UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

Database Systems

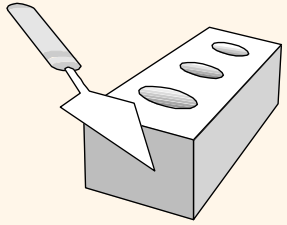
M & W 4:00 to 5:30 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON (must have)

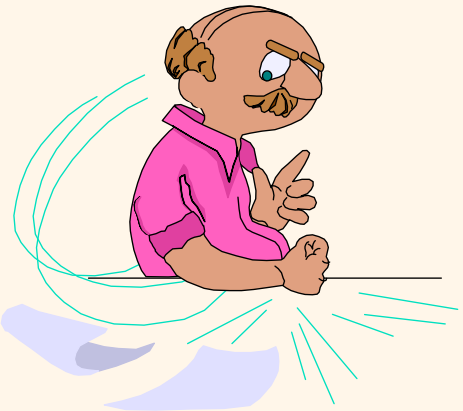
NO CHATTING during LECTURE

VH, unhide Section 13: SET 3 STORAGE I



COSC 3380

4 to 5:30

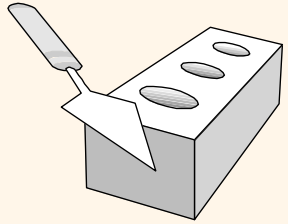


PLEASE
LOG IN
CANVAS

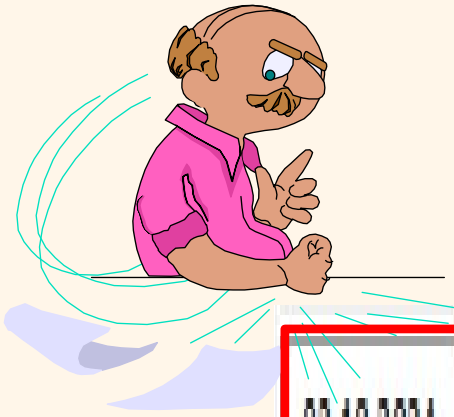
Please close all other windows.

03.18.2024 (16 - Mo)	**ZyBook SET 3 - 1**	**Set 3** LECTURE 12 STORAGE I
03.20.2024 (17 - We)	**ZyBook SET 3 - 2**	**Set 3** LECTURE 13 STORAGE II
03.25.2024 (18 – Mo)	**ZyBook SET 3- 3**	**Set 3** LECTURE 14 STORAGE III B TREE INDEX
03.27.2024 (19 - We)	**ZyBook SET 3- 4**	**Set 3** LECTURE 15 STORAGE IV HASH INDEX
04.01.2024 (20 - Mo)		**EXAM 3 Practice** **(PART of 20 points)**
04.03.2024 (21 – We)	**TA Download** **ZyBook SET 3 Sections** **(4 PM)** **(PART of 30 points)**	**EXAM 3 Review** **(PART of 20 points)**
04.08.2024 (22 - Mo)		**EXAM 3** **(PART of 50 points)**

COSC 3380



Class 16



From 4:00 to 4:07 PM – 5 minutes.

03.18.2024 (16 - Mo)	ZyBook SET 3 - 1	Set 3 LECTURE 12 STORAGE I

CLASS PARTICIPATION 20 points — 20% of Total

STORAGE I

Class 16 BEGIN PARTICIPATION
Not available until Mar 18 at 4:00pm | Due Mar 18 at 4:07pm | 100 pts

VH, publish

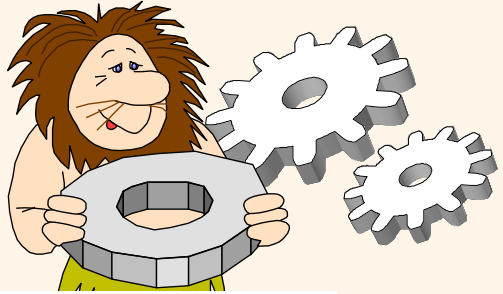
This is a synchronous online class.

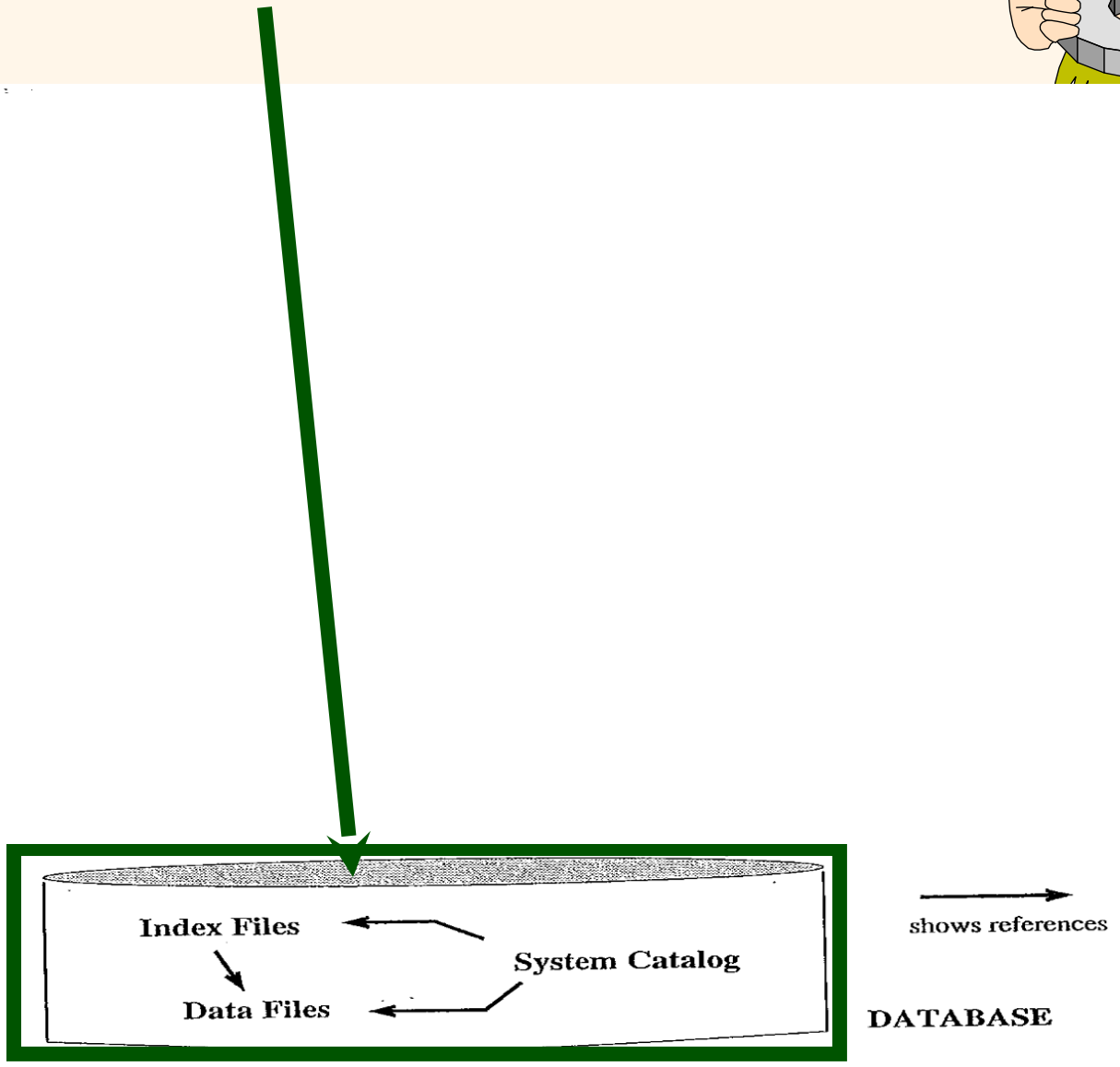
Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

A DBMS

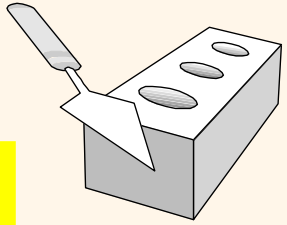




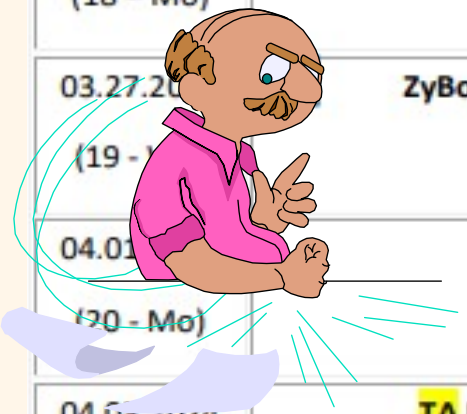
Tables
Indexes

COSC 3380

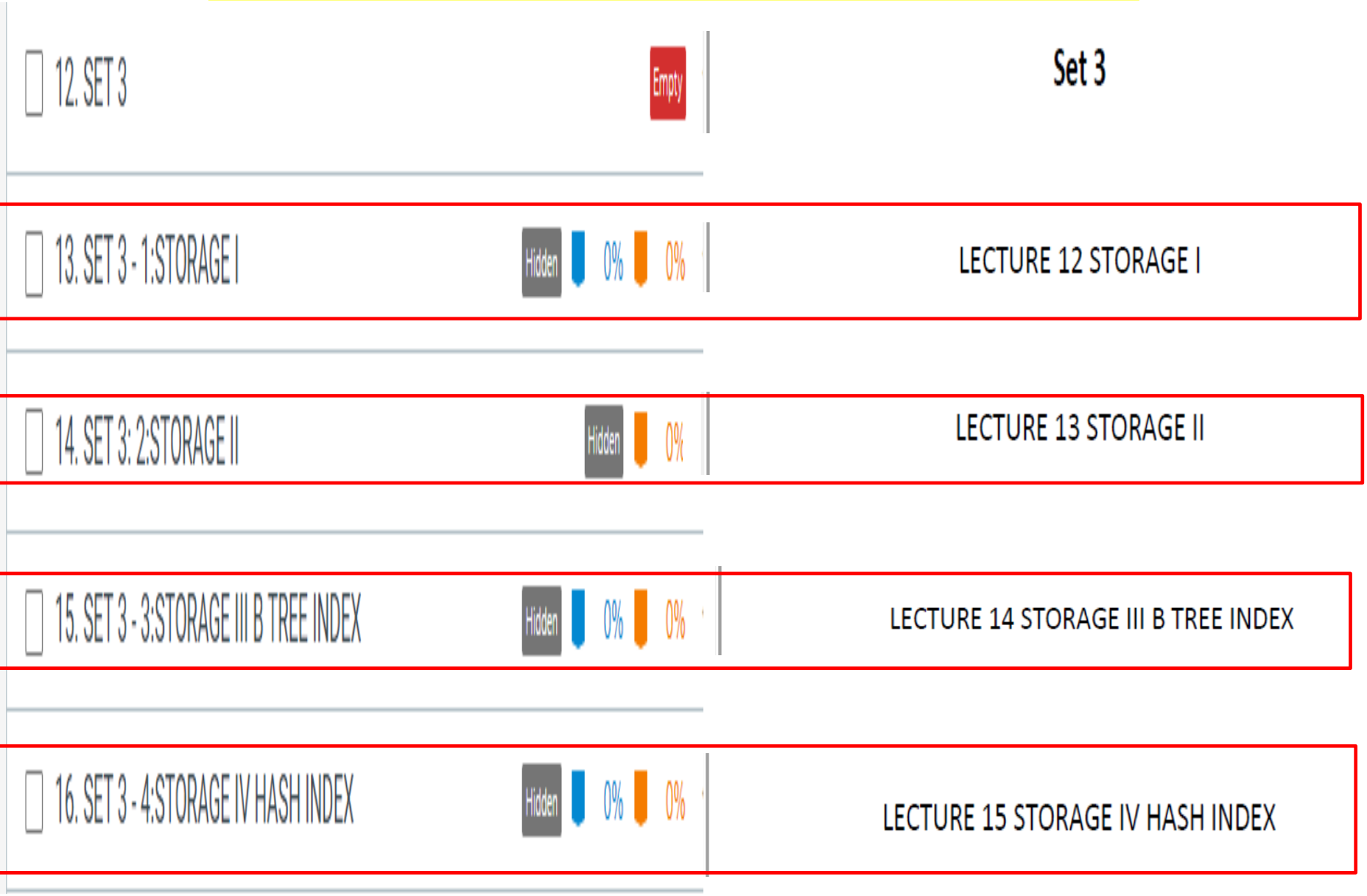
ZyBook SET 3



03.18.2024 (16 - Mo)	ZyBook SET 3 - 1	Set 3 LECTURE 12 STORAGE I
03.20.2024 (17 - We)	ZyBook SET 3 - 2	Set 3 LECTURE 13 STORAGE II
03.25.2024 (18 – Mo)	ZyBook SET 3- 3	Set 3 LECTURE 14 STORAGE III B TREE INDEX
03.27.2[illegible] (19 - [illegible])	ZyBook SET 3- 4	Set 3 LECTURE 15 STORAGE IV HASH INDEX
04.01 (20 - Mo)		EXAM 3 Practice (PART of 20 points)
04.03.2024 (21 – We)	TA Download ZyBook SET 3 Sections (4 PM) (PART of 30 points)	EXAM 3 Review (PART of 20 points)



Mapping ZyBook SET 3 to the Lectures

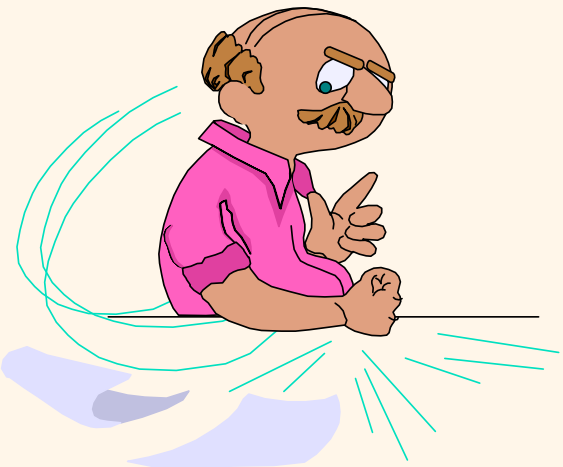


From 4:07 to 5:00 PM – 53 minutes.

