B.E IV Sem CSE (AICTE) Main Examination-December2020
(Code: 2947/AICTE)
Database Management Systems

PART-A (Answer any five)

1) 2 marks

Data Abstraction: *A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained. For the system to be usable, it must retrieve data efficiently. There are several levels of abstraction:*

1. **Physical Level:**
2. **Logical Level:**
3. **View Level:**

OR

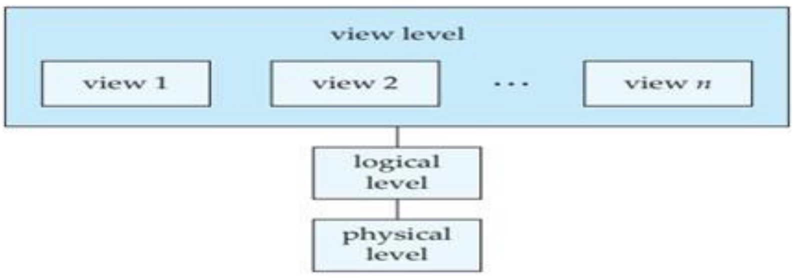


Figure The three levels of data abstraction.

2) 2 marks

CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

3) 2 marks

Weak Entity Sets: An entity set that is dependent on other entity set (or) an entity set that does not have a primary key is referred to as a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

4) 2 marks

Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check** (*P*), where *P* is a predicate

5) 2 marks

Transitive Dependency: When an indirect relationship causes functional dependency it is called Transitive Dependency.
If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

6) 2 marks

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

7) 2 marks

In static hashing, function *h* maps search-key values to a fixed set of *B* of bucket addresses. Databases grow or shrink with time.

8) 2 marks

Shadow Paging is recovery technique that is used to recover database. In this recovery technique, database is considered as made up of fixed size of logical units of storage which are referred as **pages.** pages are mapped into physical blocks of storage, with help of the **page table** which allow one entry for each logical page of database. This method uses two page tables named **current page table** and **shadow page table**.
The entries which are present in current page table are used to point to most recent database pages on disk. Another table i.e., Shadow page table is used when the transaction starts which is copying current page table. After this, shadow page table gets saved on disk and current page table is going to be used for transaction. Entries present in current page table may be changed during execution but in shadow page table it never get changed. After transaction, both tables become identical.

9) 2 marks

The Two-Phase Locking Protocol: This is a protocol which ensures conflict- serializable schedules.

Phase 1: Growing Phase: The transaction may obtain locks and transaction may not release locks.

Phase 2: Shrinking Phase: The transaction may release locks and transaction may not obtain locks.

Rigorous two-phase locking is even stricter: here *all* locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

10) 2 marks

Log-Based Recovery:

After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

PART-B (Answer any four)

