ALLAMA IQBAL OPEN UNIVERSITY

USMAN NOTES
www.usmannotes.com

Name:

Registration No.:

Course Code: 3579

Program: Postgraduate

Semester: Autumn 2026

ALLAMA IQBAL OPEN UNIVERSITY, ISLAMABAD

(Department of Computer Science)

Course: Database Application (3579) • Level: Postgraduate

Semester: Autumn 2026

Assignment No. 1

Total Marks: 100 • Pass Marks: 50

SOLVED ASSIGNMENT GUIDE — For Study & Reference

Note: This guide is prepared for study and reference purposes only. Students are advised to read the relevant parts of the course book, understand the concepts, try the SQL examples and design exercises themselves in a practice database, and then write the assignment in their own words. Copying this guide word for word is not recommended. No units are printed in the question booklet for this course, so none are shown on this title page.

USMAN NOTES www.usmannotes.com

Question 1: From File-Based Systems to Database Systems; Modern Business Requirements (20 Marks)

Question as printed: a) How has the evolution from file-based systems to database systems transformed the way organizations handle data? Discuss the impact on data organization, accessibility, and decision-making. b) In what ways do modern business requirements expose the weaknesses of traditional file systems, and how do DBMS features such as data integration and centralized control provide effective solutions?

Introduction

Data is the memory of an organisation. Every admission in a university, every account in a bank, every order in a shop and every patient record in a hospital is first of all a piece of data that must be stored accurately, found quickly, shared safely and kept for as long as it is needed. The history of computerised data handling is largely the history of learning how to do these four things well. In the earliest stage each application program owned its own files; in the modern stage a single database management system, or DBMS, holds an integrated collection of related data for the whole organisation and serves many applications and many users at the same time. This answer traces that evolution, explains its effects on organisation, accessibility and decision-making, and then shows why modern business requirements expose the weaknesses of the older approach and how data integration and centralised control in a DBMS answer them.

Part (a): The Evolution and Its Impact

From Manual Files and Isolated Computer Files to a Shared Database

Before computers, organisations kept paper files, registers and ledgers. A computerised file-based system simply replaced the paper cabinet with magnetic files: the payroll department kept a payroll file, the personnel department kept an employee file, and admissions, examinations and accounts in a university each kept separate files, often on different media and in different formats. Each program defined the structure of its own data, knew the file's physical format, and contained the logic for reading, updating and reporting on that file alone. This worked while applications were few, data volumes were small and departments worked independently. As soon as the same fact — an employee's name, a student's address, a

customer's balance — was needed in several places, it was copied into several files, and the copies began to disagree.

A database system changes the unit of thinking from the file to the organisation. Data are collected once, described once in a catalogue or data dictionary, and stored in tables related to one another by meaningful relationships, such as a student being enrolled on a course or an account belonging to a customer. The DBMS — software such as Oracle, MySQL, PostgreSQL or Microsoft SQL Server — sits between the users and the stored data. Programs and users state what data they need, usually in a language such as SQL, and the DBMS decides how to find it. Concepts such as data independence follow from this separation: the physical storage can be reorganised, and even the logical structure can be extended, without rewriting every application program, because programs no longer depend on the exact layout of files.

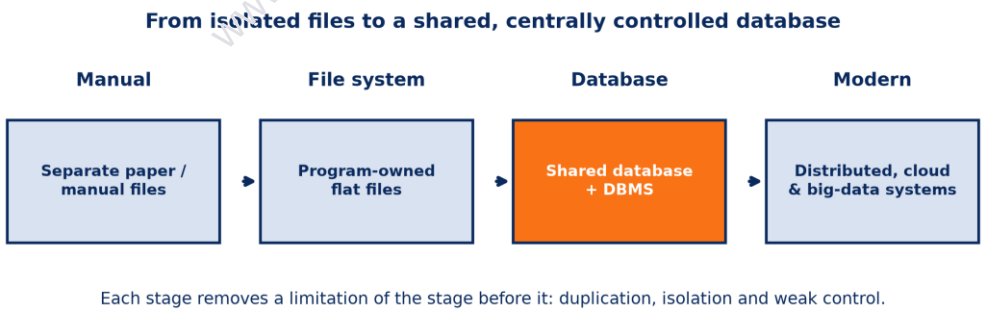


Figure 1: The evolution from isolated, program-owned files to a shared database under central DBMS control, and onward to distributed and cloud systems.

Impact on Data Organisation

Organisation improves in three related ways. First, data are integrated: related items that were scattered across departmental files are brought together and linked, so a single query can

combine, for example, a student's personal record, fee payments and examination results. Second, redundancy is controlled: a fact is normally stored once and referenced elsewhere by a key, instead of being typed again in every file that needs it. Third, standards become possible. Because one system administers the data, naming conventions, data types, codes and formats can be enforced for everyone, so 'M.Com', 'MCom' and 'Master of Commerce' do not appear as three different qualifications in three different files. The database therefore reflects the organisation as a whole rather than the accidental history of its separate programs.