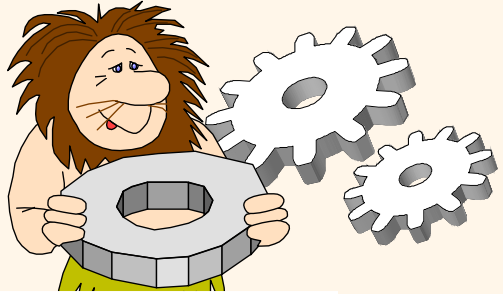
COSC 3380

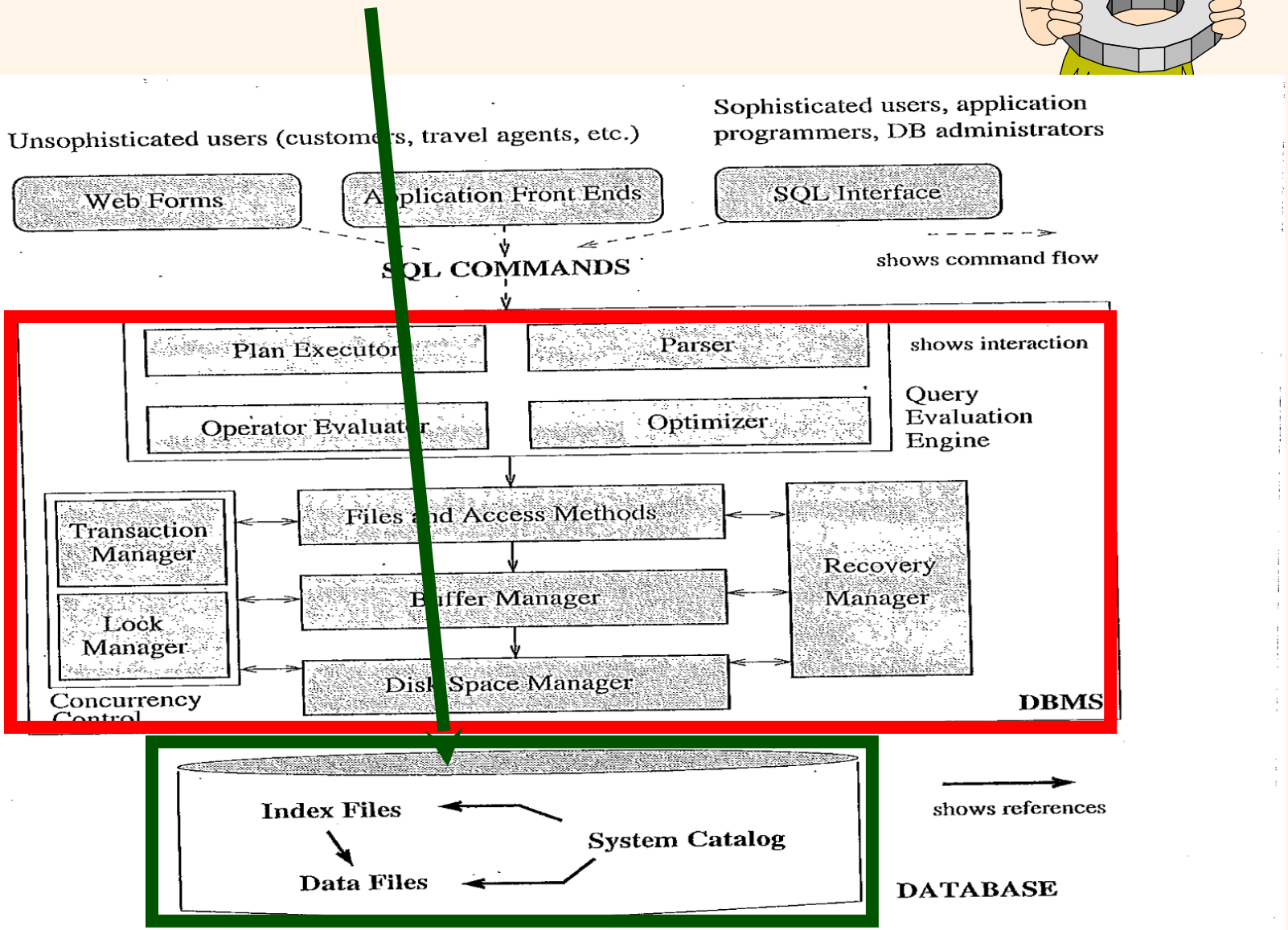
Lecture 12

Overview of Storage and ***Indexing***

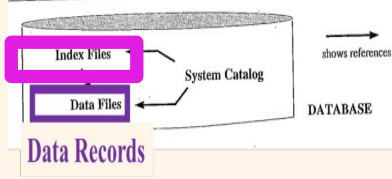


A DBMS




Unsophisticated users (customers, travel agents, etc.)
Sophisticated users, application programmers, DB administrators
Web Forms
Application Front Ends
SQL Interface
SQL COMMANDS
shows command flow
Plan Executor
Parser
Operator Evaluator
Optimizer
shows interaction
Query Evaluation Engine
Transaction Manager
Lock Manager
Concurrency Control
Files and Access Methods
Buffer Manager
Disk Space Manager
Recovery Manager
DBMS
Index Files
Data Files
System Catalog
DATABASE
shows references

*Alternative **File** Organizations (Big Picture)*



Three alternatives exist, ***each ideal for some situations, and***

not so good in others:

- **Heap** (random order) **Files**: Suitable when typical **access** is a **File scan** *retrieving* ***all Data Records***.

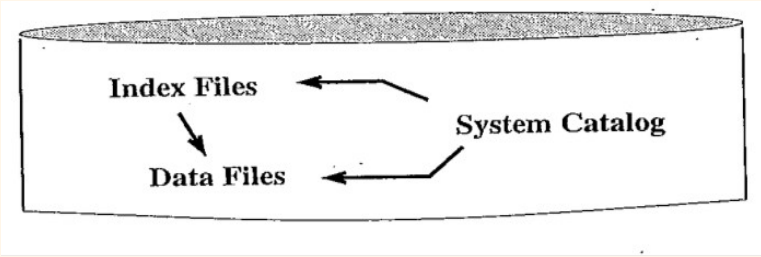
- **Sorted Files**: Best if **Data Records** must be **retrieved in some** *order*, or only a `***range'*** **of Data Records** is needed.

- **Indexes**: **Data structures to organize Data Records via *trees* or *hashing*.**
  - Like **Sorted Files**, they speed up searches for a **subset of Data Records**, based on values in certain ("search key") fields.

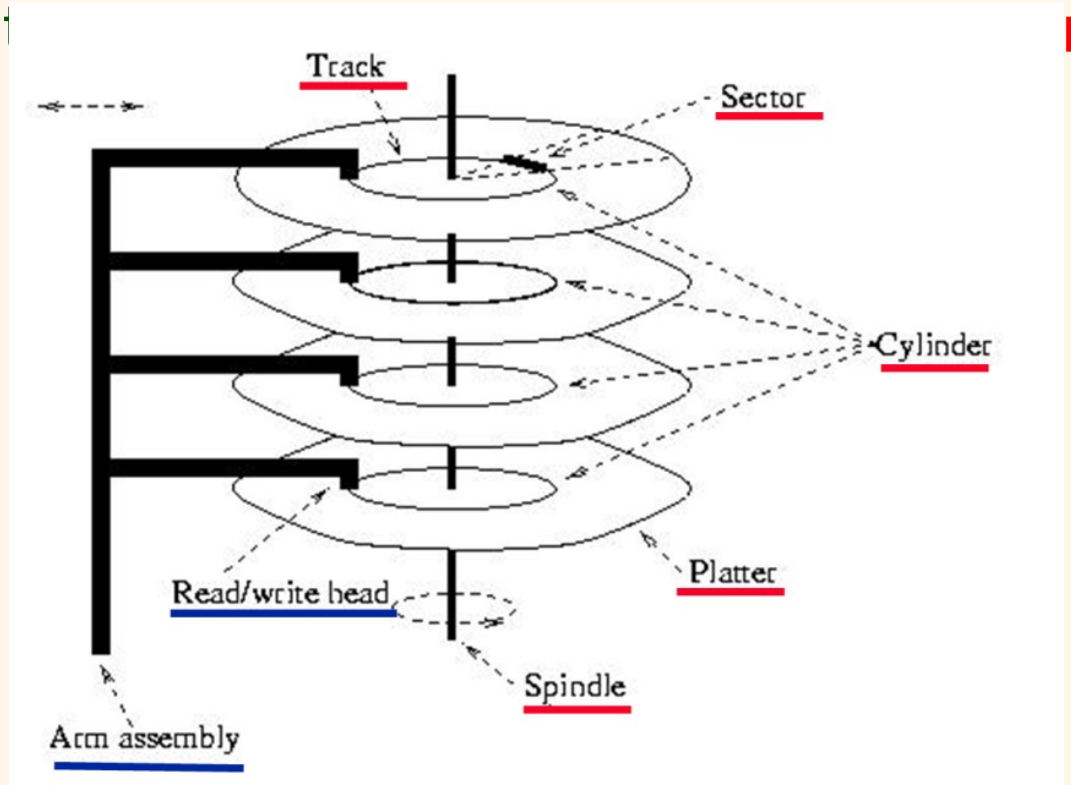
Transcript

StudId	CrsCode	Sem	Grade
666666	MGT123	F1994	4.0
123456	CS305	S1996	4.0
987654	CS305	F1995	2.0

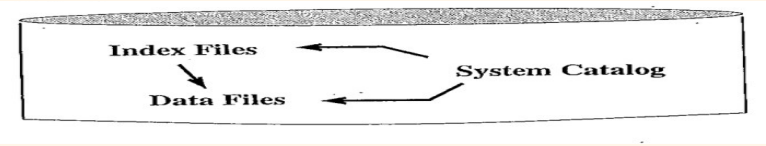
Data on External Sto

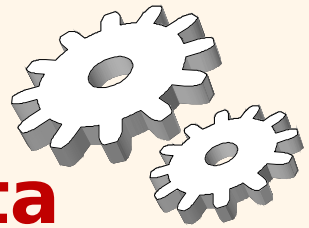


- **Disks** : Can retrieve random **Page** at **fixed cost**
  - But reading **several consecutive Pages is much cheaper** **rder**

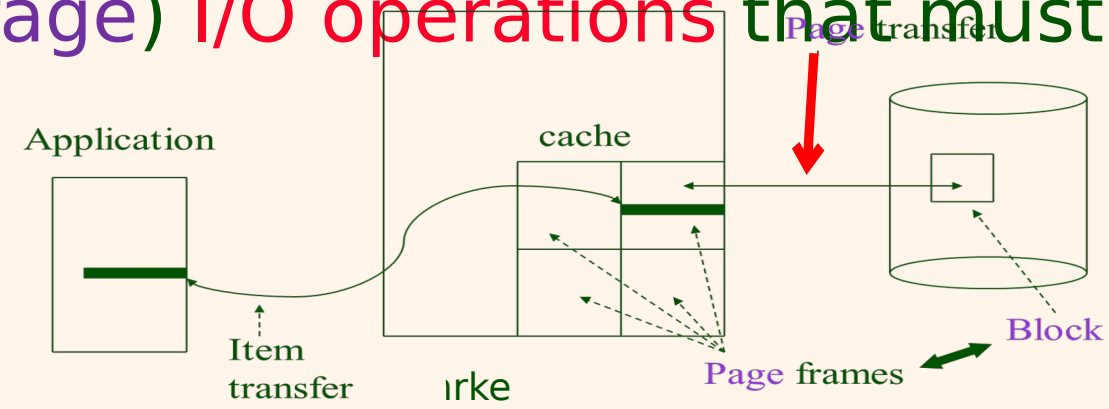


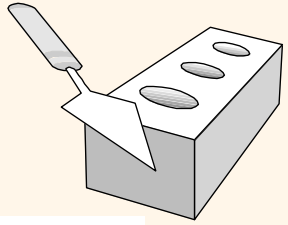
Disks





- Capable of storing large quantities of **Data** ***cheaply***
- **Non-volatile**
- **Extremely slow** compared with **cpu speed**
- Performance of **DBMS** largely a function of the number of **Disk** (Page) I/O operations that must be performed





13. SET 3 - 1:STORAGE I

0% 0%

- 13.1 Storage media Hidden — 0%
- 13.2 Table structures Hidden — 0% 0%
- 13.3 Tablespaces and partitions Hidden — 0% 0%

13.1 Storage media

Storage media

Computers use a variety of media to store data, such as random-access memory, magnetic disk, optical disk, and magnetic tape. Computer media vary on four important dimensions:

- **Speed**. Speed is measured as access time and transfer rate. ***Access time*** is the time required to access the first byte in a read or write operation. ***Transfer rate*** is the speed at which data is read or written, following initial access.
- **Cost**. Cost typically ranges from pennies to dollars per gigabyte of memory, depending on the media type.
- **Capacity**. In principle, any media type can store any amount of data. In practice, capacity is limited by cost. Expensive media have limited capacity compared to inexpensive media.
- **Volatility**. ***Volatile memory*** is memory that is lost when disconnected from power. ***Non-volatile memory*** is retained without power.

Three types of media are most important for database management:

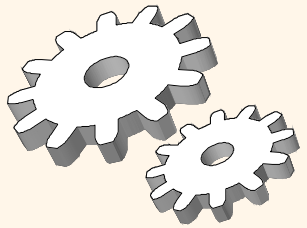
- ***Main memory***, also called ***random-access memory (RAM)***, is the primary memory used when computer programs execute. Main memory is fast, expensive, and has limited capacity.
- ***Flash memory***, also called ***solid-state drive (SSD)***, is less expensive and higher capacity than main memory. Writes to flash memory are much slower than reads, and both are much slower than main memory writes and reads.
- ***Magnetic disk***, also called ***hard-disk drive (HDD)***, is used to store large amounts of data. Magnetic disk is slower, less expensive, and higher capacity than flash memory.

Main memory is volatile. Flash memory and magnetic disk are non-volatile and therefore considered storage media. Databases store data permanently on storage media and transfer data to and from main memory during program execution.

Table 13.1.1: Media characteristics (circa 2018).