11 a) 10 Marks

Diagram and description of each component

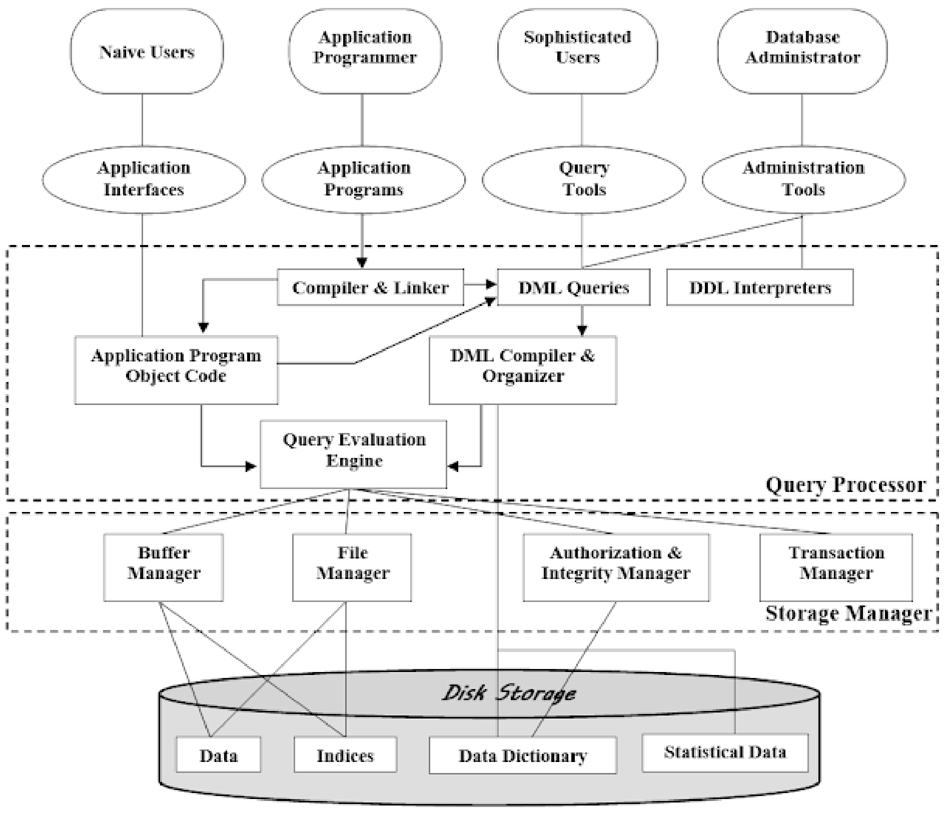


Figure: System Architecture

11 b) 5 marks

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

1) Naïve Users

2) Application Programmers

3) Sophisticated Users

4) Specialized Users

12 a) 8 marks

Expected explanation of the following

Fundamental Relational algebra operations:

SELECT (sigma)
PROJECT (π)
UNION (U)
SET DIFFERENCE (-)
CARTESIAN PRODUCT (X)
RENAME (ρ)

12 b) 7 marks

Expected answer

a) If a student has not taken any course

13 a) 8 marks

Authorization: We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of authorization. (The most common authorization is: *read authorization*, which allows reading, but not modification of data; *insert authorization*, which allows insertion of new data, but not modification of existing data; *update authorization*, which allows modification, but not deletion, of data; and *delete authorization*, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization).

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema:

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.

Authorization Specification in SQL

The **grant** statement is used to confer authorization

grant <privilege list> **on** <relation name or view name> **to** <user list>

<user list> is:

- a user-id
- **public**, which allows all valid users the privilege granted
- A role

Granting a privilege on a view does not imply granting any privileges on the underlying relations. The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Example

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users *U*1, *U*2, and *U*3 **select** authorization on the *branch* relation: **grant select on** department **to** *U*1, *U*2, *U*3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

13 b) 7 marks

Boyce-Codd Normal Form(BCNF):

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in *F+* of the form

$\alpha \rightarrow \beta$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

1. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
2. α is a superkey for *R*

Example schema *not* in BCNF: *instr_dept* (*ID, name, salary, dept_name, building, budget*) because *dept_name*→

building, budget holds on *instr_dept*, but *dept_name* is not a super key **Decomposing a Schema into BCNF**

Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

We decompose *R* into:

1. (α U β)
2. (*R* - (β - α))

In our example,

α = *dept_name*

β = *building, budget*

and *inst_dept* is replaced by

- (α U β) = (*dept_name, building, budget*)
- (*R* - (β - α)) = (*ID, name, salary, dept_name*)

Student can give any example of BCNF

14 a) 8 marks

Static Hashing

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block).
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function.**
- Hash function *h* is a function from the set of all search-key values *K* to the set of all bucket addresses *B.*
- Hash function is used to locate records for access, insertion as well as deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

