Assignment 1:

- Explain with examples the difference between a traditional file system and a Database Management System (DBMS).
- Also discuss the concepts of Data Abstraction and Independence.
- Also mention the types of SQL commands and define them with examples and use cases.
- Use the following command: Create, Use, Drop, Select (from), Insert (into) & Use constraint with each data type.

Name: **Himanshu Neeraj**

Bachelor's of Science in Artificial Intelligence
(Semester-01)

Difference between Traditional File System and DBMS

Aspect	Traditional File System	Database Management System (DBMS)
Definition	A traditional file system is a method for organizing, storing, and retrieving data on a storage medium like a hard disk or SSD, abstracting the physical hardware into a logical structure for users and applications.	A Database Management System (DBMS) is a software system that manages the storage, retrieval, and manipulation of data in a structured manner, acting as an interface between users, applications, and the database itself.
Storage of Data	Data is stored in separate files and folders. Each file works independently and does not have a structured format.	Data is stored in an organized manner using tables with rows and columns, which makes storage and management much easier.
Data Redundancy	High redundancy - the same data may be repeated in multiple files. This leads to wastage of storage space and possible data inconsistency.	Redundancy is reduced because DBMS uses normalization techniques, ensuring the same data is not stored multiple times.
Data Linking	No direct linking between files. If a relationship is required, it has to be handled manually by the programmer.	Tables can be linked easily using primary keys and foreign keys, making relationships between data clear and manageable.
Data Security	Very limited security - only basic file-level permissions (like read, write, or delete) are available.	Strong security features are provided, including authentication, authorization, and user-role management.
Querying & Retrieval	Searching and retrieving data is slow and requires custom code or manual scanning through files.	Querying is easy and fast using SQL (Structured Query Language), which makes retrieving specific information very convenient.
Consistency	Changes made in one file may not update in another file, leading to inconsistency and errors in data.	DBMS ensures consistency by updating related tables automatically and maintaining integrity using rules and constraints.
Example	In a college, student information is stored in one file and course details in another. Linking which student has taken which course becomes difficult.	In DBMS, a “Students” table and a “Courses” table can be created, and relationships can be defined between them for easy linking.

Concepts of Data Abstraction and Data Independence

Aspect	Data Abstraction	Data Independence
Definition	Data Abstraction is the process of hiding the complex details of how data is stored and managed in the database. Instead of exposing the technical implementation, it shows only the necessary and relevant information to the user. The main purpose of data abstraction is to make database usage simple by focusing on what data is required rather than how it is stored or handled internally.	Data Independence is the ability to change the schema of a database at one level without affecting the schema at the next higher level. Its main purpose is to ensure that applications remain unaffected by structural changes in the database, making the system more flexible and easier to maintain.
Types	1. External Level (View Level) This level shows only a specific part of the database to individual users according to their needs. Users do not see the whole database; instead, they see a customized view that is relevant to them. 2. Conceptual Level (Logical Level) This level gives the overall logical structure of the entire database. It describes all entities, their relationships, and the constraints that apply. It is mainly used by database administrators and designers to plan and manage the database. 3. Internal Level (Physical Level) This level focuses on how the data is actually stored in the system. It deals with file organization, indexes, storage devices, and access paths. It hides these complex storage details from the user to keep things simple.	1. Physical Data Independence Physical data independence means that any changes made at the internal level (like storage structure, file organization, or indexing) do not affect the conceptual schema or user views. In simple terms, even if the way data is stored changes, the logical design and how users see the data remain the same. 2. Logical Data Independence Logical data independence refers to the ability to make changes in the conceptual schema (such as adding new attributes, modifying relationships, or changing constraints) without affecting the external schema or application programs. This ensures that users and applications can continue working without any modification, even if the database structure is updated.
Goal	1. Simplification - To hide complex technical details of how data is stored or managed, and provide only the useful information to users. 2. Focus on relevant data - To allow users to concentrate on	1. Insulation from changes - To protect applications and users from being affected when changes are made to the database schema.