Media type	Access time (microseconds)	Transfer rate (gigabyte/second)	Cost ($/gigabyte)	Volatile
Main memory	.01 to .1	> 10	> 1	Yes
Flash memory	20 to 100	.5 to 3	around .25	No
Magnetic disk	5,000 to 10,000	.05 to .2	around .02	No

Accessing Data Through Cache

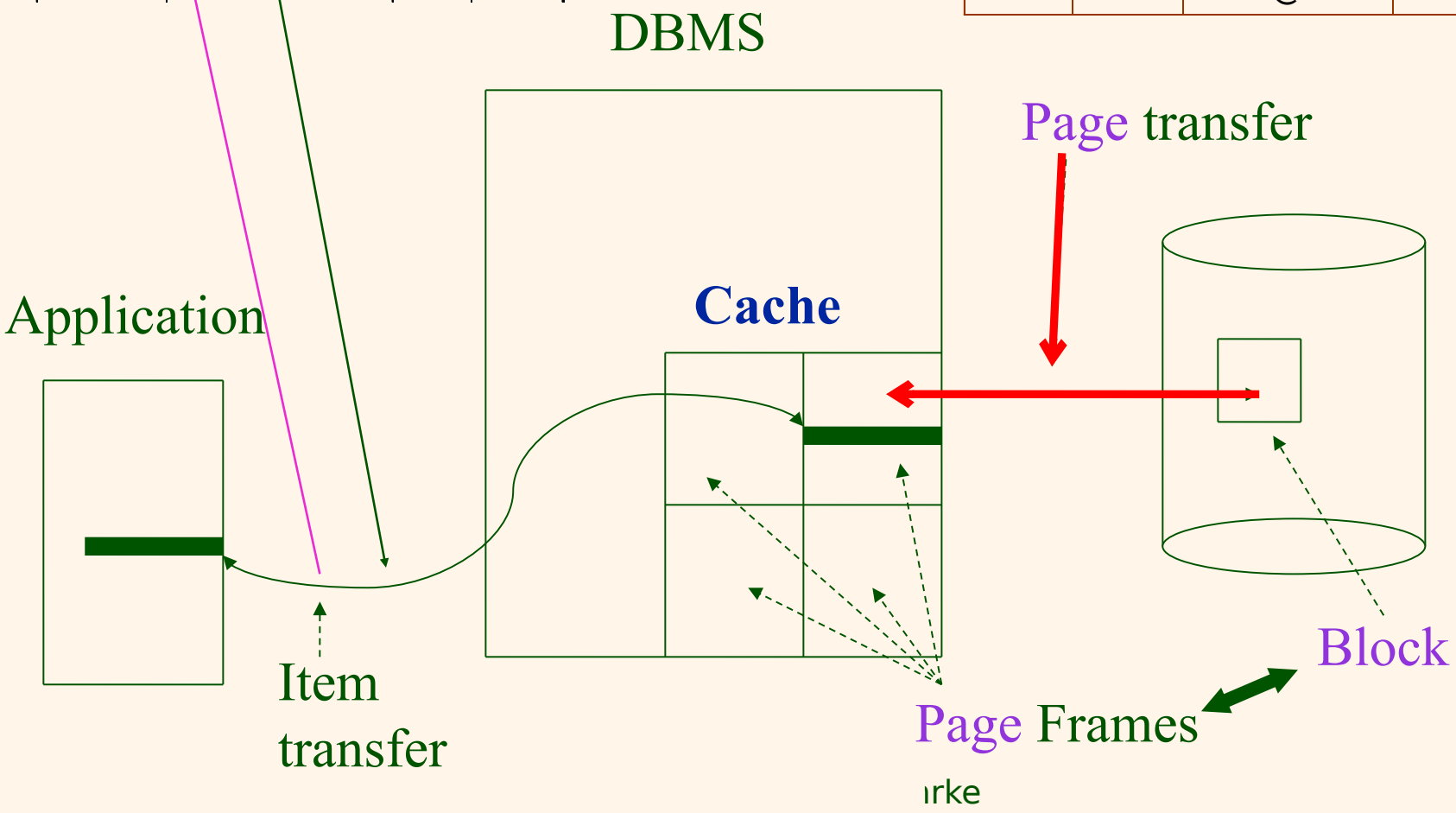


```
SELECT *
FROM Students S
WHERE S.age=19
```

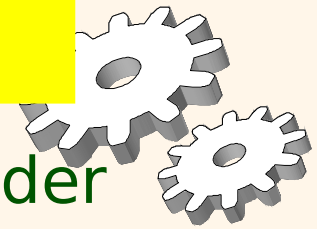
sid	name	login	age	gpa
53650	Smith	smith@math	19	3.8

S1

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8



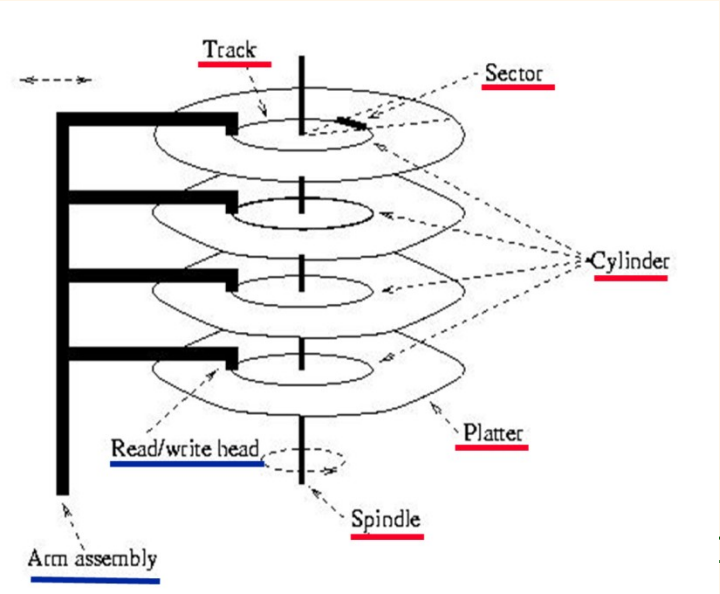
I/O Time to Access a Page (Block)



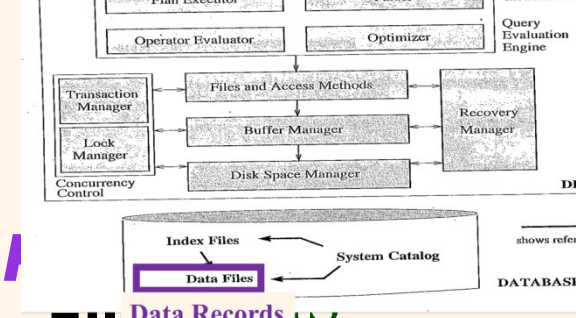
- *Seek latency* – time to position heads over cylinder containing **Page** (avg = **~10 - 20 ms**)
- *Rotational latency* – additional time for platters to rotate so that start of Block containing **Page** is under head (avg = **~5 - 10 ms**)
- *Transfer time* – time for platter to rotate over **Block** containing **Page** (depends on size of **Block**)
- *Latency* = *Seek latency* + *Rotationa*
- Our goal

– **minimize average** *Latency*,

– **reduce number** of **Page** (**Block**) tr

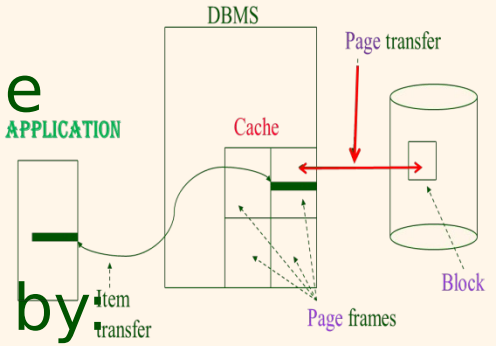


Pages and Blocks



Data Files (Relation/Table) decomposed into *P*

- Fixed size piece of contiguous information in the **File**
- **Unit of exchange** between **Disk** and Main Memory(**Cache**)



Disk divided into **Page** size **Blocks** of storage

- **Page** can be stored in any **Block**

Application's request for read **item** satisfied by:

- Read **Page** containing **item** to Buffer Memory in DBMS
- Transfer **item** from Buffer Memory to Application

Application's request to change **item** satisfied by

- Read **Page** containing **item** to Buffer Memory in DBMS (if it is not alrea there)
- Update **item** in DBMS (Main Memory) Buffer
- (Eventually) copy Buffer **Page** to **Page** on **Disk**

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

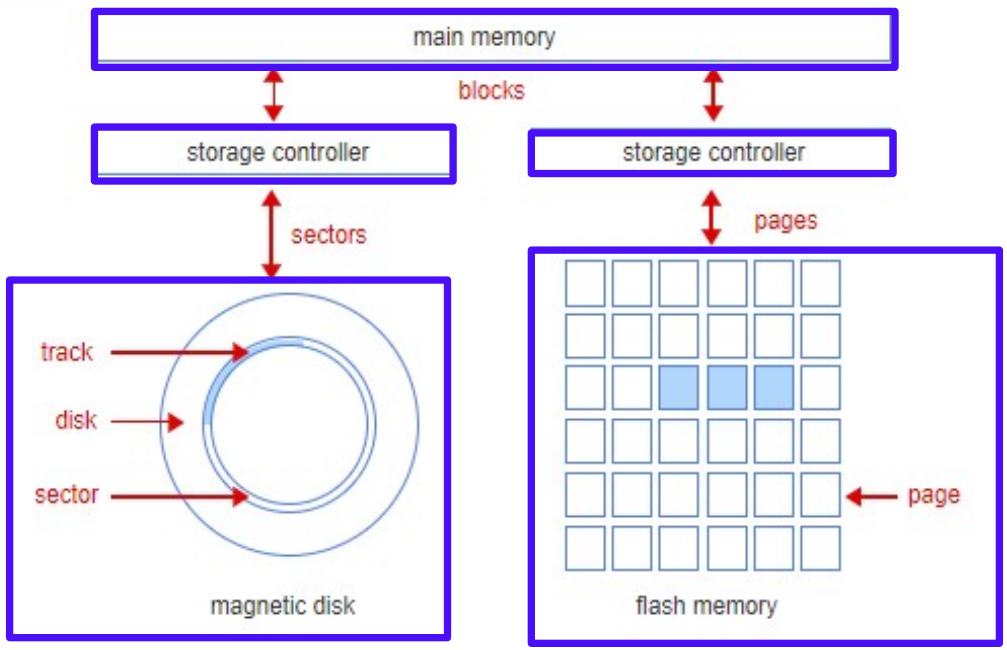
Sectors, pages, and blocks

Magnetic disk groups data in ***sectors***, traditionally 512 bytes per sector but 4 kilobytes with newer disk formats. Flash memory groups data in ***pages***, usually between 2 kilobytes and 16 kilobytes per page.

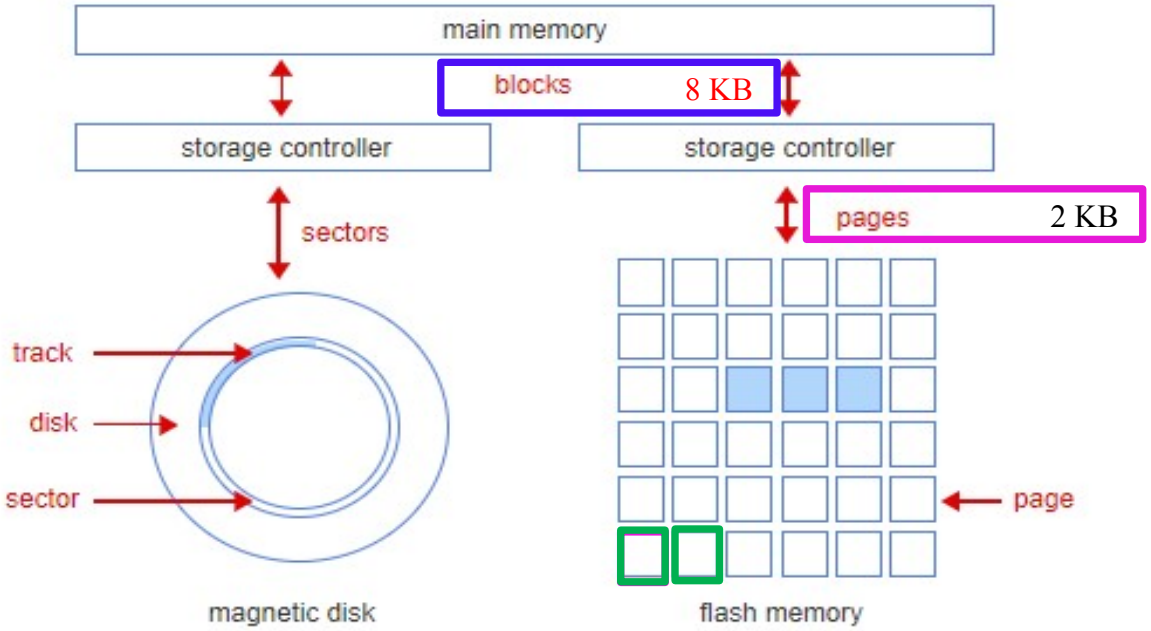
Databases and file systems use a uniform size, called a ***block***, when transferring data between main memory and storage media. Block size is independent of storage media. Storage media are managed by controllers, which convert between blocks and sectors or pages. This conversion is internal to the storage device, so the database is unaware of page and sector sizes.

Block size must be uniform within a database but, in most database systems, can be specified by the database administrator. Database systems typically support block sizes ranging from 2 kilobytes to 64 kilobytes. Smaller block sizes are usually better for transactional applications, which access a few rows per query. Larger block sizes are usually better for analytic applications, which access many rows per query.

PARTICIPATION ACTIVITY 13.1.2: Sectors, pages, and blocks.



Sectors, pages, and blocks.



Sectors, pages, and blocks.

2) The database administrator specifies an eight-kilobyte block size. Flash memory page size is two kilobytes. A user runs a query that reads two pages of flash memory. How many blocks are transferred to main memory?

- ○ One-half
- ◉ One
- ○ Four

Correct

Data is transferred to the database in blocks. Although four kilobytes are read from flash memory, a minimum of one eight-kilobyte block must be transferred into memory.

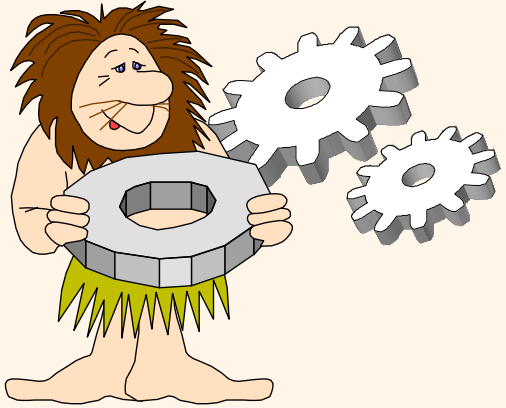
?????