Impact on Accessibility

In a file system, access was effectively restricted to the questions the original programs had been written to answer. A new question — list all students who have paid the fee but not submitted the assignment — meant writing a new program, merging files and waiting for a programmer. In a database system, authorised users can ask new, unplanned questions directly through a query language, a report generator or a dashboard, and receive an answer in seconds. Access also becomes concurrent and selective: many users can work on the same database at the same time, each seeing the view of the data appropriate to his or her job — a clerk sees admissions forms, an examiner sees results entry, and management sees summaries — while the DBMS controls who may see or change what. Accessibility therefore improves in speed, in flexibility and in fairness, without giving up control.

Impact on Decision-Making

Decisions are only as good as the information behind them. When data are fragmented and inconsistent, managers distrust reports, keep private copies and decide on the basis of

incomplete figures. An integrated database gives decision-makers a single, current and consistent version of the facts: enrolment trends, fee recovery, pass rates, stock levels or loan defaults can be analysed across departments and across years. This supports every level of decision — operational decisions such as generating today's list of unpaid accounts, tactical decisions such as opening an extra section of a popular course, and strategic decisions such as launching a new programme in a region where enquiries are growing. Later developments built on the same foundation, including data warehouses, business intelligence tools and data analytics, all presuppose that the underlying operational data are integrated and reliable in the way only a database approach makes routine.

Table 1: File-based systems and database systems compared

Aspect	Traditional file-based system	Database system (DBMS)
Data ownership	Each program / department owns its files	Data are a shared organisational resource
Redundancy	High; the same fact copied in many files	Controlled; a fact is stored once and referenced by keys
Consistency	Copies disagree after partial updates	One update is visible to all authorised users
New questions	Need a new program	Answered by a query, view or report
Standards and security	Vary from file to file; weak	Enforced centrally by the DBMS and administrator
Data independence	Programs depend on file layout	Programs are insulated from

		storage changes

Part (b): Modern Business Requirements and the Weaknesses They Expose

What Modern Organisations Now Require

Modern business requirements are very different from those of the batch-processing era. Organisations work online and continuously: customers expect a bank balance, an order status or an admission decision at any hour, often from a phone. Volumes are large and growing, users are numerous and geographically dispersed, and different departments, branches and partner organisations must work from the same facts in real time. Rules also change frequently — a fee schedule, a tax rate, a new programme, a revised eligibility rule — and the system must absorb the change once, correctly, without stopping work. Finally, data have legal and ethical value: personal and financial data must be protected, auditable and recoverable after failure. Each of these requirements presses exactly on a known weakness of traditional file systems.

- Real-time, multi-user working exposes the file system's lack of concurrency control: two clerks updating copies of the same record can overwrite each other's work, and there is no transaction concept to keep a multi-step operation all-or-nothing.
- Growth exposes inflexibility: because programs are tied to file structures, adding a field or a new relationship means changing and retesting many programs, so change is slow, expensive and risky.

- Integration across departments exposes duplication and inconsistency: marketing, sales and accounts holding different addresses and balances for the same customer cannot produce a trustworthy combined report.
- Self-service and ad hoc reporting expose restricted access: if every new report needs a programmer, information arrives too late to be useful.
- Privacy, audit and recovery requirements expose weak security: file-level passwords and operating-system permissions cannot easily express 'this user may see marks but not fee records', and backup is a matter of copying files rather than recovering to a consistent point.

How DBMS Features Provide Effective Solutions

Data integration is the first solution. In a DBMS, data from different functions are designed as one logical database, related by keys, so sales, stock and accounts work from the same product and customer records. Integration does not merely save storage; it makes cross-functional questions answerable and removes the contradictions that arise when departments maintain private versions of the truth. Centralised control is the second solution. A database administrator, working through the DBMS, defines the schema, the integrity constraints, the authorisations and the backup policy once, and the system enforces them for every application and every user. Central control here means central administration of rules, not necessarily that all data sit on one machine; modern DBMSs apply the same discipline to distributed and cloud databases.

Around these two foundations the DBMS adds supporting mechanisms that this course returns to repeatedly: transactions with atomicity, consistency, isolation and durability (the ACID properties) for safe concurrent work; a query optimiser and indexes for performance; views for giving each class of user a safe, simplified picture of the data; constraints for integrity; and logging, backup and recovery for reliability. Taken together, they convert data handling from a collection of local habits into a managed organisational function. That is why, although files remain entirely appropriate for simple, single-purpose tasks such as logs or documents, the core operational data of any sizeable modern organisation are kept in a database system.

Conclusion

The evolution from file-based systems to database systems transformed data organisation by integrating related data and controlling redundancy, transformed accessibility by allowing concurrent, selective and ad hoc access through a managed interface, and transformed decision-making by giving managers one consistent and analysable version of the facts. Modern requirements — online working, many simultaneous users, frequent change, large volumes and strict duties of protection and recovery — expose precisely the weaknesses of isolated files: duplication, inconsistency, inflexibility, weak security and poor support for new questions. DBMS features, above all data integration and centralised control supported by transactions, constraints, views and recovery, provide effective solutions to each of those weaknesses, which is why the database approach is now the normal foundation of organisational information systems.