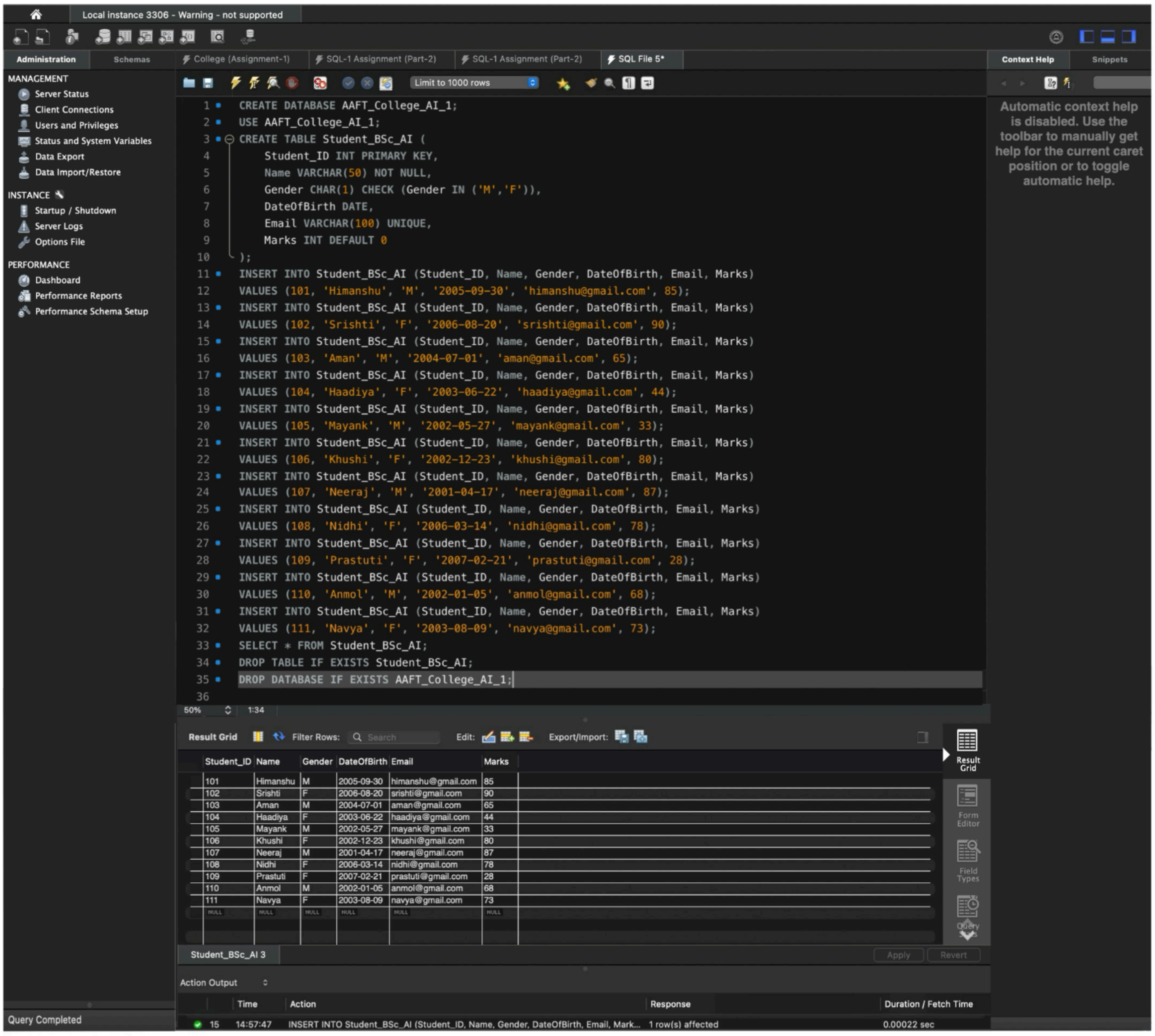
	what data they need, without worrying about how it is organized or maintained internally. 3. Ease of use - To make interaction with the database simple, even for non-technical users. 4. Separation of concerns - To divide database design into different levels (external, conceptual, and internal), so that each level deals with only its own responsibility.	2. Flexibility - To allow modifications at one level (physical or logical) without disturbing higher levels. 3. Reduced maintenance - To minimize the need for rewriting application programs when the database structure changes. 4. Stability of applications - To ensure that user applications keep working properly, even if the way data is stored or structured changes.
Benefits	1. Separation of Views Each user only sees the part of the database that is relevant to them. For example, a bank customer sees account details, while an employee sees customer and loan data. This makes the system easier to use. 2. Hides Complexity The internal storage details (like indexing, file organization, or memory management) are hidden from end users. Users only focus on what data they need, not how it is stored. 3. Data Security Since not all users see the entire database, sensitive information can be restricted. For example, a teller cannot see the bank's complete financial records. 4. Supports Data Independence Changes at one level (like internal storage structure or logical schema) do not affect other levels. This reduces the impact of modifications and makes applications stable. 5. Flexibility in Design The database can be modified or extended at one level without disturbing users at another level. This makes database management more flexible and efficient.	1. Flexibility in Physical Storage: You can make changes to the database's physical storage (like how files are stored or how indexing is done) without affecting the user's application programs. This makes it easier to optimize performance. 2. Ease of Maintenance: When changes are made to the database, such as adding new fields or changing a field's data type, the logical and physical levels can be handled separately. This makes maintenance much easier. 3. Insulation from Changes: User applications only interact with the logical view of the data. When a change is made at either the physical or logical level, the applications are "insulated" (protected) from it. This means you don't need to rewrite their code. 4. Security and Abstraction: Users don't need to know the "how" of data storage. This not only reduces complexity but also increases security because users only have access to the data that is relevant to them. 5. Simplicity for Developers: Application developers don't have to concern themselves with the internal details of the database. They can focus solely on the logical model, which makes the development process simple and efficient.
Example	● A bank database stores details about customers (name, account number, balance, transactions).	● Physical Data Independence: If the bank changes the way account details are stored (say from simple

	● A customer only sees their personal details and transactions (external level). ● The DBA sees the overall design of how customers, accounts, and transactions are related (conceptual level). ● The system manages where and how these records are stored on disk (internal level). ● This mechanism hides the complexity of storage from the customer, showing only what is necessary.	files to indexed files for faster search), the applications used by customers remain unchanged. ● Logical Data Independence: If the bank decides to add a new attribute like Customer Email in the conceptual schema, it does not affect the customer's existing applications or external views. ● This mechanism ensures stability and flexibility of applications even when the database evolves.