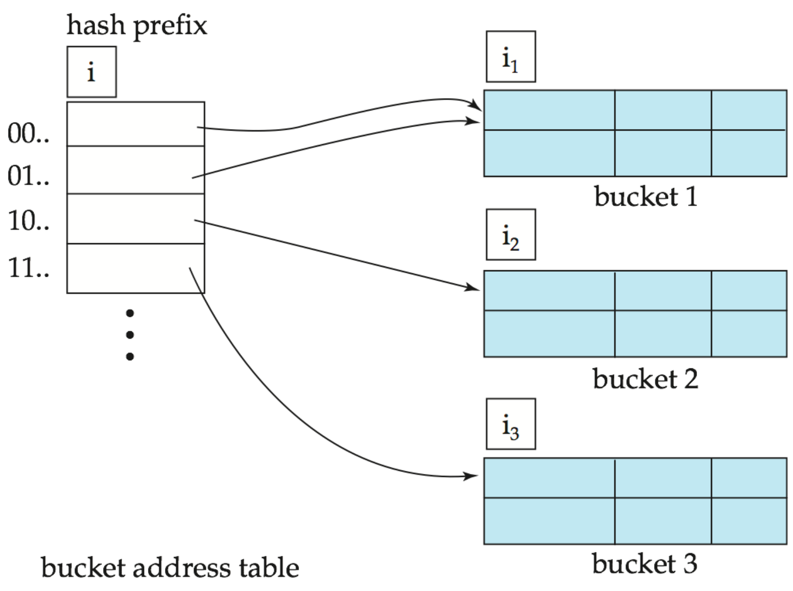
Example of Hash File Organization

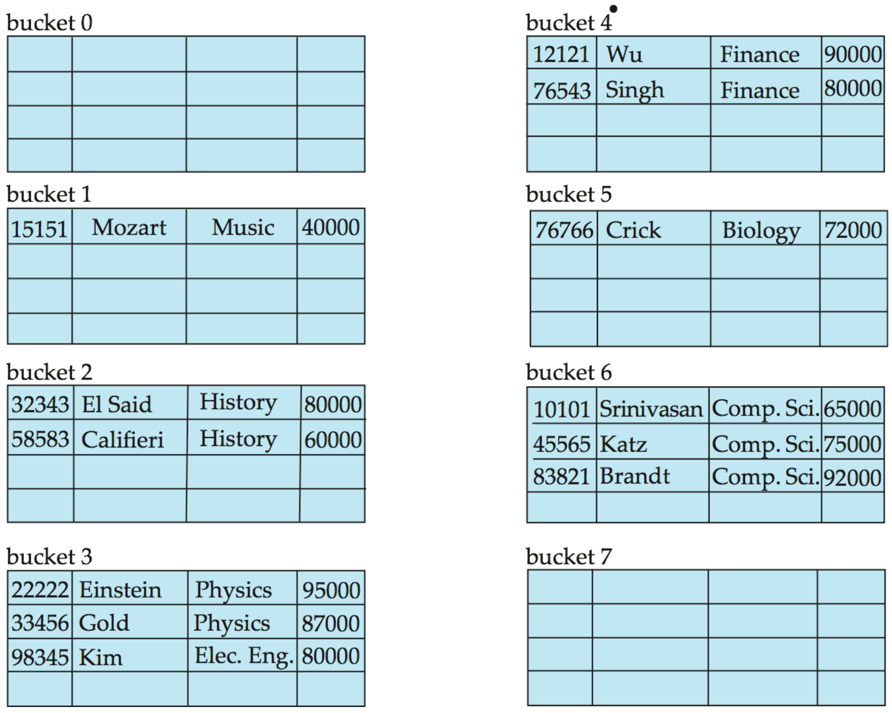
Hash file organization of *instructor* file, using *dept_name* as key There are 10 buckets,

- The binary representation of the *i*th character is assumed to be the integer *i.*
- The hash function returns the sum of the binary representations of the characters modulo 10.

  EX: h(Music) = 1 h(History) = 2

  h(Physics) = 3 h(Elec. Eng.) = 3




bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hash file organization of *instructor* file, using *dept_name* as key

Dynamic Hashing

- Good for database that grows and shrinks in size
- Allows the hash function to be modified dynamically
- **Extendable hashing** – one form of dynamic hashing
    - Hash function generates values over a large range, typically *b*-bit integers, with *b*= 32.
    - At any time use only a prefix of the hash function to index into a table of bucket addresses.
    - Let the length of the prefix be *i* bits, $0 \leq i \leq 32$.
    - Bucket address table size = 2i. Initially *i* = 0
    - Value of *i* grows and shrinks as the size of the database grows and shrinks.
    - Multiple entries in the bucket address table may point to a bucket
    - Thus, actual number of buckets is < 2*i*
        - The number of buckets also changes dynamically due to coalescing and splitting of buckets.

General Extendable Hash Structure

In this structure, $i2 = i3 = i$, whereas $i1 = i - 1$

Use of Extendable Hash Structure

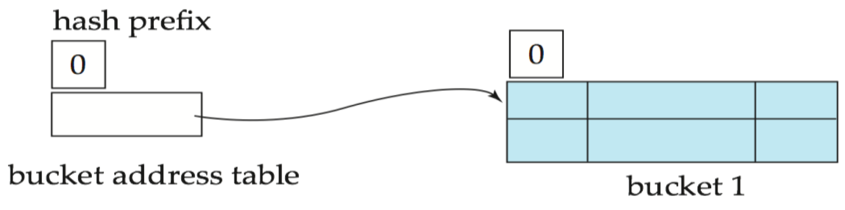
- Each bucket *j* stores a value i_j
  - *All* the entries that point to the same bucket have the same values on the first *ij* bits.
- To locate the bucket containing search-key K_j:
    - Compute *h(Kj) = X*
    - Use the first *i* high order bits of *X* as a displacement into bucket address table, and follow the pointer to appropriate bucket
- To insert a record with search-key value *Kj*
  - follow same procedure as look-up and locate the bucket, say *j*.
  - If there is room in the bucket *j* insert record in the bucket.
  - Else the bucket must be split and insertion re-attempted.
    - Overflow buckets used instead in some cases