Question 2: Efficiency in Data Management; Redundancy and Inconsistency (20 Marks)

Question as printed: a) Why is efficiency a critical factor in data management systems, and how do database systems improve performance compared to traditional file processing methods? b) How do redundancy and inconsistency affect system performance in file-based environments, and what mechanisms do DBMS use to minimize these issues?

Introduction

A data system can be logically correct and still be practically useless if it is too slow, wastes storage, or forces users to wait while records are searched one by one. Efficiency — obtaining the required result with a reasonable use of time, storage and processing effort — is therefore not a luxury added after design; it is one of the main reasons organisations moved from file processing to database systems. This answer explains why efficiency matters so much, how a DBMS improves performance in comparison with traditional file processing, and then examines the closely related problems of redundancy and inconsistency: how they damage performance in file-based environments and which DBMS mechanisms minimise them.

Part (a): Why Efficiency Is Critical, and How DBMS Improves Performance

Why Efficiency Matters

First, data volumes grow relentlessly. A method that works for a thousand records by reading a file from the beginning may be hopeless for ten million records, because the work grows in proportion to the file size. Second, time is operationally valuable: a cashier, an admissions clerk

or an online customer experiences the system's response time directly, and slow systems create queues, abandoned transactions and extra staff work. Third, inefficiency is multiplied in multi-user systems, where hundreds of users compete for the same processor, memory and disks; a wasteful query by one user delays everyone else. Fourth, inefficient storage and repeated processing cost money — in hardware, in cloud charges, in backup time and in maintenance. Finally, efficiency determines what is possible at all: real-time fraud checks, instant balance enquiries and live dashboards exist only because the underlying data system can answer in a fraction of a second.

How Database Systems Improve Performance

The most important performance device is selective access through indexes. Instead of scanning every record, the DBMS uses an index — commonly a B+ tree — to go almost directly to the records that satisfy a condition on an indexed column, much as a reader uses a book's index instead of reading every page. Next, the DBMS contains a query optimiser: given a declarative SQL statement, it considers alternative execution plans, estimates their cost from statistics about the data, and chooses an efficient plan, for example using an index, joining small tables before large ones, or filtering early. A file-processing program, by contrast, executes exactly the access path the programmer coded, whether or not it is still the best one as data grow.

Performance also improves through buffering and caching, in which frequently used pages of data are kept in main memory rather than read repeatedly from disk; through set-oriented processing, in which one SQL statement processes a whole set of rows inside the DBMS instead

of a program fetching records one at a time across a network; and through controlled concurrency, in which locking, multiversion techniques and transaction scheduling allow many users to work in parallel safely instead of locking a whole file for one user. Storage efficiency contributes as well: a well-designed, normalised database stores each fact once, so there is simply less data to scan, back up and maintain. None of this removes the need for good design — a missing index or a badly written query can still be slow — but the DBMS gives the administrator tools to measure and improve performance, such as execution plans and monitoring, that simply do not exist in a collection of independent files.

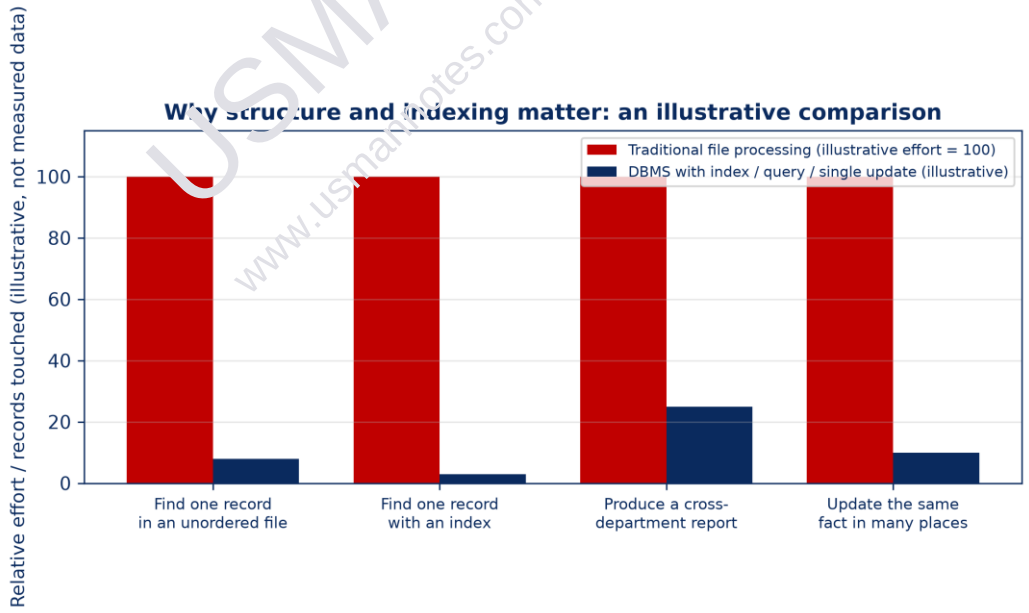


Figure 2: Illustrative comparison of relative effort. The figures are a teaching illustration of orders of magnitude, not measurements from a particular system: an indexed, optimised DBMS search touches far fewer records than a full file scan.