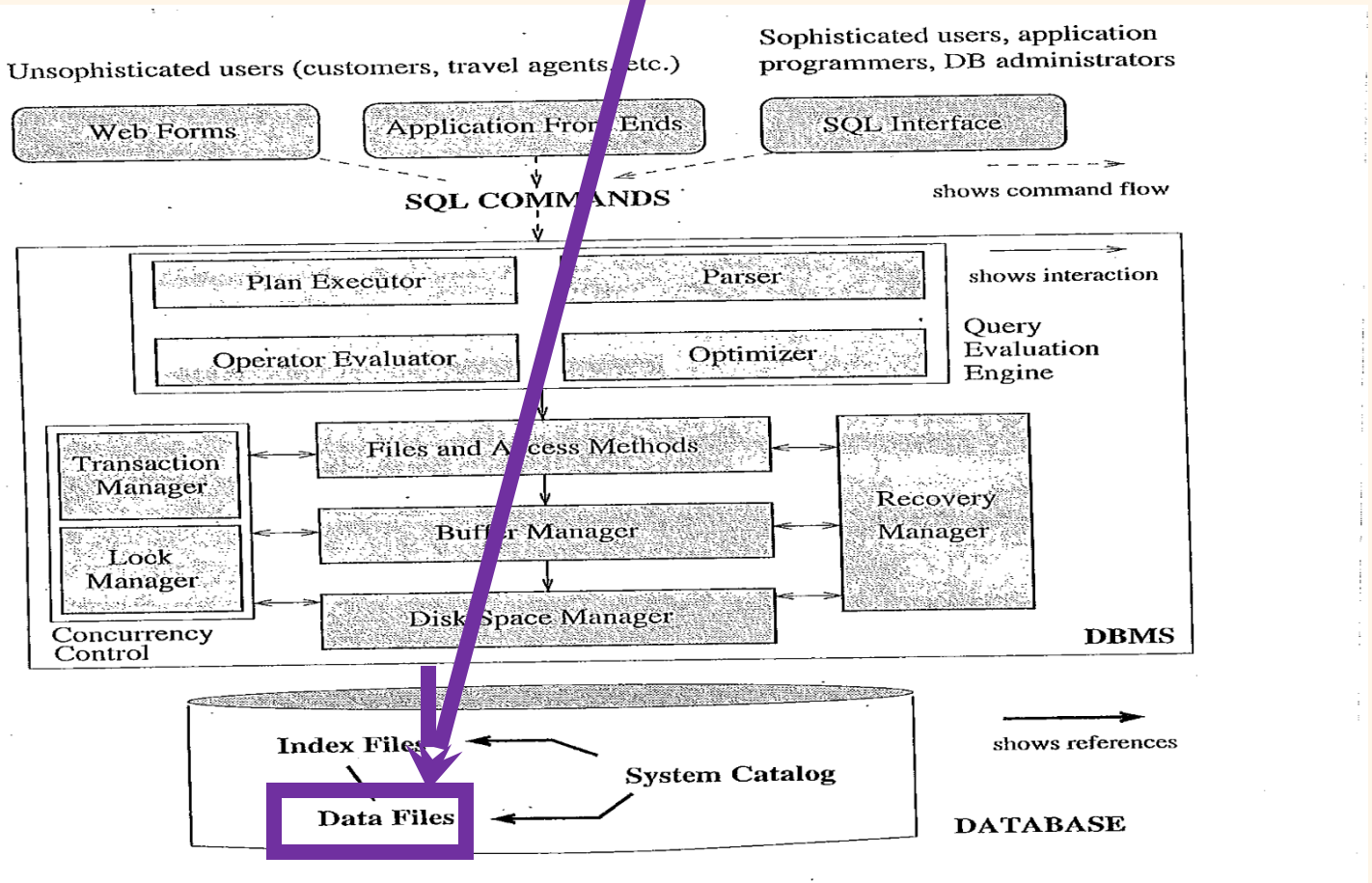
1

A DBMS

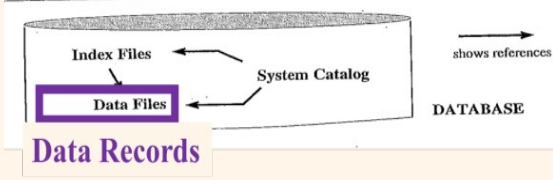
S1

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8





Data on External Storage



- **File organization:** Method of arranging a **File** (**Table/ Relation**) of **Data Records** (**rows/tuples**) on **External Storage**.
  - Record id (**rid**) is sufficient to physically locate **Data Record**
    - **rid** is the pair **<pid, slot#>**
      - **pid** – **Page** number,
      - **slot#** - **record** withing the **Page**

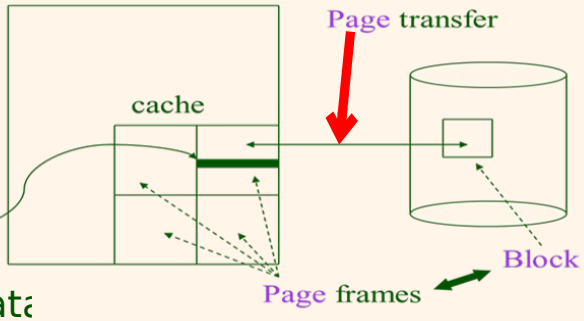
Data record

rid

Transcript

StudId	*CrsCode*	*Sem*	*Grade*
666666	MGT123	F1994	4.0
123456	CS305	S1996	4.0
987654	CS305	F1995	2.0

<0,0>
<0,1>
<0,2>



Row-oriented storage

Accessing storage media is relatively slow. Since data is transferred to and from storage media in blocks, databases attempt to minimize the number of blocks required for common queries.

Most relational databases are optimized for transactional applications, which often read and write individual rows. To minimize block transfers, relational databases usually store an entire row within one block, which is called ***row-oriented storage***.

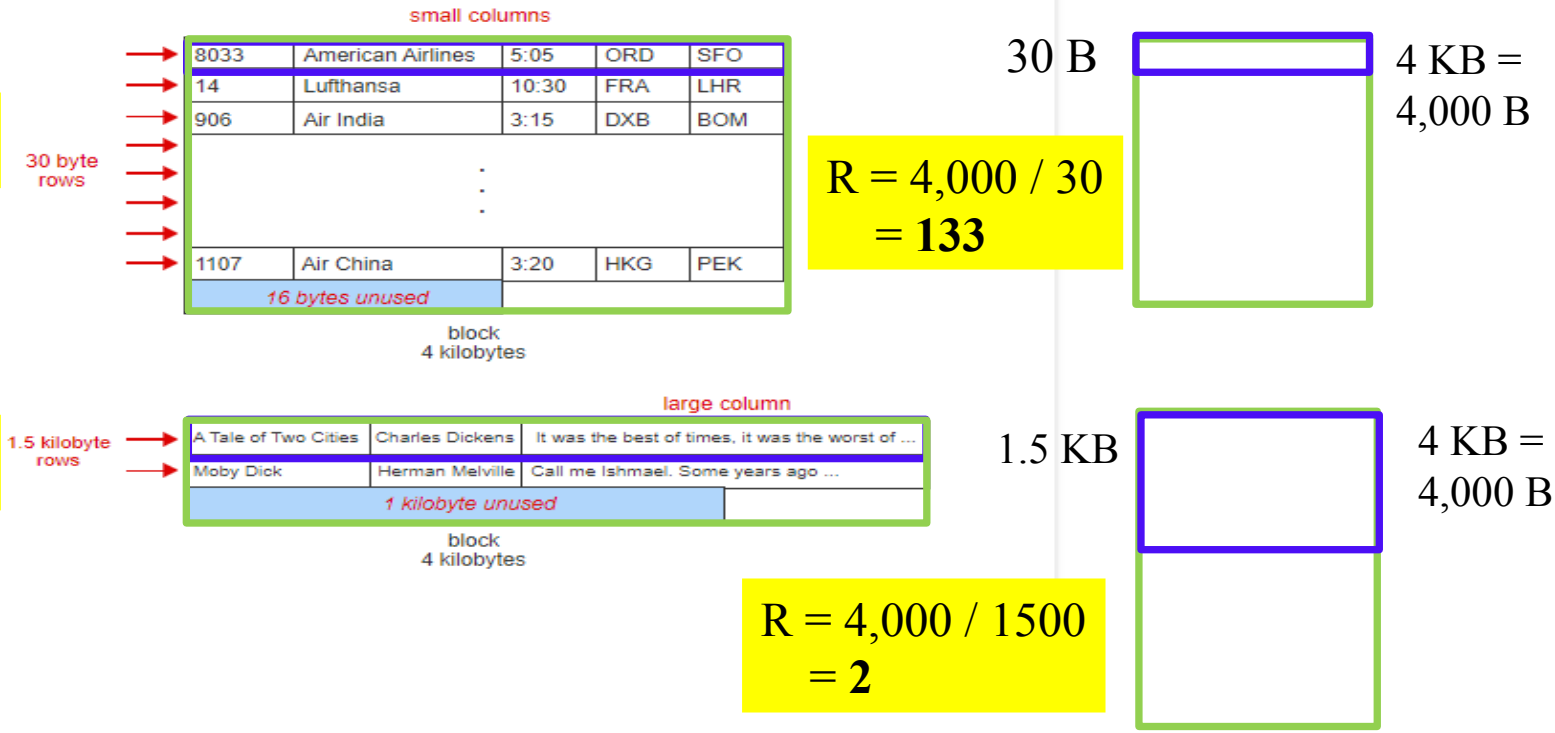
Row-oriented storage performs best when row size is small relative to block size, for two reasons:

- **Improved query performance.** When row size is small relative to block size, each block contains many rows. Queries that read and write multiple rows transfer fewer blocks, resulting in better performance.
- **Less wasted storage.** Row-oriented storage wastes a few bytes per block, since rows do not usually fit evenly into the available space. The wasted space is less than the row size. If row size is small relative to block size, this wasted space is insignificant.

Consequently, database administrators might specify a larger block size for databases containing larger rows.

Sometimes a table contains a very large column, such as 1 megabyte documents or 10 megabyte images. For tables with large columns, each row usually contains a link to the large column, which is stored in a different area. The large column might be stored in files managed by the operating system or in a special storage area managed by the database. This approach keeps row size small and improves performance of queries that do not access the large column.

PARTICIPATION ACTIVITY | 13.1.4: Row-oriented storage.



Row-oriented storage

16 KB /block * 1024 bytes/KB = 16,384 bytes/block

small columns

Data record

8033	American Airlines	5:05	ORD	SFO
14	Lufthansa	10:30	FRA	LHR
906	Air India	3:15	DXB	BOM
. . .				
1107	Air China	3:20	HKG	PEK

100 bytes /data record (row)

16 KB block

16,384 bytes

R = 163 data records (rows)/ block

unused bytes = 16,384 bytes/block – 163,000 bytes used = 84 bytes free

PARTICIPATION ACTIVITY 13.1.6: Row-oriented storage.

1) A database uses 16-kilobyte blocks. A table contains 18,200 rows of 100 bytes each. Ignoring space used by the system for overhead, how many bytes are unused on each block? Assume one kilobyte = 1,024 bytes.

- ○ None
- ◉ 84
- ○ 384

Correct

16 kilobytes = 1,024 x 16 = 16,384 bytes. Since rows are not split across blocks, 16,384 bytes / 100 bytes/row = 163 rows are stored on each block. The number of unused bytes = 16,384 bytes - (163 rows x 100 bytes/row) = 84 bytes.

How many data records (rows) fit in one block = R ?

R = 16,384 bytes/block / 100 bytes/data record (row) = 163 data records /block

TA time (Alvaro)
(CA 13.2.1 Step 5 – Table structure)

CHALLENGE ACTIVITY | 13.2.1: Table structures.

R – number of **data records** per **block**

B – number of BLOCKS

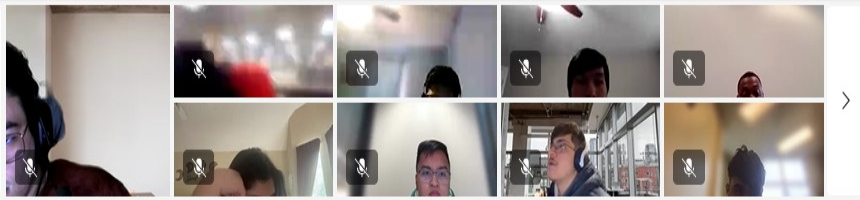
Assume blocks are 5,400 bytes and rows are 60 bytes.

How many blocks would be required to fit 30,670 rows ?

Ex: 5 341

R = 5,400 bytes/block / 60 bytes/row
= 90 rows/block

B = 30,670 rows / 90 rows/block
= 341 blocks



AU

Urtaza, Alvaro A

View all

Participants

Invite someone or dial a number

Share invite

In this meeting (111) Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- RA Aysola, Riya
- SB Banza, Sean Paolo B
- HB Bui, Hieu
- Burger, Jake
- VC Carrillo-Zepeda, Victor E
- DC Carter, Deric

learn.zybooks.com/zybook/UHCOSC3380HilfordSpring2024/chapter/13/section/2

zyBooks My library > COSC 3380: Database Systems home > 13.2: Table structures

zyBooks catalog | Help/FAQ | Alvaro Urtaza

Assume blocks are 3,600 bytes and rows are 40 bytes.

How many blocks would be required to fit 13,575 rows into a single bucket?

151

If only 1 block is used per bucket, how many rows in total can be added to 3 buckets?

270

1 2 3 4 5

Check Next **Done**. Click any level to practice more. Completion is preserved.

✔ Expected: 151 blocks, 270 rows

(3,600 bytes/block) / (40 bytes/row) = 90 rows/block

So 13,575 rows would need:
13,575 rows / (90 rows/block) = 150 blocks with remainder 75, so 151 blocks.

Each bucket has 1 block, and 1 block has 90 rows. So 3 buckets can have up to 90 x 3 = 270 rows.

View solution *(instructors only)*

Urtaza, Alvaro A

4:29 PM 3/18/2024

Type here to search

72°F Partly sunny

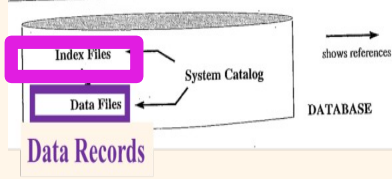
4:29 PM 3/18/2024

TA, Jordan (A – L).

TA, Alvaro (M – Z).

Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(3.18.2024 Attendance X missing LastName.docx)

Alternative File Organizations



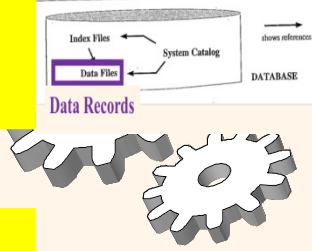
Three alternatives exist, ***each ideal for some situations, and not so good in others:***

- **Heap** (random order) **Files**: Suitable when typical **access** is a **File scan** *retrieving* ***all Data Records***.