Example

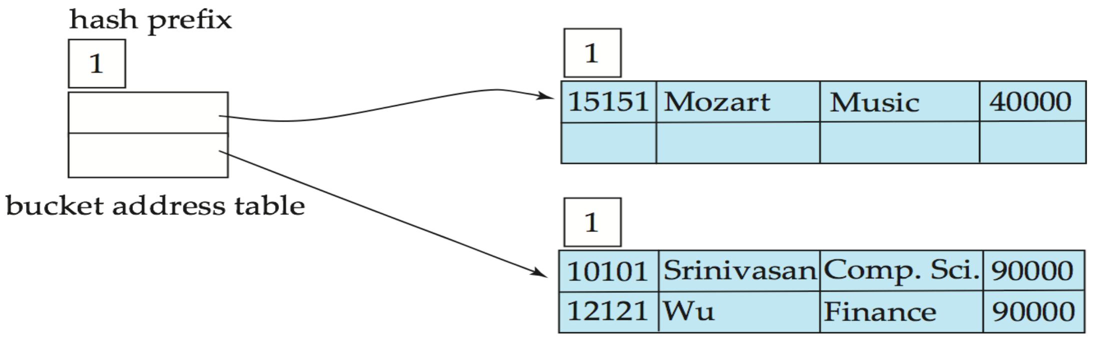
Use of Extendable Hash Structure: Example

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

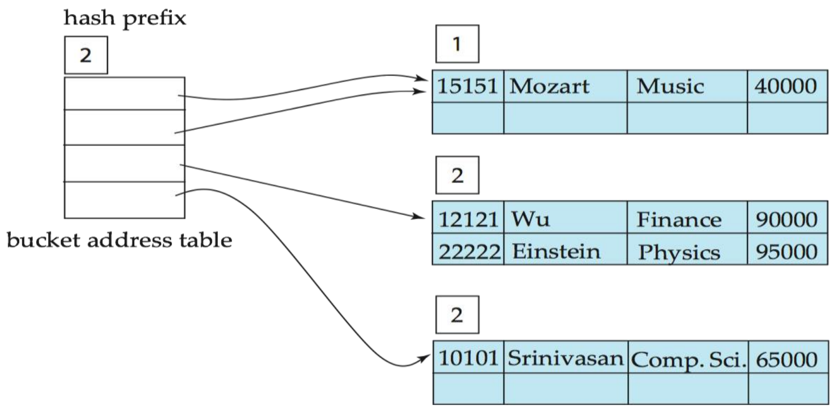
Initial Hash structure, bucket size = 2



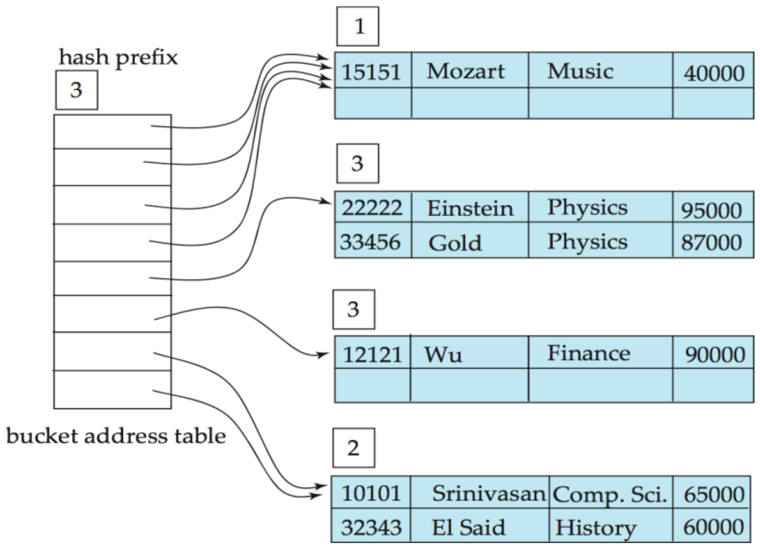
Hash structure after insertion of "Mozart", "Srinivasan", and "Wu" records