Table 2: Where performance is gained

Performance device	What it does	Typical benefit

Index (for example B+ tree)	Orders key values and points to matching records	Finding a record without scanning the whole table
Query optimiser	Chooses a low-cost execution plan using statistics	Better joins, filtering and access paths automatically
Buffering / caching	Keeps hot data pages in memory	Avoids repeated slow disk reads
Set-oriented SQL	Processes sets of rows inside the DBMS	Less program code and less data movement
Concurrency control	Schedules many transactions safely	Many users work in parallel instead of queuing for a file
Controlled redundancy	Stores each fact once in a normalised design	Less to store, scan, update and back up

USMAN NOTES www.usmannotes.com

Part (b): Redundancy, Inconsistency and Performance

What Redundancy and Inconsistency Are

Redundancy is the unnecessary duplication of the same data in more than one place. In a file-based environment it arises naturally, because each application keeps the data it needs: the admissions file, the examination file and the fee file may each store a student's name, father's name, address and programme. Inconsistency is the natural consequence: when a fact changes and is updated in one file but not in the others — a student changes address, a customer is recorded with a corrected name — the files contradict one another, and the organisation no longer knows which copy is correct. A related family of problems, studied again under

normalisation in Assignment No. 2, consists of insertion, update and deletion anomalies, where storing combined facts in one file makes ordinary operations awkward or dangerous.

How They Damage Performance

The performance damage is direct and cumulative. Storage is wasted, so files are larger, scans are longer, backups take more time and hardware costs rise. Every update must be repeated in every copy, turning one logical change into many physical writes, often in different programs run at different times; this multiplies processing, lengthens batch runs and increases the chance that some copies are missed. Searching becomes slower and less certain, because a report may have to read and reconcile several files, detect conflicting values and decide which to trust — work that produces no new information. Programmes also become larger and harder to maintain, since each must know about the duplications it depends on. Perhaps most seriously, inconsistency destroys trust: staff keep private spreadsheets, verify figures by telephone and re-enter data, all of which is hidden extra work caused by the system's inefficiency. Redundancy is therefore not only a question of correctness; it is a continuing tax on every operation the system performs.

DBMS Mechanisms That Minimise Redundancy and Inconsistency

- Single integrated database with keys: the fact is stored once in its own table — a student once in the Student table — and other tables refer to it by the primary key / foreign key relationship instead of copying the details.

- Normalisation: the design process of decomposing tables according to functional dependencies removes most unintended duplication and the anomalies that accompany it, as discussed in detail in Assignment No. 2, Question 1.
- Integrity constraints: primary key, unique, foreign key, check and not-null constraints, enforced by the DBMS on every transaction, prevent duplicate identities, orphan references and invalid values from entering the database in the first place.
- Transactions and concurrency control: a multi-step update either completes entirely or is rolled back entirely, and concurrent users are isolated from one another's half-finished work, so partial updates cannot leave copies — or related records — in contradictory states.
- Views and centralised definitions: where users need a combined or derived picture, a view presents it without physically duplicating the data; the data dictionary keeps one definition of each item for all applications.

A balanced judgement is needed on redundancy itself. A DBMS does not forbid every duplication: a designer may deliberately introduce controlled redundancy — for example a stored total, a replicated copy on another server, or a denormalised reporting table — to gain performance or availability. The difference is that in a database system such redundancy is a conscious design decision, documented and maintained by the system or by defined procedures, whereas in a file system it is uncontrolled, undocumented and discovered only when the copies disagree. The goal is therefore not the impossible one of zero duplication in every circumstance, but the practical one of eliminating uncontrolled redundancy and protecting whatever duplication remains.

Conclusion

Efficiency is critical because data volumes, user numbers and expectations of immediate response all keep growing, and because waste in a shared system is multiplied across every user and every operation. Database systems improve performance through indexes, query optimisation, buffering, set-oriented processing, managed concurrency and compact normalised storage — tools that file processing does not provide. Redundancy and inconsistency attack the same system from the other side, wasting storage, multiplying update work, slowing searches and destroying trust in results. The DBMS answers with integration, keys, normalisation, constraints, transactions and views, reducing duplication to a controlled design choice rather than an accident, and in doing so it improves correctness and performance together.

Question 3: Changing Needs, Flexibility and Scalability (20 Marks)

Question as printed: a) How do changing organizational needs influence the requirement for flexible and scalable data systems? Compare how file systems and DBMS handle such changes. b) Why are scalability and adaptability important in modern applications, and how does DBMS ensure long-term system growth and modification with minimal disruption?