- **Sorted Files**: Best if **Data Records** must be **retrieved in some** *order*, or only a `***range'*** **of Data**

Classes 18 & 19

Transcript File Stored as a Heap File



B = 3

R = 4

StudId	CrsCode	Sem	Grade	rid
666666	MGT123	F1994	4.0	<0,0>
123456	CS305	S1996	4.0	<0,1>
987654	CS305	F1995	2.0	<0,2>
				<0,3>
717171	CS315	S1997	4.0	<1,0>
666666	EE101	S1998	3.0	<1,1>
765432	MAT123	S1996	2.0	<1,2>
515151	EE101	F1995	3.0	<1,3>
234567	CS305	S1999	4.0	<2,0>
878787	MGT123	S1996		

- Heap (random order) files: Suitable when typical access is a file scan retrieving all records.

Heap table

Row-oriented storage performs better than column-oriented storage for most transactional databases. Consequently, relational databases commonly use row-oriented storage. A ***table structure*** is a scheme for organizing rows in blocks on storage media.

Databases commonly support four alternative table structures:

- Heap table
- Sorted table
- Hash table
- Table cluster

Each table in a database can have a different structure. Databases assign a default structure to all tables. Database administrators can override the default structure to optimize performance for specific queries.

In a ***heap table***, no order is imposed on rows. The database maintains a list of blocks assigned to the table, along with the address of the first available space for inserts. If all blocks are full, the database allocates a new block and inserts rows in the new block.

When a row is deleted, the space occupied by the row is marked as free. Typically, free space is tracked as a linked list, as in the animation below. Inserts are stored in the first space in the list, and the head of the list is set to the next space.

Heap tables optimize insert, update, and delete operations. Heap tables are particularly fast for bulk load of many rows, since rows are stored in load order. Heap tables are not optimal for queries that read rows in a specific order, such as a range of primary key values, since rows are scattered randomly across storage media.

PARTICIPATION ACTIVITY 13.2.1: Heap table.

TA time (Alvaro)
(CA 13.2.1 Steps 1, 2 – Table structure)

CHALLENGE ACTIVITY | 13.2.1: Table structures.

Refer to the heap table below. Free space A can accommodate two rows.

Heap Table

ID	Name	State
511	Sahar	HI
1507	Artyom	SC
1127	Wei	AR
1418	Lachlan	KS
576	Harry	NV
free space A		

free space B

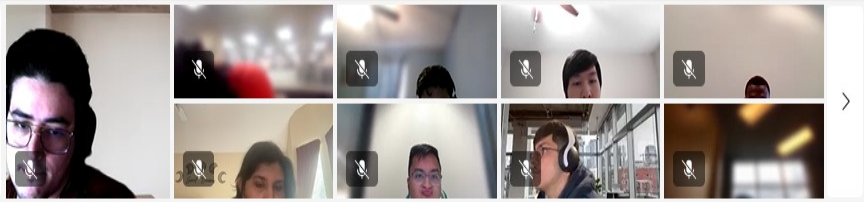
Where does the free space pointer point to?

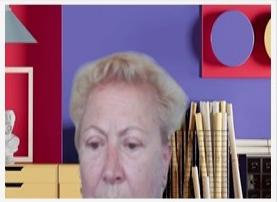
Select

free space A

free space B -> free space A

1 | 2





292	Enrique	AR
free space A		

Where does the free space pointer point to?

free space A

Row 586 is deleted and becomes free space B. What does the free space linked list look like?

free space B → free space A

1 2 3 4 5

Check Next

✔ Expected:
free space A
free space B → free space A

In a heap table, the database maintains a pointer to the first free space in the table. In the above table, free space A is the only free space, so the pointer points to free space A.

Row 586 is deleted, so the free space pointer moves to free space B. Free space B is linked to free space A, the next free space.

View solution (Instructors only)

Participants

Invite someone or dial a number

Share invite

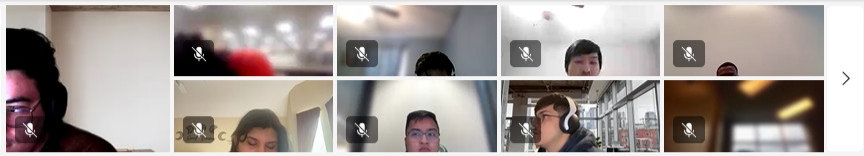
In this meeting (113) Mute all

Hilford, Victoria Organizer
Adhikari, Rohit
Akram, Ali
Akukwe, Benetta O
Alsayed, Sami H
Alvarez, Stephanie
Anayor-Achu, Ogochukwu E
Avci, Hatice Kubra
Aysola, Riya
Bahl, Anish
Banza, Sean Paolo B
Bui, Hieu
Burger, Jake

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

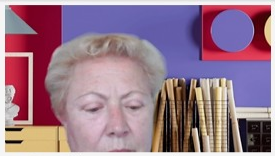
47:51

Take control | Pop out | Chat | People 113 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave



AU
Urtaza, Alvaro A

View all



zyBooks My library > COSC 3380: Database Systems home > 13.2: Table structures

zyBooks catal... 125% Reset ...aro Urtaza

654	Haruki	NM
1926	Lois	GA
free space A		

Row 1743 is deleted and becomes free space B.
Then, 4 new rows are inserted into the table.
Where are the 4 inserts stored?

First insert: free space B | Third insert: free space A

Second insert: free space A | Fourth insert: new block

1 2 3 4 5

Check Next

✔ Expected: free space B, free space A, free space A, new block

When row 1743 is deleted, the free space pointer moves to free space B. Free space B is linked to free space A, the next free space.

Because the free space pointer now points to free space B, the first insert is stored in free space B.
Then, the free space pointer is reset to point to free space A, so the second insert is stored in free space A.
Free space A still has room for another row, so the third insert is also stored in free space A.
Since the block is now full, the free space pointer is set to point to the beginning of the new block, where the fourth insert is stored.

View solution (Instructors only)

Feedback?

Table clusters

Urtaza, Alvaro A

Participants

Invite someone or dial a number

Share invite

In this meeting (113) Mute all

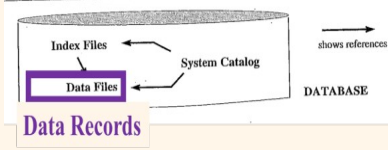
- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B
- HB Bui, Hieu
- Burger, Jake

Type here to search

72°F Partly sunny

4:36 PM
3/18/2024

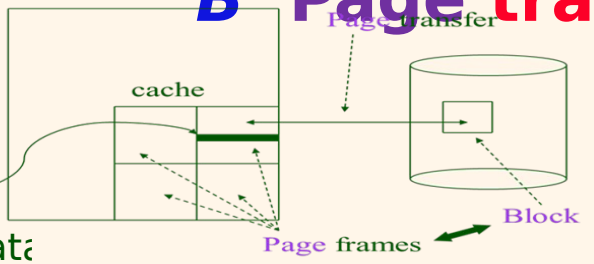
Heap File - Performance



- Assume **File** (**Table/Relation**) contains **B Pages**

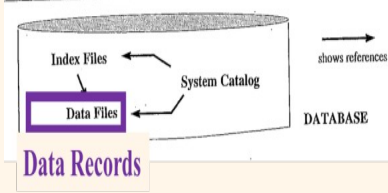
- Inserting *a **Data Record (row)***:
  - **Access path** is **scan**
  - **Avg**. ***B*/2 Page transfers** if **row** already exists
  - ***B*+1** Page **transfers** if **row** does not already exist

- Deleting *a **Data Record (row)***:
  - **Access path** is **scan**
  - **Avg**. ***B*/2+1 Page transfers** if **row**
  - ***B*** **Page transfers** if **row** does not



StudId	CrsCode	Sem	Grade	
666666	MGT123	F1994	4.0	
123456	CS305	S1996	4.0	page 0
987654	CS305	F1995	2.0	
717171	CS315	S1997	4.0	
666666	EE101	S1998	3.0	page 1
765432	MAT123	S1996	2.0	
515151	EE101	F1995	3.0	
234567	CS305	S1999	4.0	page 2
878787	MGT123	S1996	3.0	

Heap File - Performance



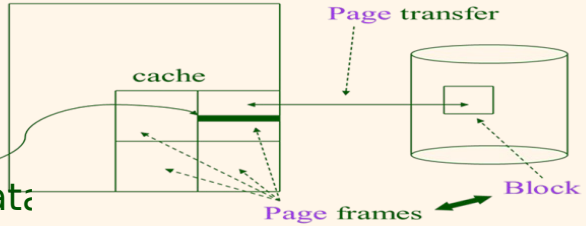
Query

- Access path is **scan** – **O**(B)
- Organization ***efficient*** if **SQL** query returns **all Rows/Data Records/Tuples** and **order of access is not important**

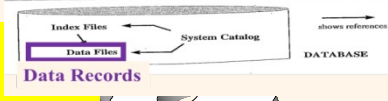
```
SELECT * FROM Transcript
```

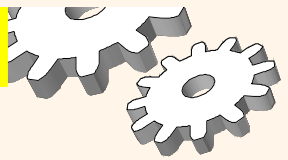
- Organization ***inefficient*** if a ***few*** **Rows** are requested
  - SELECT T.*Grade*
    FROM Transcript T
    WHERE T.*StudId*=12345 **AND** T.*CrsCode* ='CS305' **AND** T.*Sem* = 'S200

Avg. *B/2* Pages read to get **a single Row**



Sorted File





Rows/Data Records/Tuples are **Sorted** based on some **attribute(s)**

- **Access path** is Binary Search - **O**($\log_2 B$)
- Equality or Range **SQL** query based on that **attribute** has cost ***log$_2$B*** to retrieve Page containing **first Row/Data Record**
- **Successive Rows** are in **same** (or **successive**) **Page**(s) and Cache hits are likely
- By storing **all Pages** on the same track, **minimized**

Example – **Transcript** Sorted on ***StudId*** :

```
SELECT T.CrsCode,
       T.Grade
FROM   Transcript T

WHERE  T.StudId = 123456
```

```
SELECT T.CrsCode, T.Grade
FROM   Transcript T
WHERE  T.StudId BETWEEN
              111111 AND 199999
```

sid	name	login	age	gpa
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8

Figure 8.6 An Instance of the Students Relation, Sorted by *age*

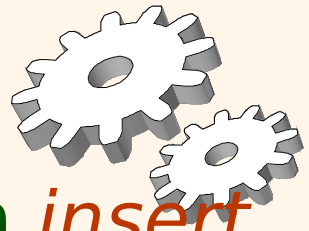
- Sorted Files: Best if records must be retrieved in some order, or only a 'range' of records is needed.

Transcript Stored as a Sorted File by StudId

B = 3
R = 4

StudId	CrsCode	Sem	Grade	rid
111111	MGT123	F1994	4.0	<0,0>
111111	CS305	S1996	4.0	<0,1>
123456	CS305	F1995	2.0	<0,2>
				<0,3>
123456	CS315	S1997	4.0	<1,0>
123456	EE101	S1998	3.0	<1,1>
232323	MAT123	S1996	2.0	<1,2>
234567	EE101	F1995	3.0	<1,3>
234567	CS305	S1999	4.0	<2,0>
				<2,1>
				<2,2>
313131	MGT123	S1996	3.0	<2,3>

Maintaining Sorted Order



- **Problem**: After the correct position for an *insert* has been determined, inserting **the Row requires** (on average)

  B/2 reads, and *B/2* writes (because **shifting** is necessary to make space)

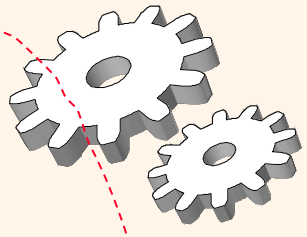
- **Partial Solution 1**: Leave empty space in each **Page**: *fillfactor*

- **Partial Solution 2**: Use *overflow* ***Pages*** (*chains*)
  - Disadvantages:
    - Successive **Pages** no longer stored contiguously
    - Overflow **chain not sorted**, hence **c**

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8

Figure 8.6 An Instance of the Students Relation, Sorted by *age*

Overflow



3			
111111	MGT123	F1994	4.0
111111	CS305	S1996	4.0
111111	ECO101	F2000	3.0
122222	REL211	F2000	2.0

Page 0

Pointer to overflow chain

-			
123456	CS315	S1997	4.0
123456	EE101	S1998	3.0
232323	MAT123	S1996	2.0
234567	EE101	F1995	3.0

Page 1

These Pages are Not overflown

-			
234567	CS305	S1999	4.0
313131	MGT123	S1996	3.0

Page 2

Pointer to next block in chain

7			
111654	CS305	F1995	2.0
111233	PSY 220	S2001	3.0

Page 3

Sorted table

In a ***sorted table***, the database designer identifies a ***sort column*** that determines physical row order. The sort column is usually the primary key but can be a non-key column or group of columns.

Rows are assigned to blocks according to the value of the sort column. Each block contains all rows with values in a given range. Within each block, rows are located in order of sort column values.

Sorted tables are optimal for queries that read data in order of the sort column, such as:

- JOIN on the sort column
- SELECT with range of sort column values in the WHERE clause
- SELECT with ORDER BY the sort column

Maintaining correct sort order of rows within each block can be slow. When a new row is inserted or when the sort column of an existing row is updated, free space may not be available in the correct location. To maintain the correct order efficiently, databases maintain pointers to the next row within each block, as in the animation below. With this technique, inserts and updates change two address values rather than move entire rows.

When an attempt is made to insert a row into a full block, the block splits in two. The database moves half the rows from the initial block to a new block, creating space for the insert.

In summary, sorted tables are optimized for read queries at the expense of insert and update operations. Since reads are more frequent than updates and inserts in many databases, sorted tables are often used, usually with the primary key as the sort column.

PARTICIPATION ACTIVITY 13.2.3: Sorted table.

Love my linked lists

Sorted table

PARTICIPATION ACTIVITY 13.2.3: Sorted table.

sort column

insert new row

14	Lufthansa	10:30	FRA	LHR
552	Aer Lingus	18:00	DUB	LHR
666	United Airlines	6:00	DUL	ROM
906	Air India	3:15	DXB	BOM
941	American Airlines	5:05	ORD	SFO
1107	Air China	3:20	HKG	PEK
8033	American Airlines	13:50	LAX	DXB
free space				

PARTICIPATION ACTIVITY 13.2.4: Sorted table.

1) Assume 100 rows fit in each block of a sorted table. If a block is full and an insert causes the block to split, roughly what percentage of the new and old blocks is empty?

- ○ 33%
- ◉ 50%
- ○ 90%

Correct

When an insert causes a full block to split, roughly half the rows are reassigned to the new block. Both new and old blocks are roughly half full.

?????

TA time (Alvaro)
(CA 13.2.1 Steps 3 – Table structure)

CHALLENGE ACTIVITY | 13.2.1: Table structures.

