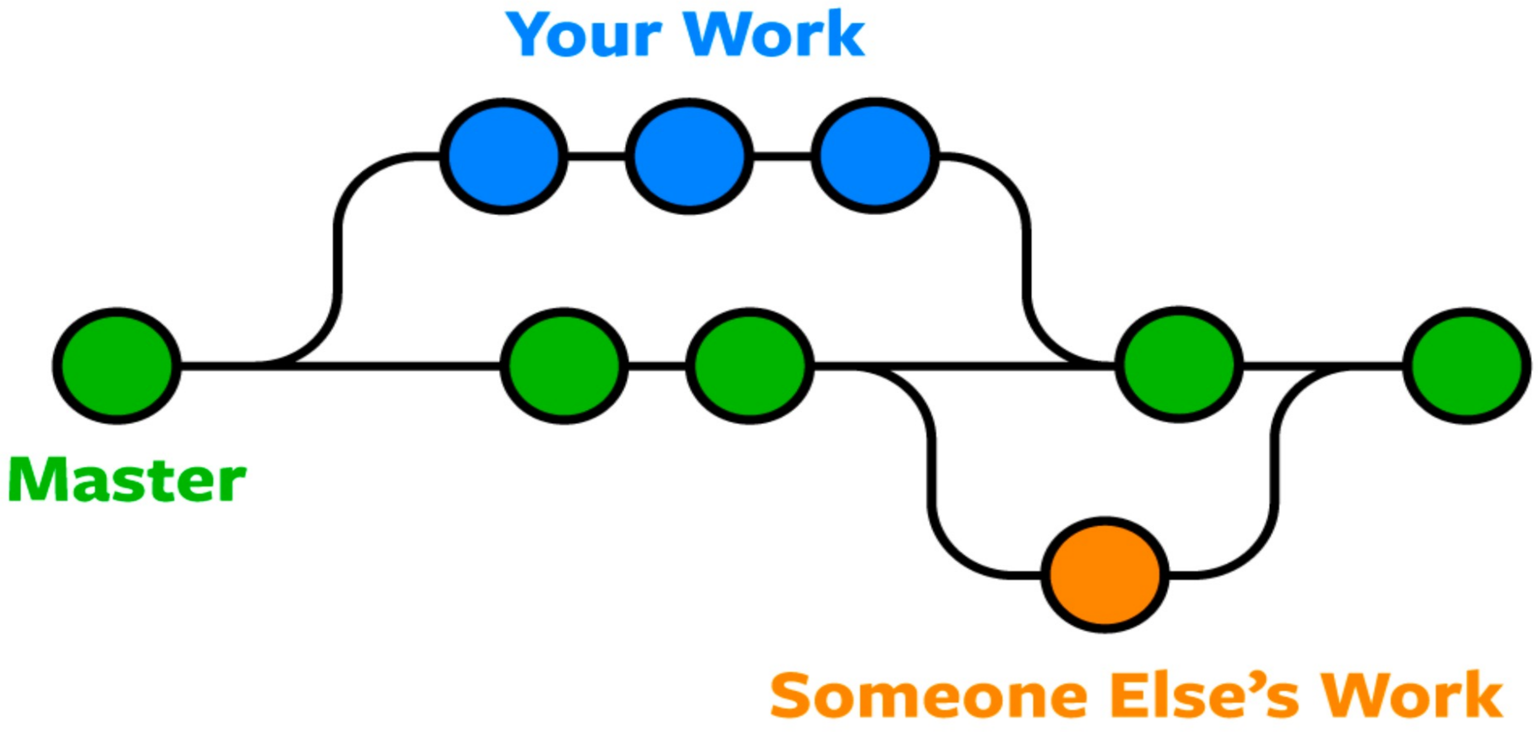
Key Concepts – Branching off of the Master Branch

- The start of a branch points to a specific commit.
- When you want to make any changes to your project you make a new branch based on a commit.