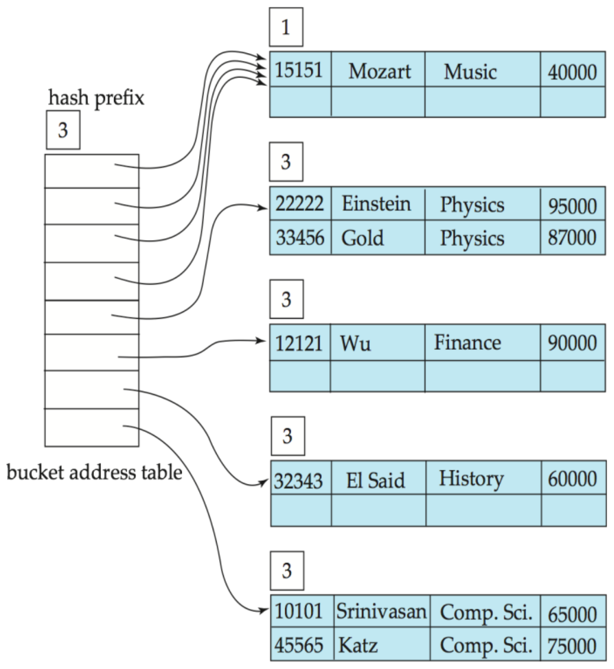
Hash structure after insertion of Einstein record



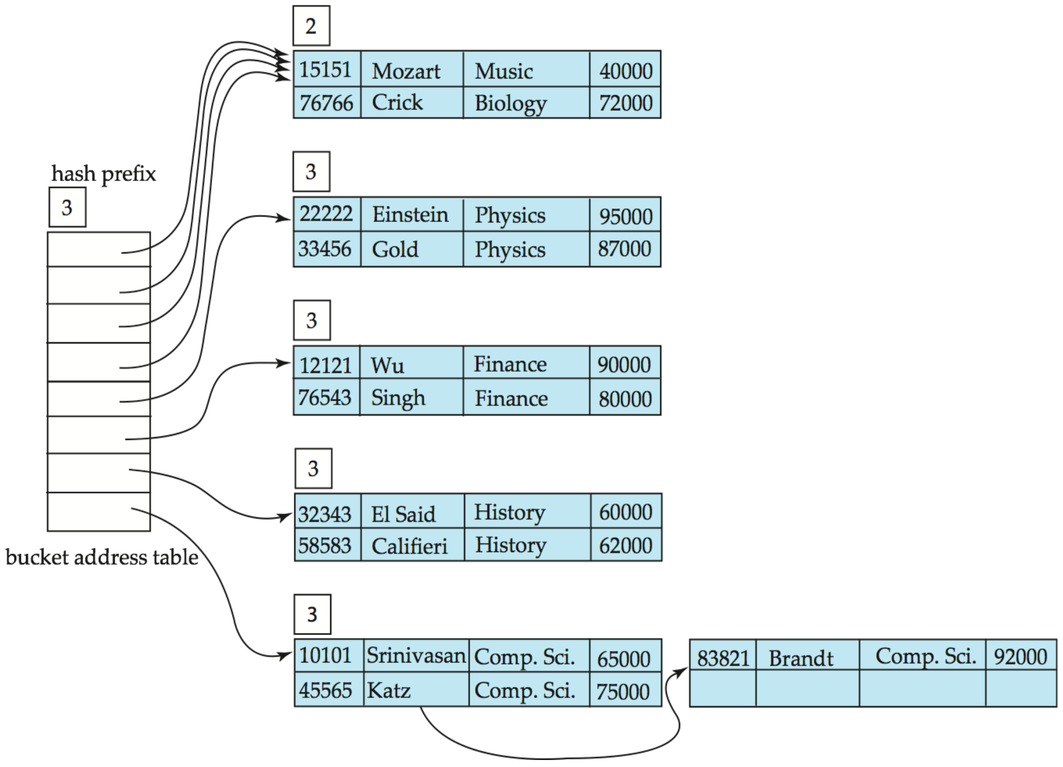
Hash structure after insertion of Gold and El Said records



Hash structure after insertion of Katz record


hash prefix
3
bucket address table
1
15151 Mozart Music 40000
3
22222 Einstein Physics 95000
33456 Gold Physics 87000
3
12121 Wu Finance 90000
3
32343 El Said History 60000
3
10101 Srinivasan Comp. Sci. 65000
45565 Katz Comp. Sci. 75000