399850.26550.qx3zqy7

Sorted Table

Jump to level 1

Table A

ID	Name	State
10	Imani	VT
17	Wei	NE
20	Yuna	AR
25	Priya	HI
29	Deja	WA

free space

Refer to table A, which is a sorted table. Order is maintained with links and the first column is the sort column.

A row with key 19 is inserted at the end of the table. Which of the following pointers are added?

Select

Table B

ID	Name	State
25	Asha	WV
18	Sara	KS
17	Tatsuki	KY
16	Ximena	OK
24	Suki	MT

free space

Refer to table B. Which of the following (if any) are sort columns?

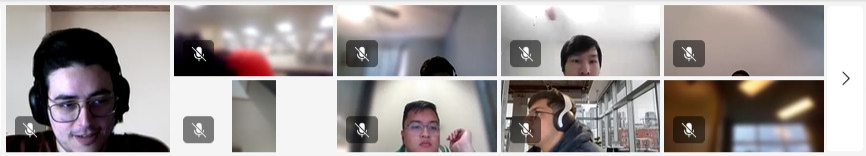
- ☐ ID
- ☐ Name
- ☐ State
- ☐ (ID, Name)
- ☐ (ID, State)
- ☐ (Name, State)

3

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

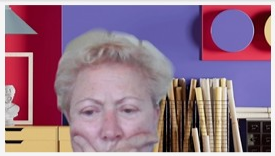
58:58

Take control | Pop out | Chat | People 113 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave



AU Urtaza, Alvaro A

View all



zyBooks My library > COSC 3380: Database Systems home > 13.2: Table structures

zyBooks catalog | Help/FAQ | Alvaro Urtaza

25	Aditya	MO
free space		

20 → 21, 21 → 22

Table B

ID	Name	State
22	Aoife	AZ
14	Maite	DE
12	Sumit	MS
20	Zehra	NC
28	Yurena	NM
free space		

Refer to table B. Assume each row's pointer points to the row that immediately follows. Which of the following (if any) are sort columns?

- ☐ ID
- ☐ Name
- ☑ State
- ☐ (ID, Name)
- ☐ (ID, State)
- ☐ (Name, State)

1 2 **3** 4 5

Check Next

✔ Expected:
20 → 21, 21 → 22
State

A pointer links a row to the row with the next higher key.
Since 20 < 21 < 22, pointers 20 → 21 and 21 → 22 are added to table A.

The sort column can be a non-key column. In table B, the State column is sorted, so the State column is a valid sort column, regardless of which column contains the primary key values.

View solution (Instructors only)

Feedback?

Urtaza, Alvaro A

Participants

Invite someone or dial a number

Share invite

In this meeting (113) Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B
- HB Bui, Hieu
- Burger, Jake

Type here to search

74°F Partly sunny

4:47 PM 3/18/2024

Hash table

In a ***hash table***, rows are assigned to buckets. A ***bucket*** is a block or group of blocks containing rows. Initially, each bucket has one block. As a table grows, some buckets eventually fill up with rows, and the database allocates additional blocks. New blocks are linked to the initial block, and the bucket becomes a chain of linked blocks.

The bucket containing each row is determined by a hash function and a hash key. The ***hash key*** is a column or group of columns, usually the primary key. The ***hash function*** computes the bucket containing the row from the hash key.

Hash functions are designed to scramble row locations and evenly distribute rows across blocks. The ***modulo function*** is a simple hash function with four steps:

1. Convert the hash key by interpreting the key's bits as an integer value.
2. Divide the integer by the number of buckets.
3. Interpret the division remainder as the bucket number.
4. Convert the bucket number to the physical address of the block containing the row.

As tables grow, a fixed hash function allocates more rows to each bucket, creating deep buckets consisting of long chains of linked blocks. Deep buckets are inefficient since a query may read several blocks to access a single row. To avoid deep buckets, databases may use dynamic hash functions. A ***dynamic hash function*** automatically allocates more blocks to the table, creates additional buckets, and distributes rows across all buckets. With more buckets, fewer rows are assigned to each bucket and, on average, buckets contain fewer linked blocks.

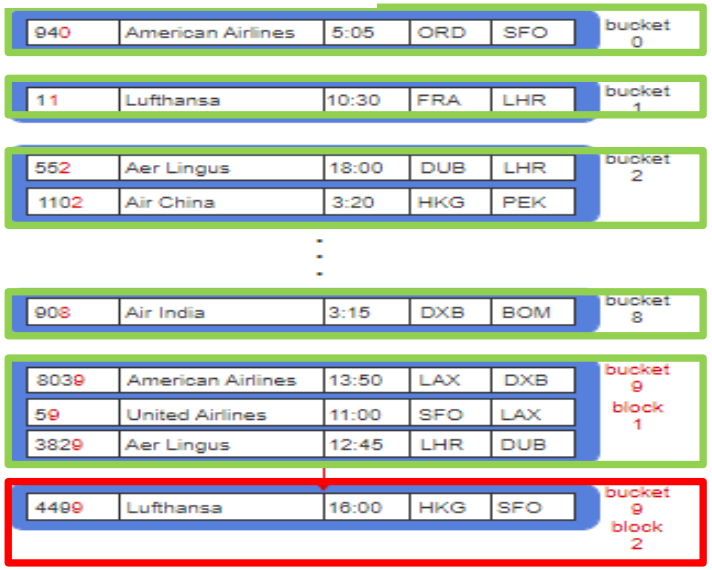
Hash tables are optimal for inserts and deletes of individual rows, since row location is quickly determined from the hash key. For the same reason, hash tables are optimal for selecting a single row when the hash key value is specified in the WHERE clause. Hash tables are slow on queries that select many rows with a range of values, since rows are randomly distributed across many blocks.

PARTICIPATION ACTIVITY 13.2.5: Assigning rows to buckets with a modulo function.

Flight

Flight Number	AirlineName	Depart Time	Depart Airport	Arrive Airport
11	Lufthansa	10:30	FRA	LHR
552	Aer Lingus	18:00	DUB	LHR
908	Air India	3:15	DXB	BOM
940	American Airlines	5:05	ORD	SFO
1102	Air China	3:20	HKG	PEK
8039	American Airlines	13:50	LAX	DXB
59	United Airlines	11:00	SFO	LAX
3829	Aer Lingus	12:45	LHR	DUB
4499	Lufthansa	16:00	HKG	SFO

hash key



Hash table

PARTICIPATION ACTIVITY | 13.2.5: Assigning rows to buckets with a modulo function.

Flight

Flight Number	AirlineName	Depart Time	Depart Airport	Arrive Airport
11	Lufthansa	10:30	FRA	LHR
552	Aer Lingus	18:00	DUB	LHR
908	Air India	3:15	DXB	BOM
940	American Airlines	5:05	ORD	SFO
1102	Air China	3:20	HKG	PEK
8039	American Airlines	13:50	LAX	DXB
59	United Airlines	11:00	SFO	LAX
3829	Aer Lingus	12:45	LHR	DUB
4499	Lufthansa	16:00	HKG	SFO

hash key

PARTICIPATION ACTIVITY | 13.2.6: Hash table.

Assume blocks are 8 kilobytes, rows are 200 bytes, and the hash function is modulo 13.

1) If each bucket initially has one block, how many rows fit into a single bucket?

- 13
- 40
- 520

Correct

Each bucket initially has one block, and each block is about 8,000 bytes. 8,000 bytes / 200 bytes/row = 40 rows per bucket.

200 B

8KB = 8,000 B

R = 8,000 / 200 = 40 data records / page

?????

TA time (Alvaro)
(CA 13.2.1 Step 4– Table structure)

CHALLENGE ACTIVITY | 13.2.1: Table structures.

Hash Table

Consider a hash table with 5 buckets and a modulo function. Which bucket does each hash key map to?

Hash key	Bucket
250	Ex: 5
1428	
322	
557	
1439	

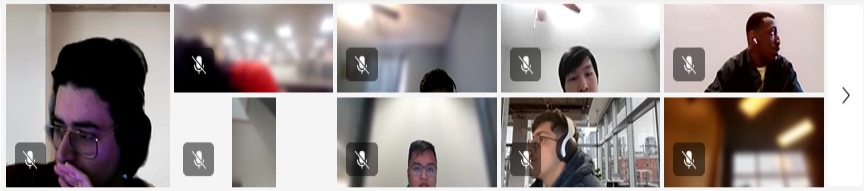
250 mod 5 = 0

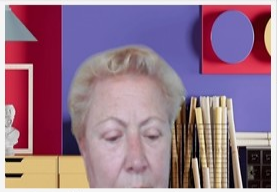
1428 mod 5 = 3

322 mod 5 = 2

557 mod 5 = 2

1439 mod 5 = 4





1070	0
1257	2

1 2 3 4 5

Check Next

✔ Expected: 1, 3, 3, 0, 2

5 buckets exist, so the hash function is modulo 5.

1211 modulo 5 is 1, so bucket 1.
253 modulo 5 is 3, so bucket 3.
948 modulo 5 is 3, so bucket 3.
1070 modulo 5 is 0, so bucket 0.
1257 modulo 5 is 2, so bucket 2.

View solution (*Instructors only*)

Feedback?

Table clusters

Table clusters, also called ***multi-tables***, interleave rows of two or more tables in the same storage area. Table clusters have a ***cluster key***, a column that is available in all interleaved tables. The cluster key determines the order in which rows are interleaved. Rows with the same cluster key value are stored together. Usually the cluster key is the primary key of one table and the corresponding foreign key of another, as in the animation below.

Table clusters are optimal when joining interleaved tables on the cluster key, since physical row location is the same as output order. Table clusters perform poorly for many other queries:

- *Join on columns other than cluster key.* In a join on a column that is not the cluster key, physical row location is not the same as output order, so the join is slow.

Table clusters

Table clusters, also called ***multi-tables***, interleave rows of two or more tables in the same storage area. Table clusters have a ***cluster key***, a column that is available in all interleaved tables. The cluster key determines the order in which rows are interleaved. Rows with the same cluster key value are stored together. Usually the cluster key is the primary key of one table and the corresponding foreign key of another, as in the animation below.

Table clusters are optimal when joining interleaved tables on the cluster key, since physical row location is the same as output order. Table clusters perform poorly for many other queries:

- **Join on columns other than cluster key**. In a join on a column that is not the cluster key, physical row location is not the same as output order, so the join is slow.
- **Read multiple rows of a single table**. Table clusters spread each table across more blocks than other structures. Queries that read multiple rows may access more blocks.
- **Update clustering key**. Rows may move to different blocks when the clustering key changes.

Table clusters are not optimal for many queries and therefore are not commonly used.

PARTICIPATION ACTIVITY 13.2.7: Table cluster.

Airline

AirlineName	Code	Country
Lufthansa	LH	Germany
Aer Lingus	EI	Ireland
Air India	AI	India
American Airlines	AA	United States
Air China	CA	China

Flight

Flight Number	AirlineName	Depart Time	Depart Airport	Arrive Airport
11	Lufthansa	10:30	FRA	LHR
552	Aer Lingus	18:00	DUB	LHR
908	Lufthansa	3:15	DXB	BOM
940	American Airlines	5:05	ORD	SFO
1102	Air China	3:20	HKG	PEK
8039	American Airlines	13:50	LAX	DXB

Lufthansa	LH	Germany		
11	Lufthansa	10:30	FRA	LHR
908	Lufthansa	3:15	DXB	BOM
Aer Lingus	EI	Ireland		
552	Aer Lingus	18:00	DUB	LHR
Air India	AI	India		
American Airlines	AA	United States		
940	American Airlines	5:05	ORD	SFO
8039	American Airlines	13:50	LAX	DXB
Air China	CA	China		
1102	Air China	3:20	HKG	PEK

13.3 Tablespaces and partitions

Tablespaces

Tablespaces and partitions are supported by most databases but are not specified in the SQL standard. Most implementations are similar, but SQL syntax and capabilities vary. This section describes the MySQL implementation.

A ***tablespace*** is a database object that maps one or more tables to a single file. The CREATE TABLESPACE statement names a tablespace and assigns the tablespace to a file. The CREATE TABLE statement assigns a table to a tablespace. Indexes are stored in the same tablespace as the indexed table.

Figure 13.3.1: SQL for tablespaces.

```
CREATE TABLESPACE TablespaceName
[ ADD DATAFILE 'FileName' ];

CREATE TABLE TableName
( ColumnName ColumnDefintion, ... )
[ TABLESPACE TablespaceName ];
```

Feedback?

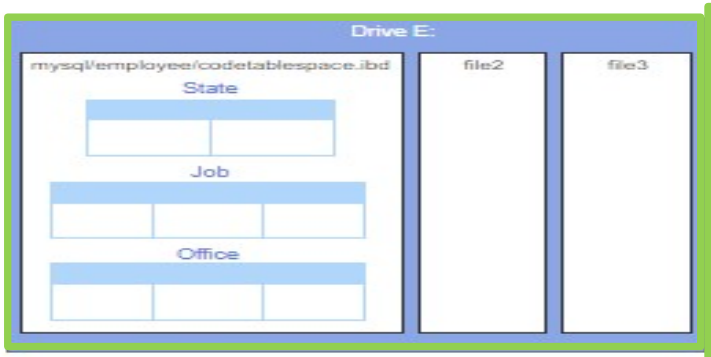
By default, most databases automatically create one tablespace for each table, so each table is stored in a separate file. Database administrators can manually create tablespaces and assign one or multiple tables to each tablespace. Database administrators can improve query performance by assigning frequently accessed tables to tablespaces stored on fast storage media.

In most cases, databases perform better with a single table per tablespace:

- Individual tables can be backed up independently of other tables.
- When a table is dropped, the associated file is deleted and storage is released. When multiple tables are stored in one tablespace, all tables must be dropped to release storage.
- Concurrent updates of multiple tables are usually faster when each table is stored in a separate file.
- Blocks of a new file are usually allocated contiguously on a few tracks of a disk drive. As files are updated, blocks become scattered, or ***fragmented***, across many tracks. Queries that scan tables on heavily fragmented files are slow because the disk drive must read many tracks. When tables are updated, storing one table per file minimizes fragmentation and optimizes table scans.

In some cases, assigning multiple tables to one tablespace can improve performance. Each tablespace must be managed by the database and incurs a small amount of overhead. Storing many small tables in one tablespace reduces overhead and, if the tables are commonly accessed in the same query, may improve query performance. If the tables are read-only, assigning the tables to one tablespace does not increase fragmentation.