Introduction

No organisation stands still. New programmes, products, branches, regulations, partners and technologies appear continuously, and the data system must absorb all of them. A system that fitted yesterday's organisation perfectly can become today's bottleneck if every change requires months of reprogramming or if performance collapses as soon as the number of

records or users doubles. Flexibility — the ease with which the structure and content of the system can be changed — and scalability — the ability to handle growth in data, users and workload while continuing to perform acceptably — are therefore core requirements, not optional extras. This answer examines why changing needs create these requirements, compares the file-based and database approaches to change, and explains how a DBMS supports long-term growth and modification with minimal disruption.

Part (a): Changing Needs and the Two Approaches to Change

Kinds of Change an Organisation Faces

Changes come in several distinct forms, and a good data system must handle all of them. Structural changes add new kinds of data or new relationships: a university introduces an online learning programme, a bank adds a new type of account, a hospital begins recording a new test. Content and rule changes alter values and policies without altering the basic structure: fee rates, eligibility rules, tax percentages and reporting categories change regularly. Volume and user changes increase the amount of data and the number and geographical spread of the people using it, often sharply at predictable peaks such as admission or enrolment periods. Technological and organisational changes replace hardware and platforms, merge departments or organisations, open new branches, and require the same data to serve new channels such as web portals and mobile applications. Finally, reporting and legal changes demand new outputs, new retention periods and new audit trails, frequently at short notice. Each kind of change tests the data system in a different way.

How File Systems Handle Change

A traditional file system handles change badly because of data dependence: application programs contain detailed knowledge of the files they use — record layouts, field sizes, ordering and access methods. Adding a field to a record, splitting a file, changing a code's length or reorganising a file for performance therefore means finding, modifying, recompiling and retesting every program that uses that file. In a mature installation nobody may even be certain which programs those are, so change is approached cautiously, slowly and expensively, and urgently needed changes are sometimes refused or worked around with yet another separate file — which increases duplication and inconsistency further. Growth is handled no better: as files grow, sequential searches slow down in proportion; as users multiply, whole-file locking and separate departmental copies make concurrent working difficult; and moving to a larger machine or a new platform often means rewriting rather than reconfiguring.

How a DBMS Handles Change

A database system is designed around data independence, which has two levels. Physical data independence means the storage can be reorganised — indexes added, files moved, partitions introduced, hardware replaced — without changing the logical data or the applications, because the DBMS translates between the logical request and the physical storage. Logical data independence means the overall logical structure can be extended — a column added, a new table introduced, a relationship created — while existing applications continue to work through their existing views or schemas; only applications that need the new data are changed. In practice, many common changes become administrative actions rather than programming

projects: adding a column with a default value, creating an index to restore performance, defining a new view for a new report, or granting a new class of user access to an existing view. The DBMS also separates the definition of data, held in its catalogue, from the programs that use the data, so the effect of a proposed change can be inspected before it is made.

Table 1: Handling change — file system and DBMS compared

Change required	File-based system	DBMS
Add a new field / attribute	Modify every program using the file; reformat data	Add a column; existing views and programs usually unaffected
Answer a new kind of question	Write and test a new program	Write a query or define a view
Restore performance as data grow	Rewrite access logic or reorganise files manually	Add an index, partition a table, adjust storage
Serve many more users	File locking and separate copies obstruct work	Concurrency control, connection management, replication
Move to new hardware / platform	Programs tied to old file formats need rewriting	Physical data independence; migrate data and catalogue
Merge two departments' data	Reconcile incompatible files by hand	Integrate schemas in one database with shared keys

USMAN NOTES www.usmannotes.com

Part (b): Scalability, Adaptability and Long-Term Growth

Why Scalability and Adaptability Matter in Modern Applications

Modern applications succeed or fail partly by how they behave under growth. A service that begins with a few thousand users may be expected to serve hundreds of thousands; an archive that begins with one year's records must hold twenty years' records and still answer enquiries about the oldest ones; and peak loads — the first day of admissions, a sale, a results announcement — may be many times the average load. If the data system cannot scale, success itself breaks it, and the organisation faces an emergency replacement at the worst possible moment. Adaptability matters for a parallel reason: applications now live for many years and pass through successive business, legal and technological environments. A system that cannot be modified safely accumulates workarounds until nobody dares to touch it, at which point the organisation is effectively governed by an accident of its own history. Scalability protects the future volume of work; adaptability protects the future shape of work.

How a DBMS Ensures Growth and Modification with Minimal Disruption

- Data independence, as explained above, is the foundation: storage can be reorganised and the logical schema extended while applications continue through stable views, so most growth work happens inside the DBMS rather than in every program.
- Indexing, partitioning and query optimisation allow very large tables to remain usable: a table can be partitioned by year, region or branch so that queries and maintenance work on the relevant part, and the optimiser adapts execution plans as statistics change.