Key Concepts – Merging

- Once you're done with your feature, you merge it back into master



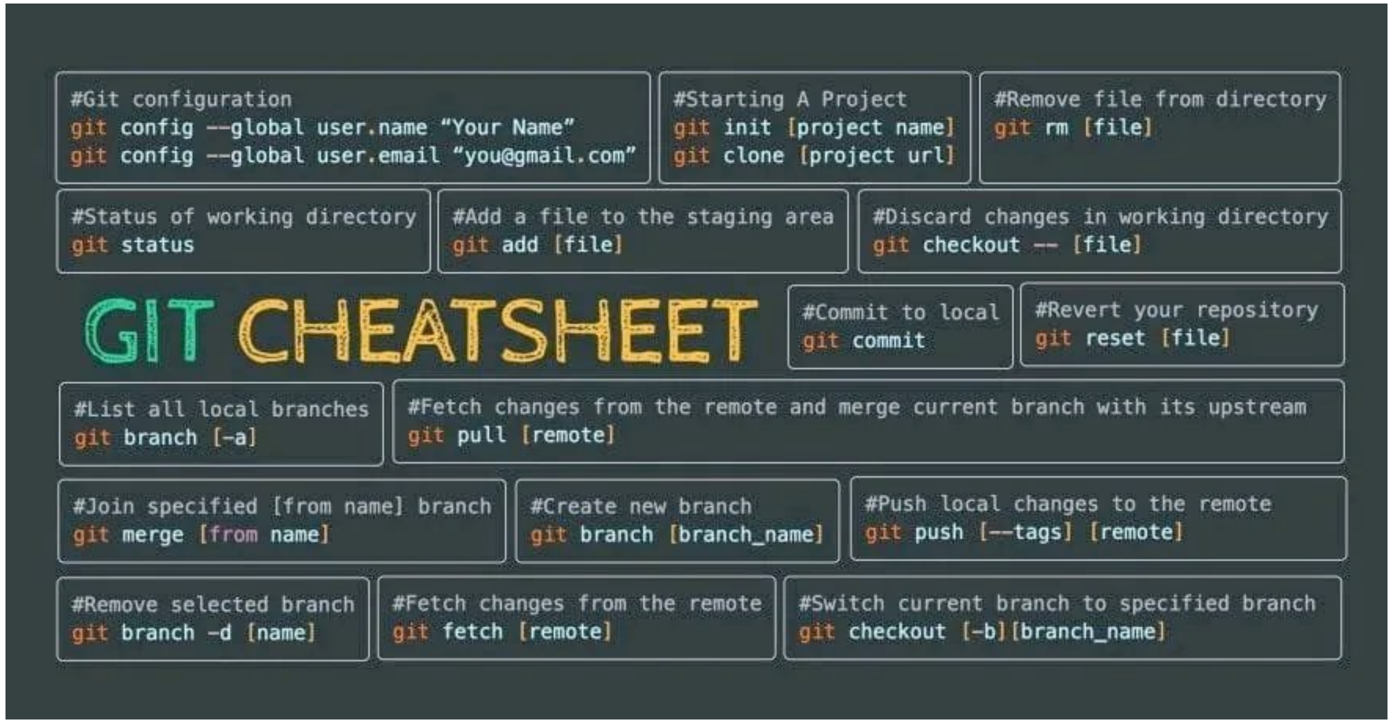
Key Concepts – How do you make a commit?

- There are a lot of ‘states’ and ‘places’ a file can be.
- Local on your computer: the ‘working directory’.
- When a file is ready to be put in a commit you add it onto the ‘index’ or ‘staging’.
- Staging is the new preferred term – but you can see both ‘index’ and ‘staging’ being used.

The process:

- Make some changes to a file
- Use the ‘git add’ command to put the file onto the staging environment
- Use the ‘git commit’ command to create a new commit

Git Commands


GIT CHEATSHEET

```
#Git configuration
git config --global user.name "Your Name"
git config --global user.email "you@gmail.com"

#Starting A Project
git init [project name]
git clone [project url]

#Remove file from directory
git rm [file]

#Status of working directory
git status

#Add a file to the staging area
git add [file]

#Discard changes in working directory
git checkout -- [file]

#Commit to local
git commit

#Revert your repository
git reset [file]

#List all local branches
git branch [-a]

#Fetch changes from the remote and merge current branch with its upstream
git pull [remote]

#Join specified [from name] branch
git merge [from name]

#Create new branch
git branch [branch_name]

#Push local changes to the remote
git push [--tags] [remote]

#Remove selected branch
git branch -d [name]

#Fetch changes from the remote
git fetch [remote]

#Switch current branch to specified branch
git checkout [-b][branch_name]
```