PARTICIPATION ACTIVITY | 13.3.1: Assigning three tables to the same tablespace.




```
CREATE TABLESPACE CodeTablespace
ADD DATAFILE
    'E:/mysql/employee/codetablespace.ibd'

CREATE TABLE State (
    StateCode CHAR(2),
    StateName VARCHAR(20)
)
TABLESPACE CodeTablespace;

CREATE TABLE Job (
    JobCode SMALLINT,
    JobTitle VARCHAR(30),
    JobLevel SMALLINT
)
TABLESPACE CodeTablespace;

CREATE TABLE Office (
    OfficeCode CHAR(8),
    OfficeName VARCHAR(20),
    OfficeType VARCHAR(4)
)
TABLESPACE CodeTablespace;
```

Partitions

A ***partition*** is a subset of table data. One table has many partitions that do not overlap and, together, contain all table data. A ***horizontal partition*** is a subset of table rows. A ***vertical partition*** is a subset of table columns. MySQL and most relational databases partition tables horizontally, not vertically.

Each partition is stored in a separate tablespace, specified either explicitly by the database administrator or automatically by the database. When a table is partitioned, table indexes are also partitioned. Each partition contains index entries only for rows in the partition.

Partitions can be defined in several ways. Often, rows are assigned to partitions based on values of a specific column. Each partition may be associated with a continuous range of values or an explicit list of values. Ex:

- The Employee table has a HireDate column. One partition is created for each year. Employees are assigned to partitions based on the year hired.
- The Employee table has an OfficeID column with 10 possible values. One partition is created for each office. Employees working in the same office are stored in the same partition.

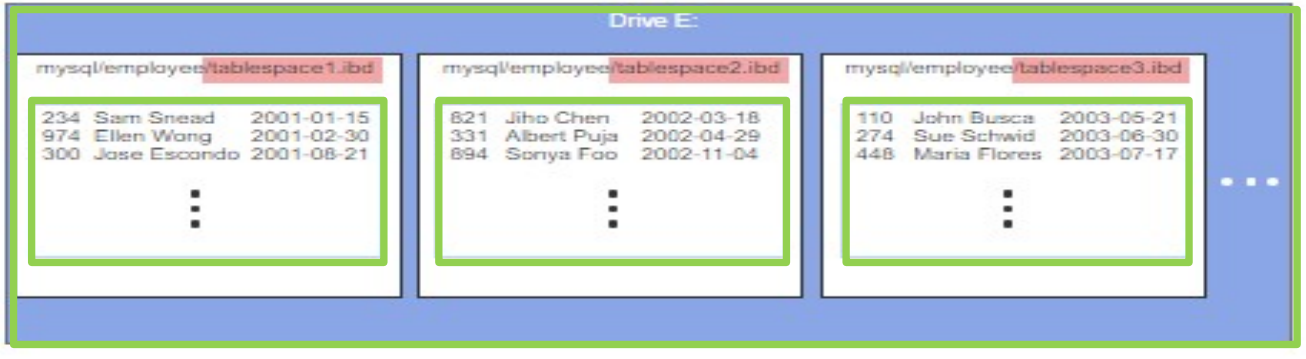
Partitions improve query performance by reducing the amount of data accessed by INSERT, UPDATE, DELETE, and SELECT statements. Ex: The Sales table contains sales transactions for the past ten years, with one partition for each year. Most queries access current year sales only. The current-year partition is a tenth of the table size, so current-year queries access less data and execute faster.

Terminology

*The term **partition** means either an individual subset of a table or, collectively, all subsets of a table. Usually, the meaning is clear from context. In this material, partition means an individual subset.*

*A shard is similar to a partition. Like a partition, a **shard** is a subset of table data, usually a subset of rows rather than columns. Unlike partitions, which are stored on different storage devices of a single computer, shards are stored on different computers of a distributed database.*

PARTICIPATION ACTIVITY 13.3.3: Employee table partitioned by hire date.



TA time (Alvaro)

(CA 13.3.1 Step 1 – Tablespaces and partitions)

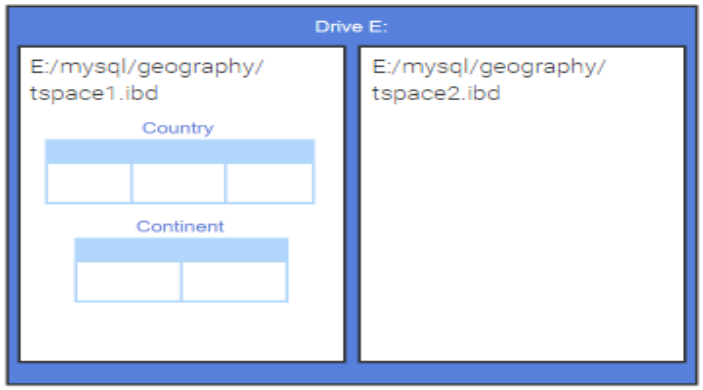
CHALLENGE ACTIVITY | 13.3.1: Tablespaces and partitions.

399850.26550.qx3zqy7

Tablespace

Jump to level 1

Continent and Country are small, read-only tables, and therefore assigned to the same tablespace.




```
CREATE TABLESPACE Tspace1
ADD DATAFILE
    'E:/mysql/geography/tspace1.ibd'

CREATE __(A)__ Tspace2
__(B)__
    'E:/mysql/geography/tspace2.ibd'

CREATE TABLE Country (
   Code SMALLINT,
   Name VARCHAR(15),
   Capital VARCHAR(15)
)
TABLESPACE Tspace1;

CREATE TABLE Continent (
   Code CHAR(2),
   Name VARCHAR(15),
)
TABLESPACE __(C)__;
```

Complete the values of A, B and C:

(A) /* Type your code here */

(B)

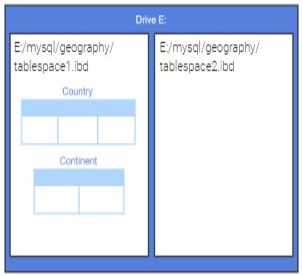
(C)

1

544874.26550.qx3zqy7

Jump to level 1

Continent and Country are small, read-only tables, and therefore assigned to the same tablespace.




```
CREATE TABLESPACE Tablespace1
__(A)__
   'E:/mysql/geography/tablespace1.ibd'

CREATE TABLESPACE Tablespace2
ADD DATAFILE
   'E:/mysql/geography/tablespace2.ibd'

CREATE TABLE Country (
   Code SMALLINT,
   Name VARCHAR(15),
   Capital VARCHAR(15)
)
TABLESPACE Tablespace1;

CREATE TABLE Continent (
   Code CHAR(2),
   Name VARCHAR(15),
)
__(B)__ __(C)__;
```

Complete the values of A, B and C:

(A) ADD DATAFILE

(B) TABLESPACE

(C) Tablespace1

1 2 3 4

Check Next

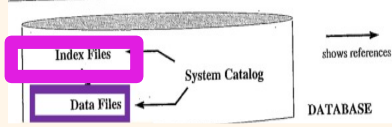
✔ Expected:
(A) ADD DATAFILE
(B) TABLESPACE
(C) Tablespace1

(A) `ADD DATAFILE 'E:/mysql/geography/tablespace1.ibd'` assigns Tablespace1 to file E:/mysql/geography/tablespace1.ibd.

(B) The second `CREATE TABLE` statement assigns the Continent table to Tablespace1.

(C) As shown in the image, the Continent table is assigned to Tablespace1. Thus, `TABLESPACE Tablespace1` is used in the CREATE TABLE statement.

Indexes – Data Structures



key,

r_{id}

- **Ordered** set of values that contains **Index** search key and *pointers*
- **More efficient** to use **Index** to access **Table** than to **scan all Rows** in **Table** sequentially

key, r_{id}

STATE_NDX INDEX

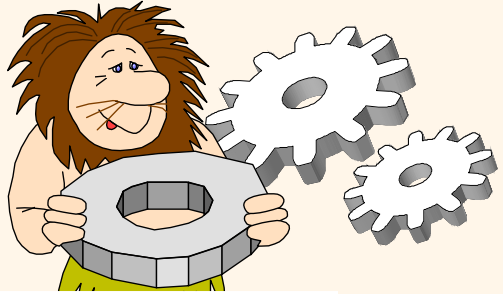
Key	Row
AZ	2
....	....
....	....
FL	1
FL	7
FL	8
FL	13245
FL	14786
....	
....	

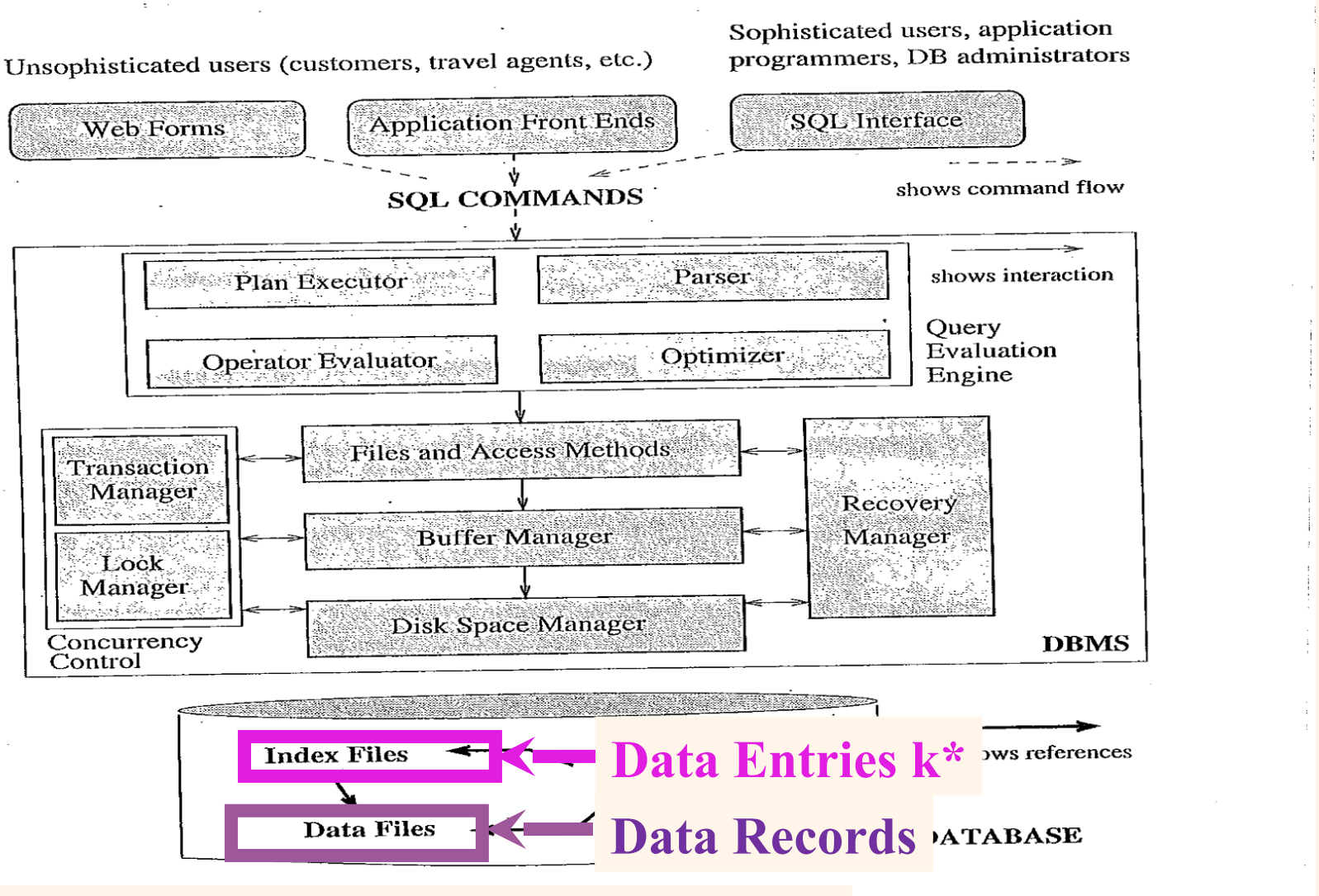
CUSTOMER TABLE
(14,786 rows)

Row #	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_STATE	CUS_BALANCE
1	10010	Ramas	Alfred	A	615	844-2573	FL	$0.00
2	10011	Dunne	Leona	K	713	894-1238	AZ	$0.00
3	10012	Smith	Kathy	W	615	894-2285	TX	$345.86
4	10013	Olowski	Paul	F				
5	10014	Orlando	Myron					
6	10015	O'Brian	Amy	B				
7	10016	Brown	James	C				
8	10017	Williams	George					
9	10018	Farriss	Anne	C				
10	10019	Smith	Olette	K				
....	.....	.....	.....	.				
....	.....	.....	.....	.				
13245	23120	Veron	George	D				
....	.....	.....	.....	.				
....	.....	.....	.....	.				
14786	24560	Suarez	Victor					

- Indexes: Data structures to organize records via trees or hashing.
  - Like Sorted Files, they speed up searches for a subset of records, based on values in certain ("search key") fields

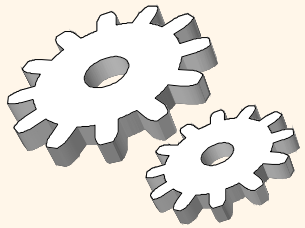
A DBMS




Unsophisticated users (customers, travel agents, etc.)
Sophisticated users, application programmers, DB administrators
Web Forms
Application Front Ends
SQL Interface
SQL COMMANDS
shows command flow
Plan Executor
Parser
Operator Evaluator
Optimizer
shows interaction
Query Evaluation Engine
Transaction Manager
Lock Manager
Concurrency Control
Files and Access Methods
Buffer Manager
Disk Space Manager
Recovery Manager
DBMS
Index Files
Data Files
Data Entries k*
Data Records
ows references
ATABASE

Indexes

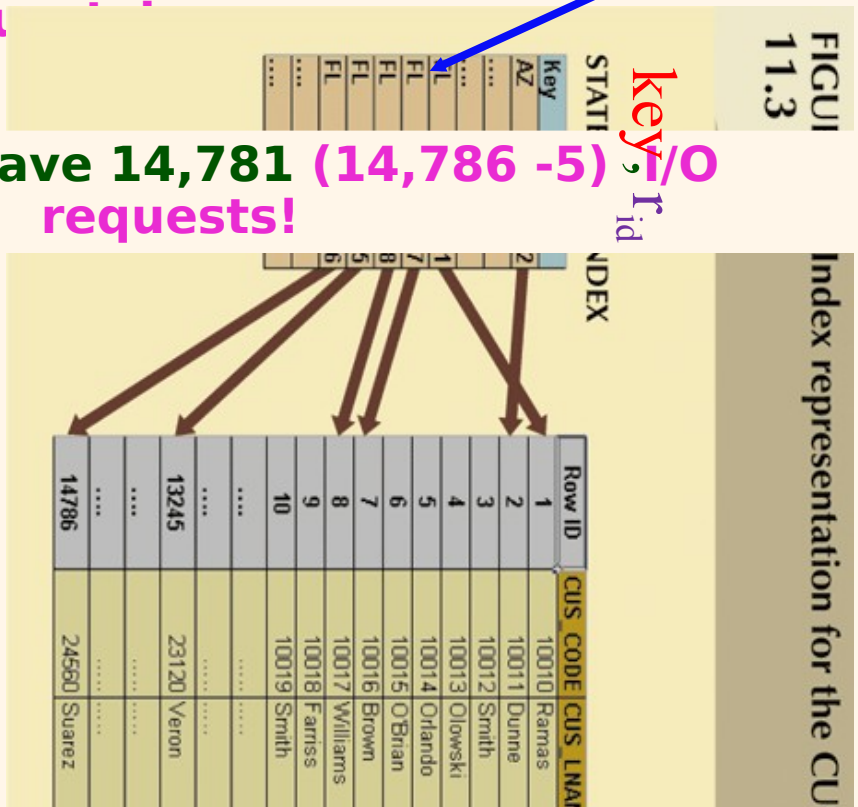
```
SELECT CUS_LNAME, CUS_STATE
FROM CUSTOMER
WHERE CUS_STATE = 'FL';
```



Without STATE_NDX INDEX, the DBMS will perform a **full Table scan**, 14,786 Da
Records

With STATE_NDX INDEX, the DBMS will use the Index to locate the first customer with a state equal to 'FL' then proceed to **read CUSTOMER row**, using the r_{id} in the STATE_NDX as a guide. Then, use the other records in STATE_NDX that match 'FL' (assume only 5!), thus **DBMS will save 14,781 (14,786 -5) I/O requ**

save 14,781 (14,786 -5) I/O requests!