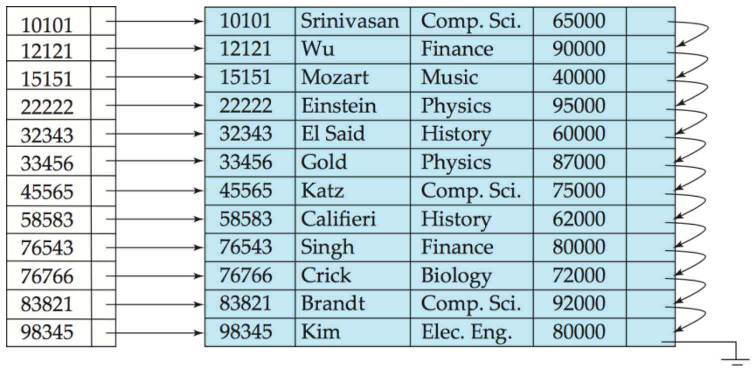
And after insertion of eleven records


hash prefix
3
bucket address table
2
15151 Mozart Music 40000
76766 Crick Biology 72000
3
22222 Einstein Physics 95000
33456 Gold Physics 87000
3
12121 Wu Finance 90000
76543 Singh Finance 80000
3
32343 El Said History 60000
58583 Califieri History 62000
3
10101 Srinivasan Comp. Sci. 65000
45565 Katz Comp. Sci. 75000
83821 Brandt Comp. Sci. 92000

14 b) 7 marks

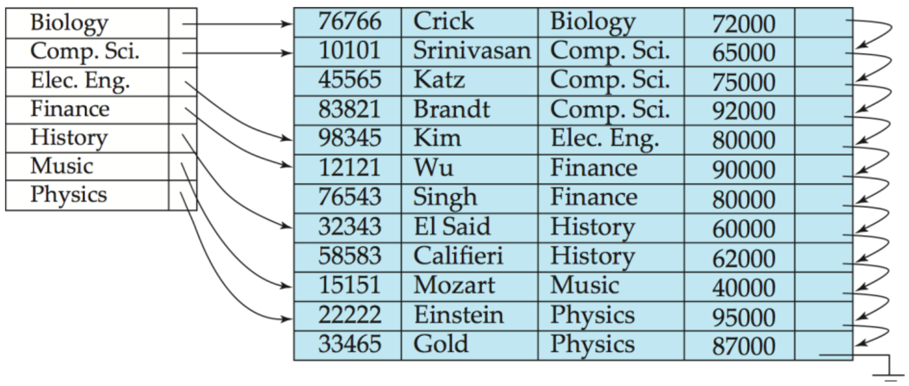
Dense index — Index record appears for every search-key value in the file.

EX: Index on *ID* attribute of *instructor* relation



10101	10101	Srinivasan	Comp. Sci.	65000
12121	12121	Wu	Finance	90000
15151	15151	Mozart	Music	40000
22222	22222	Einstein	Physics	95000
32343	32343	El Said	History	60000
33456	33456	Gold	Physics	87000
45565	45565	Katz	Comp. Sci.	75000
58583	58583	Califieri	History	62000
76543	76543	Singh	Finance	80000
76766	76766	Crick	Biology	72000
83821	83821	Brandt	Comp. Sci.	92000
98345	98345	Kim	Elec. Eng.	80000

Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

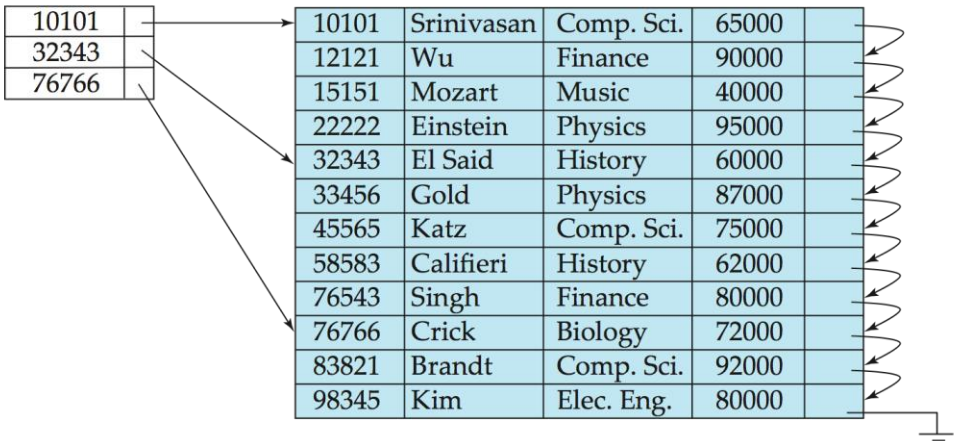


Biology	76766	Crick	Biology	72000
Comp. Sci.	10101	Srinivasan	Comp. Sci.	65000
Elec. Eng.	45565	Katz	Comp. Sci.	75000
Finance	83821	Brandt	Comp. Sci.	92000
History	98345	Kim	Elec. Eng.	80000
Music	12121	Wu	Finance	90000
Physics	76543	Singh	Finance	80000
	32343	El Said	History	60000
	58583	Califieri	History	62000
	15151	Mozart	Music	40000
	22222	Einstein	Physics	95000
	33465	Gold	Physics	87000