Git Commands

git init

This command is used to create a local repository.

```
$ git init
```

git clone

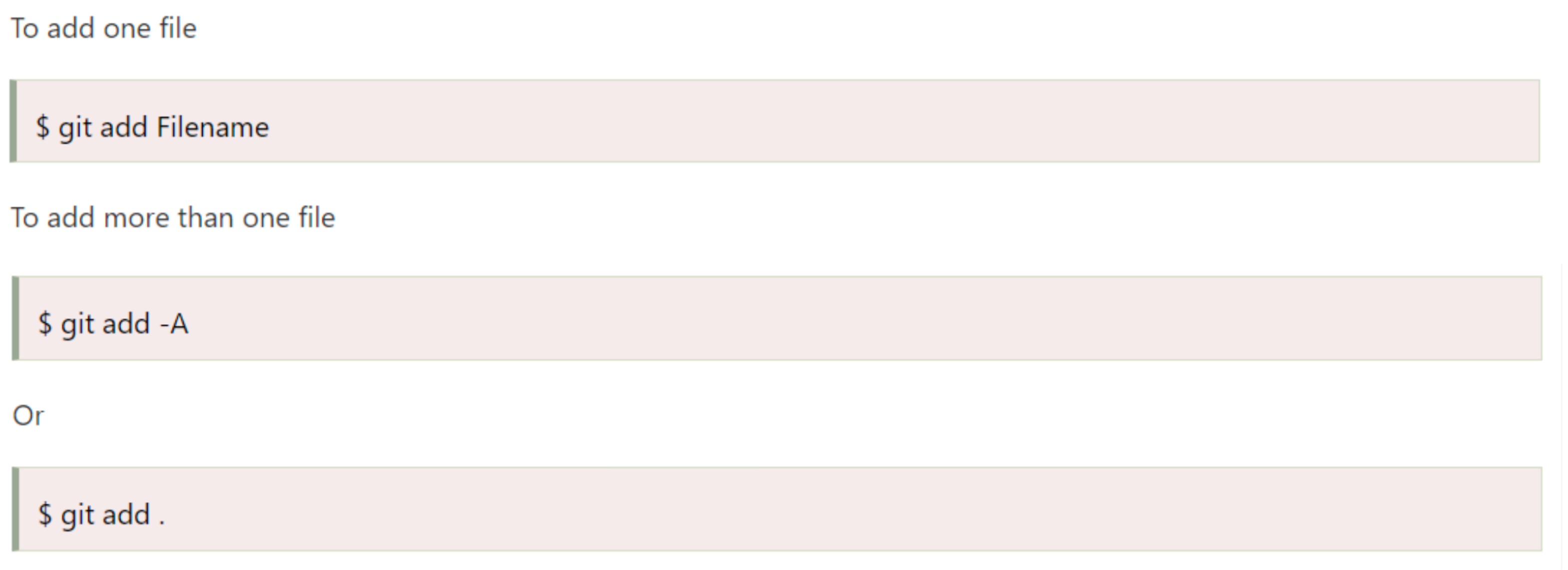
This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

```
$ git clone <repository URL>
```

Git Commands

git add

This command is used to add one or more files to staging (Index) area.



To add one file

```
$ git add Filename
```

To add more than one file

```
$ git add -A
```

Or

```
$ git add .
```

Git Commands

git commit

Commit command is used in two scenarios. They are as follows.

The -m option of commit command lets you to write the commit message on the command line. This command will not prompt the text editor.

```
$ git commit -m " Commit Message"
```

git pull

Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory

```
$ git pull URL
```

Git Commands

git push

It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

This command sends the changes made on the master branch, to your remote repository.

```
$ git push [variable name] master
```

This command pushes all the branches to the server repository.

```
$ git push --all
```

Git Commands

git status

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren’t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

```
$ git status
```

git log

This command is used to check the commit history.

```
$ git log
```

Git Commands

git fetch

Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.

```
$ git fetch <branch URL><branch name>
```

git rebase

To accept the changes, use the rebase command. It will be used as follows:

```
$ git rebase --continue
```

Git Commands

git branch

This command lists all the branches available in the repository.

```
$ git branch
```

You can create a new branch with the help of the git branch command.

```
$ git branch  <branch name>
```

Git allows you to switch between the branches without making a commit.

```
$ git checkout <branchname>
```

Git Commands

git branch

You can switch to the master branch from any other branch with the help of below command.

```
$ git branch -m master
```

To rename a branch, use the below command

```
$ git branch -m <old branch name><new branch name>
```

Git allows you to merge the other branch with the currently active branch.

```
$ git merge <branch name>
```

Git Commands

git merge

This command is used to merge the specified branch's history into the current branch.

```
$ git merge BranchName
```

To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict. The merge command is used as follows:

```
$ git mergetool
```

To resolve the conflict, enter in the insert mode by merely pressing I key and make changes as you want. Press the Esc key, to come out from insert mode. Type the: w! at the bottom of the editor to save and exit the changes.