- Concurrency and transaction management scale the user side of growth: locking at row rather than file level, multiversion read consistency and connection pooling allow large numbers of simultaneous users without the system serialising into a queue.
- Distribution and replication scale the organisation side: data can be distributed across branches or servers, replicated for local speed and availability, and hosted in the cloud where storage and processing capacity can be increased as demand requires, while users still see one logical database.
- Controlled schema evolution and administration tools — online index creation, in-place alterations in modern systems, backup without stopping service, and monitoring — allow many modifications to be made while the system remains in operation, reducing the need for disruptive downtime.

These mechanisms have limits that a postgraduate answer should acknowledge honestly. Scalability is not automatic or infinite: poor schema design, missing indexes, and transactions that hold locks for a long time will defeat any DBMS, and at very large scale designers may adopt specialised distributed or NoSQL systems for particular workloads. Adaptability also has a cost in discipline: changes must still be planned, tested and documented, and logical data independence protects existing views only within limits — removing or redefining data that applications depend on will always require coordinated work. The claim for the DBMS is comparative, not magical: it localises change, provides tools for growth, and makes both routine rather than heroic, whereas the file approach spreads the effect of every change across the whole application portfolio.

Conclusion

Changing organisational needs — new structures, new rules, more data, more users, new channels and new legal duties — make flexibility and scalability unavoidable requirements. File systems meet change with data dependence, so each alteration ripples through many programs and each increase in volume or users degrades the whole system. A DBMS meets change with physical and logical data independence, a centrally managed schema, views, indexing, partitioning, concurrency control and options for distribution and replication. Together these allow a database to grow over the long term and to be modified with minimal disruption, provided the design is sound and changes are administered with proper care.

Question 4: Data Integrity and Security (20 Marks)

Question as printed: a) Why are data integrity and security major concerns in data management? Evaluate how traditional file systems and DBMS differ in their approaches to addressing these concerns. b) How do DBMS features such as constraints, access control, and backup mechanisms improve data reliability compared to file-based systems?

Introduction

Two questions must be answered affirmatively before any data system can be trusted: is the data correct, and is it protected? Integrity concerns the first question — whether the stored data are accurate, consistent and valid according to the rules of the organisation. Security concerns the second — whether only authorised people can see or change data, and whether the data survive accidents, failures and attacks. This answer explains why both concerns are major, evaluates the very different approaches of file systems and database systems, and then

examines three DBMS mechanisms in detail — constraints, access control and backup with recovery — and the improvement in reliability they produce. The discussion is academic and general; questions about the legal liability of particular persons in a particular case belong to qualified legal professionals and the courts, not to a database textbook.

Part (a): Why Integrity and Security Matter, and the Two Approaches

Why These Concerns Are Major

Data drive decisions and obligations. A wrong balance can overdraw a customer's account or conceal a fraud; a wrong result can deny a student a degree or award one wrongly; a leaked personal record can harm the person concerned, expose the organisation to legal penalties and destroy public trust. Errors and breaches also propagate: because reports, payments and further records are derived from stored data, one incorrect or unauthorised change can generate many incorrect consequences before it is noticed. Modern conditions intensify both concerns. Data are shared by many users and applications, accessed over networks, and held in large volumes that no one can check by eye; threats range from ordinary typing mistakes and program bugs to hardware failure, ransomware and deliberate insider misuse. Integrity and security must therefore be provided by the system itself, continuously and automatically, rather than depending on every user and every program always behaving perfectly.

The File-System Approach

In a file-based system, integrity and security are largely the responsibility of each application program and of the operating system's file permissions. A careful programmer can validate input in one program, but another program updating the same or a related file may omit the

same check; nothing at the level of the data themselves prevents an invalid value, a duplicate identity or a reference to a non-existent record from being stored. Consistency between files is maintained, if at all, by running update programs in the right order and hoping none fails in the middle. Security is coarse: a user who can open a file can usually read or copy all of it, and the distinction between seeing one field and another, or between reading and updating, is hard to express. Audit trails, encryption and recovery depend on separate arrangements that vary from application to application. The result is not that file systems have no protection, but that protection is fragmented, duplicated and easy to bypass or forget.

The DBMS Approach

A DBMS centralises both concerns in the system that holds the data. Integrity rules are declared once, as part of the database definition, and the DBMS enforces them on every insertion, update and deletion, whoever or whatever issues it. Security is likewise enforced by the DBMS: users are authenticated, authorisations are granted on specified objects and operations, and views can restrict a user to the rows and columns his or her job requires. Around these stand the transaction mechanism, which guarantees that a group of related changes is applied completely or not at all and that concurrent users do not corrupt one another's work, and the recovery mechanism, which uses logs and backups to restore a consistent state after failure. Because enforcement lives with the data rather than inside each program, a new application automatically inherits the existing protections instead of having to reinvent them.