Types of SQL

Types	Definition	Common Command	Example	Use Case
Data Definition Language (DDL)	DDL commands are used to define and manage the structure of database objects such as tables, schemas, and indexes. In simple words, they decide how the database will look.	CREATE, ALTER, DROP, TRUNCATE	CREATE TABLE Students (StudentID INT PRIMARY KEY, Name VARCHAR(50), Age INT);	When a new table needs to be created in the database to store student information.
Data Manipulation Language (DML)	DML commands are used to manipulate the actual data stored in the database. They allow us to insert, modify, or remove records.	INSERT, UPDATE, DELETE	INSERT INTO Students (StudentID, Name, Age) VALUES (101, 'Himanshu', 20);	When we want to add a new student record into the table.
Data Query Language (DQL)	DQL is mainly used to retrieve data from the database. It helps us ask questions (queries) from the database and get the required results.	SELECT	SELECT Name, Age FROM Students WHERE Age > 18;	When we need to find the names and ages of students who are above 18 years old.
Data Control Language (DCL)	DCL commands deal with rights, permissions, and other controls of the database system. These commands ensure security and control of data access.	GRANT, REVOKE	GRANT SELECT ON Students TO user1;	When a user should be given read-only permission to access student data.
Transaction Control Language (TCL)	TCL commands are used to manage transactions in the database. A transaction is a group of SQL statements that are executed together. TCL ensures that either all statements are successfully executed or none of them is applied, which keeps data consistent.	COMMIT, ROLLBACK, SAVEPOINT	BEGIN; UPDATE Students SET Age = 21 WHERE StudentID = 101; COMMIT;	When we want to update a student's age but also make sure that in case of an error, the update can be cancelled using ROLLBACK.

Implementation of SQL Commands with Constraints



Local instance 3306 - Warning - not supported

```sql
CREATE DATABASE AAFT_College_AI_1;
USE AAFT_College_AI_1;
CREATE TABLE Student_BSc_AI (
    Student_ID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Gender CHAR(1) CHECK (Gender IN ('M','F')),
    DateOfBirth DATE,
    Email VARCHAR(100) UNIQUE,
    Marks INT DEFAULT 0
);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (101, 'Himanshu', 'M', '2005-09-30', 'himanshu@gmail.com', 85);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (102, 'Srishti', 'F', '2006-08-20', 'srishti@gmail.com', 90);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (103, 'Aman', 'M', '2004-07-01', 'aman@gmail.com', 65);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (104, 'Haadiya', 'F', '2003-06-22', 'haadiya@gmail.com', 44);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (105, 'Mayank', 'M', '2002-05-27', 'mayank@gmail.com', 33);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (106, 'Khushi', 'F', '2002-12-23', 'khushi@gmail.com', 80);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (107, 'Neeraj', 'M', '2001-04-17', 'neeraj@gmail.com', 87);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (108, 'Nidhi', 'F', '2006-03-14', 'nidhi@gmail.com', 78);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (109, 'Prastuti', 'F', '2007-02-21', 'prastuti@gmail.com', 28);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (110, 'Anmol', 'M', '2002-01-05', 'anmol@gmail.com', 68);
INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Marks)
VALUES (111, 'Navya', 'F', '2003-08-09', 'navya@gmail.com', 73);
SELECT * FROM Student_BSc_AI;
DROP TABLE IF EXISTS Student_BSc_AI;
DROP DATABASE IF EXISTS AAFT_College_AI_1;
```

Student_ID	Name	Gender	DateOfBirth	Email	Marks
101	Himanshu	M	2005-09-30	himanshu@gmail.com	85
102	Srishti	F	2006-08-20	srishti@gmail.com	90
103	Aman	M	2004-07-01	aman@gmail.com	65
104	Haadiya	F	2003-06-22	haadiya@gmail.com	44
105	Mayank	M	2002-05-27	mayank@gmail.com	33
106	Khushi	F	2002-12-23	khushi@gmail.com	80
107	Neeraj	M	2001-04-17	neeraj@gmail.com	87
108	Nidhi	F	2006-03-14	nidhi@gmail.com	78
109	Prastuti	F	2007-02-21	prastuti@gmail.com	28
110	Anmol	M	2002-01-05	anmol@gmail.com	68
111	Navya	F	2003-08-09	navya@gmail.com	73
NULL	NULL	NULL	NULL	NULL	NULL

Student_BSc_AI 3

Action Output

```
    Time      Action                                                                      Response            Duration / Fetch Time
15  14:57:47  INSERT INTO Student_BSc_AI (Student_ID, Name, Gender, DateOfBirth, Email, Mark...  1 row(s) affected   0.00022 sec
```

Query Completed