Version Control System

Version Control System (VCS) A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes, document development (for other developers and yourself, not for users)

Database Management

DBMS (Database Management System)

Database Management System (DBMS) is software for storing and retrieving users’ data while considering appropriate security measures. It consists of a group of programs that manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software store and retrieve data.

DBMS allows users to create their own databases as per their requirements. The term “DBMS” includes the user of the database and other application programs. It provides an interface between the data and the software application.

Relational Data Model in DBMS

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts in DBMS

- **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
- **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
- **Tuple** – It is nothing but a single row of a table, which contains a single record.
- **Relation Schema:** A relation schema represents the name of the relation with its attributes.
- **Cardinality:** Total number of rows present in the Table.
- **Column:** The column represents the set of values for a specific attribute.
- **Relation key** – Every row has one, two or multiple attributes, which is called relation key.
- **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Relational Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Key Constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

must exist in the table.

Relational Integrity Constraints

Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

DBMS Keys

KEYS in DBMS is an attribute or set of attributes which helps you to identify a row (tuple) in a relation (table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

Employee ID	FirstName	LastName
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

In the above-given example, employee ID is a primary key because it uniquely identifies an employee record. In this table, no other employee can have the same employee ID.

Types of Keys in DBMS

- **Primary Key –** s a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key. In the example below, StudID is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

- **Alternate Key –** is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key. In table above, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

Types of Keys in DBMS

- **Foreign Key –** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

DeptCode	DeptName
1	Science
2	English
5	Computer

Teacher ID	DeptCode	Fname	Lname
B002	2	David	Warner
B017	2	Sara	Joseph
B009	1	Mike	Brunton

In this table, adding the foreign key, Deptcode to the Teacher name, we can create a relationship between the two tables. This concept is also known as Referential Integrity.

- **Surrogate Key –** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.

Types of Keys in DBMS

- **Composite Key –** is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

OrderNo	PorductID	Product Name	Quantity
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

Entity Relationship (ER) Diagram Model

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

ER Model stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them. Creating an ER Model in DBMS is considered as a best practice before implementing your database.

Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

One-to-One Relationships: One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.

One-to-Many Relationships: One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

For example, one class is consisting of multiple students.