Table 1: Integrity and security — the two approaches

Concern	Traditional file system	DBMS

Validity of values	Checked, if at all, inside each program	Declared constraints enforced centrally on all operations
Consistency between related data	Depends on correct ordering of separate programs	Transactions and referential integrity keep related data together
Who may see / change data	Coarse file permissions; often all-or-nothing	Authentication, privileges, roles and views at table, row and column level
Concurrent use	Risk of overwriting and partial updates	Locking / multiversion control with isolation between transactions
Failure and recovery	Copy files back; consistency uncertain	Log-based recovery to a consistent point; backup and restore managed
Accountability	Logs, if any, differ per application	Central audit facilities record who did what

USMAN NOTES www.usmannotes.com

Part (b): Constraints, Access Control and Backup

Constraints: Protecting Correctness at the Point of Entry

Constraints are rules the DBMS checks automatically. Entity integrity, through a primary key that is unique and not null, ensures every record is identifiable and never duplicated. Referential integrity, through foreign keys, ensures that a reference always points to an existing

record — an enrolment cannot name a student who does not exist — and controls what happens when a referenced record is deleted or changed. Domain and check constraints confine a column to legitimate values, such as marks between 0 and 100 or a date of joining not in the future; unique constraints prevent the same national identity number or registration number being registered twice; and not-null constraints ensure essential facts are never simply left blank. The reliability gain is fundamental: invalid data are rejected at the moment of entry by the system itself, in every application, instead of being discovered months later in a report — if they are discovered at all.

Access Control: Protecting Data from Misuse

DBMS access control works in layers. Authentication establishes who the user is, increasingly supported by strong passwords or additional factors. Authorisation then states precisely what that identity may do — select, insert, update or delete — on which tables, views or procedures, using the GRANT and REVOKE facilities of SQL and grouping users into roles such as clerk, examiner or auditor so that permissions follow jobs rather than individuals. Views add a further, powerful restriction: a user may be given access only to a view showing, for example, names and results but not fee or personal-contact columns, and only for his or her own department's rows. Encryption protects stored and transmitted data against theft of files or interception, and auditing records access and change so that misuse can be detected and investigated. Together these implement the principle of least privilege — each user receives the minimum access his or her work requires — which file permissions cannot express with anything like the same precision.

Backup and Recovery: Protecting Data against Loss

Reliability also means surviving failure. A DBMS keeps a transaction log recording changes, supports full, incremental and differential backups taken on a schedule — in modern systems often while the database remains in use — and provides recovery procedures that restore the last backup and replay or undo log entries to return the database to a consistent state, either as it was just before the failure or at a chosen point in time. Archive copies, off-site or cloud copies and periodic test restores complete the discipline; an untested backup is a hope, not a safeguard. In a file environment, by contrast, backup usually means copying files at a moment when some may be mid-update, and restore means putting those copies back in whatever mutually inconsistent state they happened to be in, with no log to reconcile them. The DBMS does not abolish the need for a backup policy — someone must still decide frequency, retention and testing — but it provides the mechanisms that make a reliable policy possible.

USMAN NOTES www.usmannotes.com

Conclusion

Integrity and security are major concerns because incorrect data produce incorrect decisions and obligations, because breaches harm individuals and organisations, and because in shared, networked, high-volume systems neither correctness nor protection can be left to individual care. File systems leave both to separate programs and coarse file permissions, so protection is fragmented and easily missed. A DBMS centralises enforcement with the data: constraints reject invalid data at entry, access control with authentication, privileges, roles, views, encryption and audit restricts data to authorised use, and logging with backup and recovery

restores a consistent state after failure. These mechanisms together explain the marked improvement in data reliability that a database system offers over file-based processing.

Question 5: A Bank or a University — File System versus DBMS in Practice (20 Marks)

Question as printed: a) Consider an organization like a bank or university. How would data management challenges differ if it were used a traditional file system instead of a DBMS? b) How does the implementation of DBMS improve operational efficiency, data consistency, and user experience in such real-world environments?

Introduction

Abstract comparisons become clear when they are tested against a real organisation. A bank and a university are both excellent test cases: each serves very many people, keeps records that must remain accurate for years, runs many interlocking functions at once, and is trusted with sensitive personal and financial information. This answer takes such an organisation — using a university in most examples, with banking parallels — and asks first what its data management challenges would look like under a traditional file system, and second how implementing a DBMS improves operational efficiency, data consistency and the experience of students, customers and staff. The examples are illustrative of typical practice; they are not a description of any one named institution's internal systems.

Part (a): The Challenges under a Traditional File System

One Person, Many Partial Records

Consider a single student. Admissions holds her application file, the accounts office holds her fee file, each department holds enrolment lists, the examination section holds her results file, and the library holds her borrowing file. In a file-based installation each of these is a separate system with its own copy of her name, address, phone number and programme, collected on different forms at different times and typed by different clerks. The challenges follow immediately. Her address changes and is corrected in the admissions file, but dispatch continues to send material to the old address from another file. Her programme is recorded differently in two files, so she is omitted from a merit list compiled from one of them. No single enquiry can answer the simple question 'what is this student's complete position?' because the answer lives in five places that do not communicate. A bank experiences the identical pattern when a customer has a current account, a savings account, a loan and a card, each maintained as a separate file with its own copy of the customer's details and balance logic.