Sparse Index: contains index records for only some search-key values. It is Applicable when records are sequentially ordered on search-key

To locate a record with search-key value *K* we:

- Find index record with largest search-key value < *K*
- Search file sequentially starting at the record to which the index record points



10101	10101	Srinivasan	Comp. Sci.	65000
32343	12121	Wu	Finance	90000
76766	15151	Mozart	Music	40000
	22222	Einstein	Physics	95000
	32343	El Said	History	60000
	33456	Gold	Physics	87000
	45565	Katz	Comp. Sci.	75000
	58583	Califieri	History	62000
	76543	Singh	Finance	80000
	76766	Crick	Biology	72000
	83821	Brandt	Comp. Sci.	92000
	98345	Kim	Elec. Eng.	80000

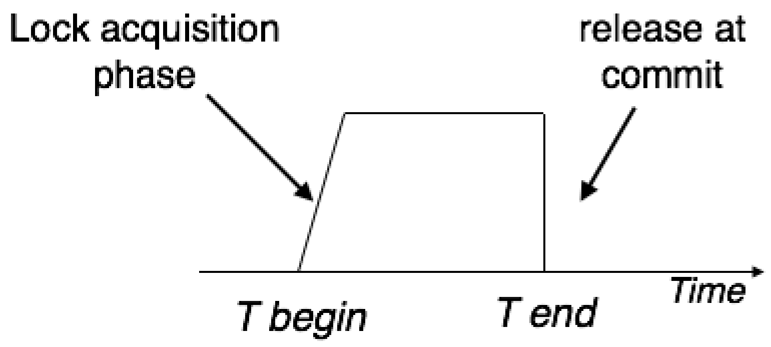
Compared to dense indices:

- Less space and less maintenance overhead for insertions and deletions.
- Generally slower than dense index for locating records.

15 a) 5 marks

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

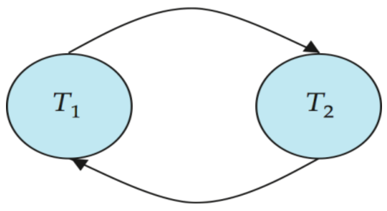


Strict-2PL does not have cascading abort as 2PL does.

15 b) 10 marks

Testing for Serializability

- Consider some schedule of a set of transactions *T*1, *T*2, ..., *Tn*
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from *Ti* to *Tj* if the two transaction conflict and *Ti* accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.



Example

Example Schedule (Schedule A) + Precedence Graph

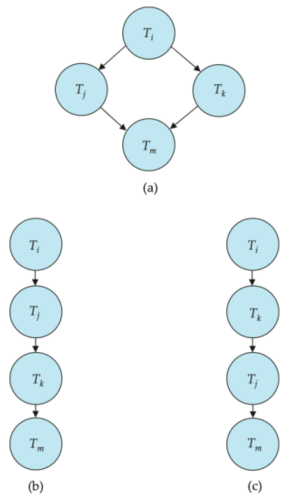
T1	*T2*	*T3*	*T4*	*T5*

	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				

Test for Conflict Serializability

A schedule is conflict serializable if and only if its precedence graph is acyclic.

- Cycle-detection algorithms exist which take order *n*2 time, where *n* is the number of vertices in the graph.
    - (Better algorithms take order *n* + *e* where *e* is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph.
    - This is a linear order consistent with the partial order of the graph.
    - For example



16 a) 8 marks

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

16 b) 7 marks

Explain ARIES algorithm.
Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is based on the Write Ahead Log (WAL) protocol. Every update operation writes a log record which is one of the following :

1. **Undo-only log record:**
2. **Redo-only log record:**
3. **Undo-redo log record:**
   In it, every log record is assigned a unique and monotonically increasing log sequence number (LSN). Every data page has a page LSN field that is set to the LSN of the log record corresponding to the last update on the page. WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk. For performance reasons, each log write is not immediately forced to disk. A log tail is maintained in main memory to buffer log writes. The log tail is flushed to disk when it gets full. A transaction cannot be declared committed until the commit log record makes it to disk.

Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table and the dirty page table. A master log record is maintained separately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk. On restart, the recovery subsystem reads the master log record to find the checkpoint's LSN, reads the checkpoint record, and starts recovery from there on.

The recovery process actually consists of 3 phases:

1. **Analysis:**
   The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.
2. **Redo:**
   Starting at the earliest LSN, the log is read forward and each update redone.
3. **Undo:**
   The log is scanned backward and updates corresponding to loser transactions are undone.

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

17 a) 4 marks

Weak Entity Sets: An entity set that is dependent on other entity set (or) an entity set that does not have a primary key is referred to as a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying** or **owner entity set**. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**. The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and the participation of the weak e

17 b) 7 marks

Aggregation: One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, consider the ternary relationship proj_guide, between an instructor, student and project. Now suppose that each instructor guiding a student on a project is required to file a monthly evaluation report. We model the evaluation report as an entity evaluation, with a primary key evaluation id. One alternative for recording the (student, project, instructor) combination to which an evaluation corresponds is to create a quaternary (4-way) relationship set eval_for between instructor, student, project, and evaluation.

Using the basic E-R modeling constructs, we obtain the following E-R diagram.

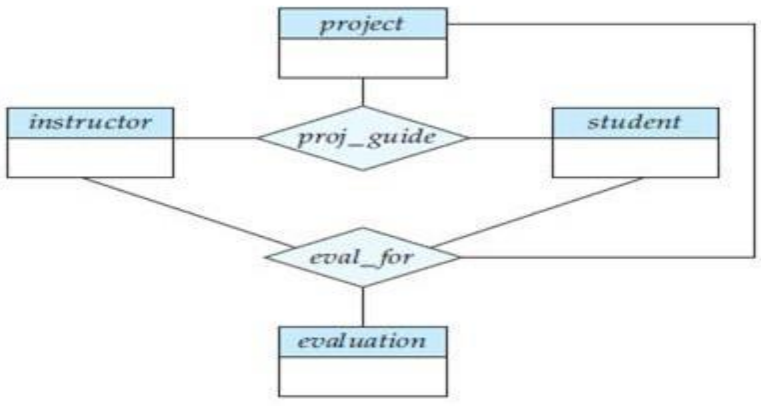


Figure E-R diagram with redundant relationships.

It appears that the relationship sets proj_guide and eval_for can be combined into one single relationship set. Nevertheless, we should not combine them into a single relationship, since some instructor, student, project combinations may not have an associated evaluation.

There is ***redundant information*** in the resultant figure, however, since every instructor, student, project combination in eval_for must also be in proj_guide. The best way to model a situation such as the one just described is to use aggregation. **Aggregation** is an abstraction through which relationships are treated as higher-level entities. Thus, for our example, we regard the relationship set proj_guide (relating the entity sets instructor, student, and project) as a higher-level entity set called proj_guide. Such an entity set is treated in the same manner as is any other entity set.

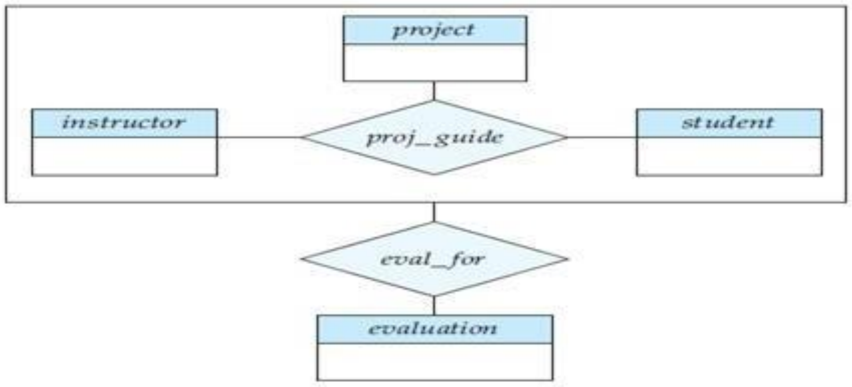


Figure E-R diagram with aggregation.

We can then create a binary relationship eval_for between proj_*guide* and *evaluation* to represent which (*student*, *project*, *instructor*) combination an *evaluation* is for, as shown in the figure

17 c) 4 marks

Assertions: An assertion is *any condition that the database must always satisfy*. Domain constraints and referential integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, “Every department must have at least five courses offered every semester” must be expressed as an assertion.