Cardinality

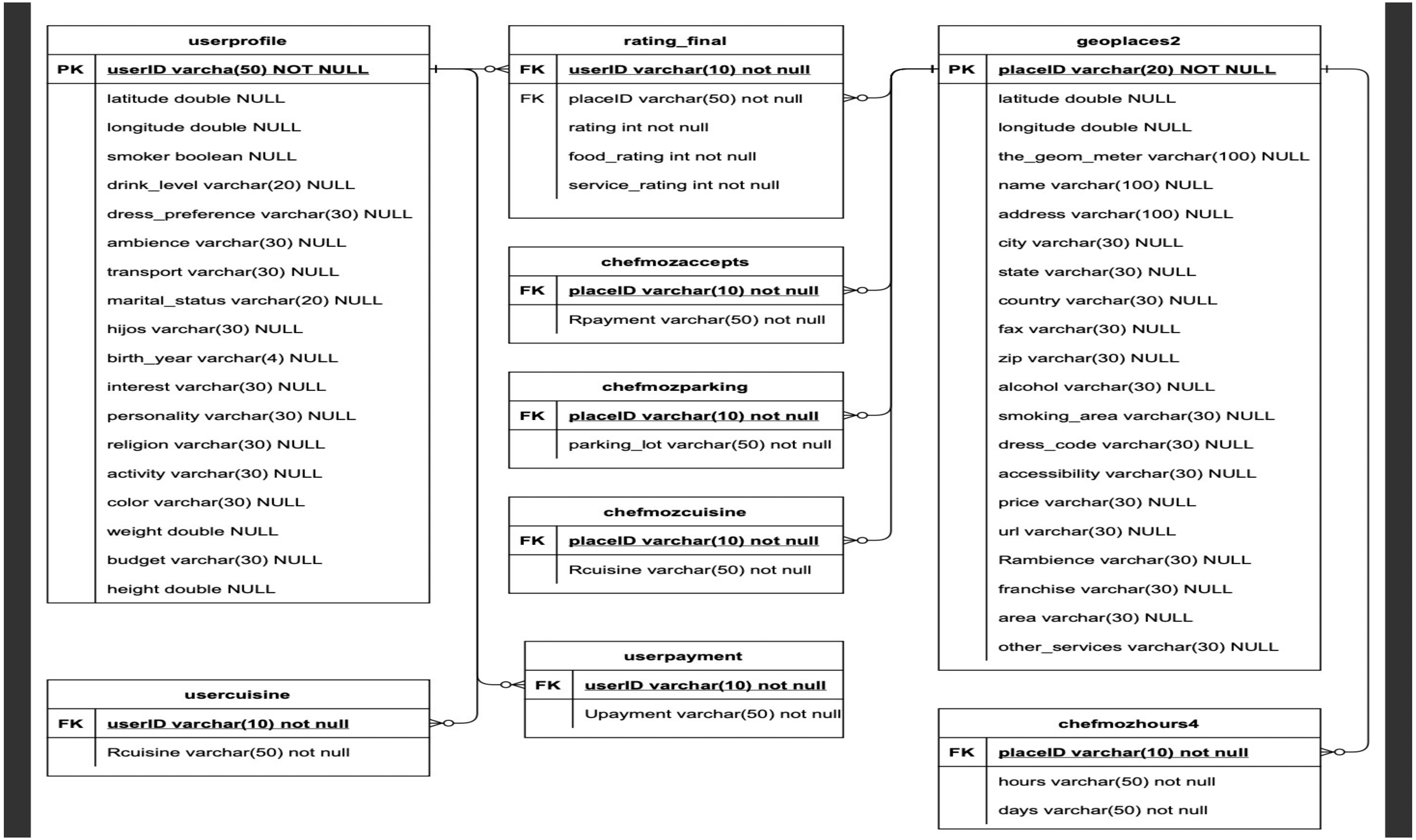
Many to One Relationships: More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

For example, many students belong to the same class.

Many-to-Many Relationships: One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.

Entity Relationship (ER) Diagram Model



userprofile	
PK	userID varcha(50) NOT NULL
	latitude double NULL
	longitude double NULL
	smoker boolean NULL
	drink_level varchar(20) NULL
	dress_preference varchar(30) NULL
	ambience varchar(30) NULL
	transport varchar(30) NULL
	marital_status varchar(20) NULL
	hijos varchar(30) NULL
	birth_year varchar(4) NULL
	interest varchar(30) NULL
	personality varchar(30) NULL
	religion varchar(30) NULL
	activity varchar(30) NULL
	color varchar(30) NULL
	weight double NULL
	budget varchar(30) NULL
	height double NULL

usercuisine	
FK	userID varchar(10) not null
	Rcuisine varchar(50) not null

rating_final	
FK	userID varchar(10) not null
FK	placeID varchar(50) not null
	rating int not null
	food_rating int not null
	service_rating int not null

chefmozaccepts	
FK	placeID varchar(10) not null
	Rpayment varchar(50) not null

chefmozparking	
FK	placeID varchar(10) not null
	parking_lot varchar(50) not null

chefmozcuisine	
FK	placeID varchar(10) not null
	Rcuisine varchar(50) not null

userpayment	
FK	userID varchar(10) not null
	Upayment varchar(50) not null

geoplaces2	
PK	placeID varchar(20) NOT NULL
	latitude double NULL
	longitude double NULL
	the_geom_meter varchar(100) NULL
	name varchar(100) NULL
	address varchar(100) NULL
	city varchar(30) NULL
	state varchar(30) NULL
	country varchar(30) NULL
	fax varchar(30) NULL
	zip varchar(30) NULL
	alcohol varchar(30) NULL
	smoking_area varchar(30) NULL
	dress_code varchar(30) NULL
	accessibility varchar(30) NULL
	price varchar(30) NULL
	url varchar(30) NULL
	Rambience varchar(30) NULL
	franchise varchar(30) NULL
	area varchar(30) NULL
	other_services varchar(30) NULL

chefmozhours4	
FK	placeID varchar(10) not null
	hours varchar(50) not null
	days varchar(50) not null