- Duplication and inconsistency: the same person appears many times with diverging details, and staff cannot tell which copy is authoritative, so time is spent reconciling — and decisions are sometimes taken on the wrong version.
- No live, shared picture: files are updated by separate programs, often in batches, so a fee paid in the morning may not be visible to the department in the afternoon; in a bank, a deposit or withdrawal not yet reflected in another file can produce a wrong balance or a wrongly refused transaction.

- Concurrency and peak-load failure: admission deadlines, result announcements and salary days concentrate work into short peaks. File locking and departmental queues mean clerks wait, counters close, and online self-service is effectively impossible to offer safely.
- Security and privacy exposure: fee, results and personal data sit in files copied between offices for reporting; access cannot be limited to 'results only, own department only', and there is no reliable record of who saw or changed what — a serious matter for examination data and for customers' financial data alike.
- Weak recovery and continuity: if a file is corrupted or a program fails midway through an update run, related files are left in mismatched states with no transaction log to restore consistency; reconstructing a student's record or a day's banking transactions becomes manual detective work.
- Slow institutional reporting: questions that cross departments — how many enrolled students are also fee defaulters, which programmes are growing, what is the bank's total exposure to one customer across all products — require merging files by special programs, so management information is late, contested or simply not produced.

Part (b): What Implementing a DBMS Changes

Operational Efficiency

With a DBMS, the institution keeps one integrated database: a student is recorded once, linked by her registration number to admissions, fee, enrolment, results and library records; a bank customer is recorded once, linked to all of his accounts. Day-to-day operations speed up throughout. Counter and office staff retrieve a complete, current record with one enquiry

instead of consulting several systems; routine documents such as fee challans, transcripts, account statements and reminder letters are generated from the same data; and self-service becomes practical, because a student portal or an ATM and mobile banking service are simply controlled views and transactions on the central database. Peak loads are handled by concurrency control and indexing rather than by locking files, and staff effort shifts from copying and reconciling data to serving people and handling genuine exceptions, which is where human attention is actually needed.

Data Consistency

Consistency improves because it is built into the structure rather than hoped for. A change of address is made once and is immediately the address everywhere; a fee payment updates the student's account in a transaction that either completes fully or not at all, so the receipt, the balance and the department's defaulter list cannot tell three different stories. Constraints prevent an enrolment for a non-existent student, a result outside the permitted marks range, or a withdrawal exceeding an account's balance where the business rule forbids it, and referential integrity keeps every related record attached to its owner. Departments stop maintaining shadow copies, because the central record is demonstrably the current one; where a department needs its own presentation of the data, it is given a view, not a private file. Institutional reports consequently agree with one another, because they are all drawn from the same facts.

User Experience

For students and customers, the improvement is experienced as simplicity and trust. There is one identity and one record: a student can check admission status, fee position, results and library issues through one portal, and a customer can see all accounts and a combined statement through one login. Services become faster and available outside office hours, notifications are accurate because they are generated from current data, and errors, when they occur, can be traced and corrected in one place with an audit trail, instead of requiring the person to visit several offices each holding a different version of his or her record. For staff, the experience improves in parallel: a single sign-on with role-based access shows each officer exactly the functions his or her job needs, training is simpler because there is one system to learn, and confidence rises because the figures on the screen can be relied upon. Accessibility for management also changes the institution's possibilities — reliable, cross-functional information supports planning of programmes, staffing, branches and services on evidence rather than on competing departmental claims.

Table 1: The same institution under the two approaches

Experience	Traditional file system	With a DBMS
A student's / customer's record	Several partial copies in departmental files	One record, linked to all services
Effect of one update	Must be repeated; some copies missed	Made once; visible everywhere at once
Peak-period working	Queues, file locking, batch delays	Concurrent transactions, online self-service

Cross-department report	Special merging programs; late and contested	Query or dashboard on integrated data
Privacy and accountability	Broad file access; little record of use	Role-based access, views and audit trail
After a failure	Files in mismatched states; manual reconstruction	Log-based recovery to a consistent state

Balance requires noting the responsibilities that come with the improvement. A central database is a critical asset: the institution must invest in competent database administration, proper hardware or cloud provision, security management, backup testing and staff training, and must plan migration from the old files carefully, cleaning and reconciling data as it is moved. A badly designed or badly administered DBMS can disappoint just as a file system can. These are, however, manageable professional tasks, and they are the price of a capability the file approach cannot offer at any price: one accurate, shared, secure and queryable record of the institution's business.

Conclusion

For a bank or a university, a traditional file system multiplies every data management challenge: people and transactions are fragmented across departmental files, copies diverge, peaks overwhelm file-based working, privacy and recovery are weak, and institution-wide questions go unanswered. Implementing a DBMS replaces fragmentation with one integrated record, maintained under constraints and transactions, accessed concurrently through role-based views and services. Operational efficiency rises because retrieval, document production

and self-service all work from the same current data; data consistency rises because each fact is stored once and protected by system-enforced rules; and user experience improves for students, customers and staff alike, in speed, in reach and in justified trust in what the system tells them.

USMAN NOTES
www.usmannotes.com