14.1 Physical design

MySQL storage engines

Logical design specifies tables, columns, and keys. The logical design process is described elsewhere in this material. ***Physical design*** specifies indexes, table structures, and partitions. Physical design affects query performance but never affects query results.

A ***storage engine*** or ***storage manager*** translates instructions generated by a query processor into low-level commands that access data on storage media. Storage engines support different index and table structures, so physical design is dependent on a specific storage engine.

MySQL can be configured with several different storage engines, including:

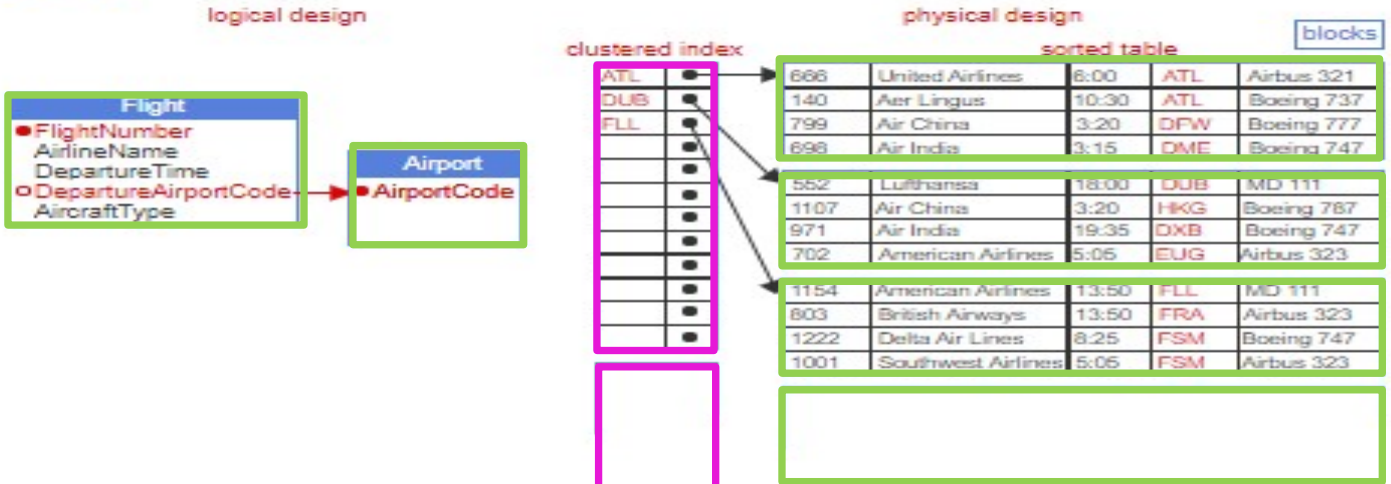
- InnoDB is the default storage engine installed with the MySQL download. InnoDB has full support for transaction management, foreign keys, referential integrity, and locking.
- MyISAM has limited transaction management and locking capabilities. MyISAM is commonly used for analytic applications with limited data updates.
- MEMORY stores all data in main memory. MEMORY is used for fast access with databases small enough to fit in main memory.

Different databases and storage engines support different table structures and index types. Ex:

- Table structure. Oracle Database supports heap, sorted, hash, and cluster tables. MySQL with InnoDB supports only heap and sorted tables.
- Index type. MySQL with InnoDB or MyISAM supports only B+tree indexes. MySQL with MEMORY supports both B+tree and hash indexes.

This section describes the physical design process and statements for MySQL with InnoDB. The process and statements can be adapted to other databases and storage engines, but details depend on supported index and table structures.

PARTICIPATION ACTIVITY | 14.1.1: Logical and physical design.



CREATE INDEX, DROP INDEX, and SHOW INDEX statements

Classes 18 & 19

The ***CREATE INDEX*** statement creates an index by specifying the index name and table columns that compose the index. Most indexes specify just one column, but a composite index specifies multiple columns.

The ***DROP INDEX*** statement deletes a table's index.

The ***SHOW INDEX*** statement displays a table's index. SHOW INDEX generates a result table with one row for each column of each index. A multi-column index has multiple rows in the result table.

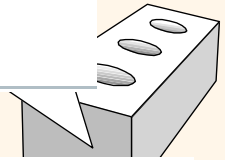
The SQL standard includes logical design statements such as CREATE TABLE but not physical design statements such as CREATE INDEX. Nevertheless, CREATE INDEX and many other physical design statements are similar in most relational databases.

Table 14.1.1: INDEX statements.

Statement	Description	Syntax
CREATE INDEX	Create an index	`CREATE INDEX IndexName ON TableName (Column1, Column2, ..., ColumnN);`
DROP INDEX	Delete an index	`DROP INDEX IndexName ON TableName;`
SHOW INDEX	Show an index	`SHOW INDEX FROM TableName;`

Feedback?

Storage I Practice Questions



The time required to read or write the first byte of data is known as ____.

a. random-access

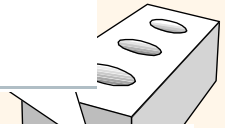
b. transfer rate

c. volatility

d. access time

?????

13. SET 3 - 1:STORAGE I



A magnetic disk groups data into ____.

a. kilobytes

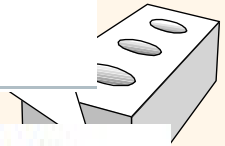
b. pages

c. blocks

d. sectors

?????

13. SET 3 - 1:STORAGE I



A relational database uses row-oriented storage to store an entire row within one ____.

a. page

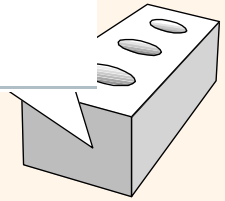
b. block

c. sector

d. Table

?????

13. SET 3 - 1:STORAGE I



The heap table structure is optimized for ____.

a. deleting all rows with primary key between two fixed values

b. inserting new rows

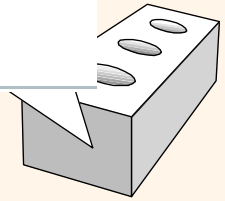
c. reading all rows with primary key between two fixed values

d. updating all occurrences of a specific value of a column, in all rows

?????

hrke

13. SET 3 - 1:STORAGE I



The sort column in a sorted table determines the ____ row order.

a. linked

b. key

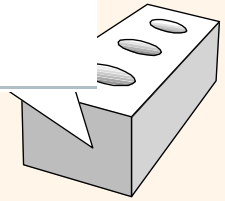
c. physical

d. dynamic

?????

hrke

13. SET 3 - 1:STORAGE I



Which table type might use the modulo function to scramble row locations?

a. Hash

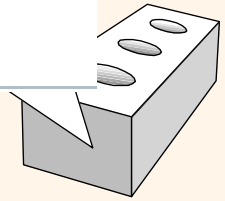
b. Heap

c. Sorted

d. Cluster

?????

13. SET 3 - 1:STORAGE I



When working with multi-tables, a column that is available in all interleaved tables is known as a ____?

a. hash function

b. bucket

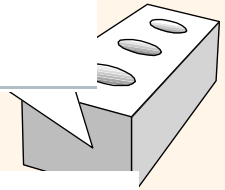
c. cluster key

d. sort column

?????

hrke

13. SET 3 - 1:STORAGE I



Which table type interleaves rows of two or more tables in the same storage area?

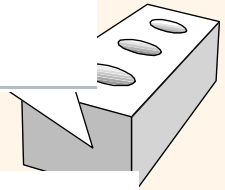
a. Heap tables

b. Table clusters

c. Hash tables

d. Sorted tables

?????



What object maps one or more tables to a single file?

a. Vertical partition

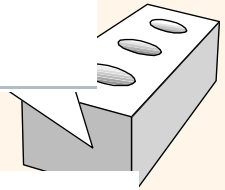
b. Horizontal partition

c. Tablespace

d. Shard

?????

hrke



A subset of table rows is called a ____.

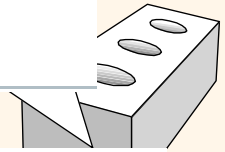
a. cluster

b. fragment

c. vertical partition

d. horizontal partition

?????



When a table is assigned to a tablespace, where is the index stored?

a. In an adjacent tablespace.

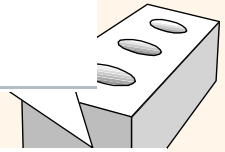
b. In the same tablespace as the table.

c. Within the table.

d. In a separate tablespace file.

?????

13. SET 3 - 1:STORAGE I



A ____ partition requires a partition expression with positive integer values.

a. horizontal

b. list

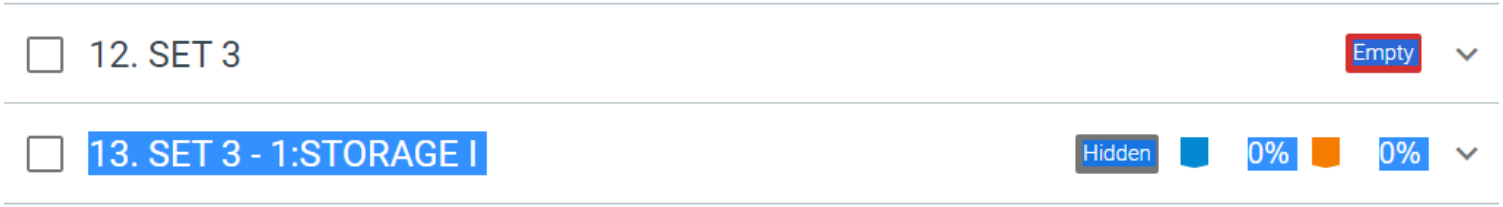
c. vertical

d. hash

?????

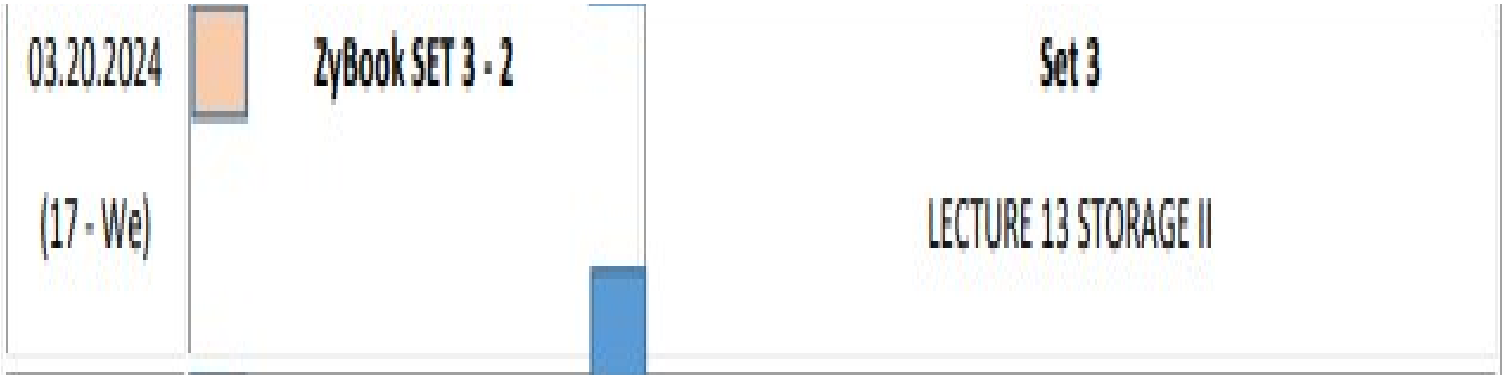
At 5:00 PM .

03.18.2024 (16 - Mo)	ZyBook SET 3 - 1	Set 3 LECTURE 12 STORAGE I



VH work on
SET 3 – 1: STORAGE I

Next



VH, unHIDE

From 5:05 to 5:15 PM – 5 minutes.

03.18.2024 (16 - Mo)	ZyBook SET 3 - 1	Set 3 LECTURE 12 STORAGE I

CLASS PARTICIPATION 20 points — 20% of Total

STORAGE I

Class 16 END PARTICIPATION
Not available until Mar 18 at 5:05pm | Due Mar 18 at 5:15pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

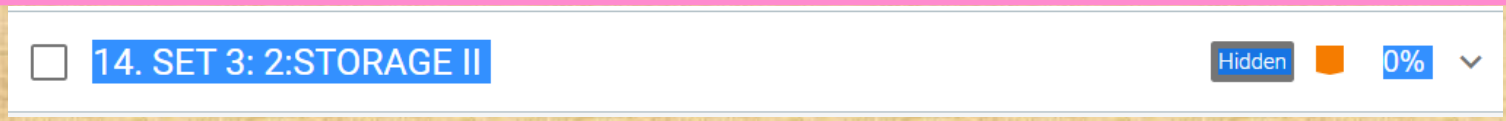
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 16

VH, unhide ZyBook Section 14.



VH, Download Attendance Report
Rename it:
3.18.2024 Attendance Report FINAL TEAMS

VH, upload Class 16 to CANVAS.