THE UNIVERSITY OF MELBOURNE

Database Architecture and Administration

Database Systems & Information Modelling
INFO90002

Week 8 – DBA

Dr Tanya Linden
David Eccles

This lecture discusses

The ‘Database Administrator’ role

Architecture – Understanding the DBMS

- concepts

Performance improvement

- concepts
- common approaches, e.g. indexes

The DBA role

Primarily concerned with “maintenance” / “ops” phase

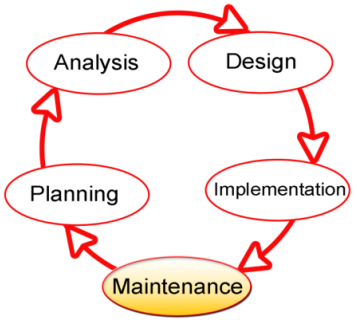
but should be consulted during all phases of development

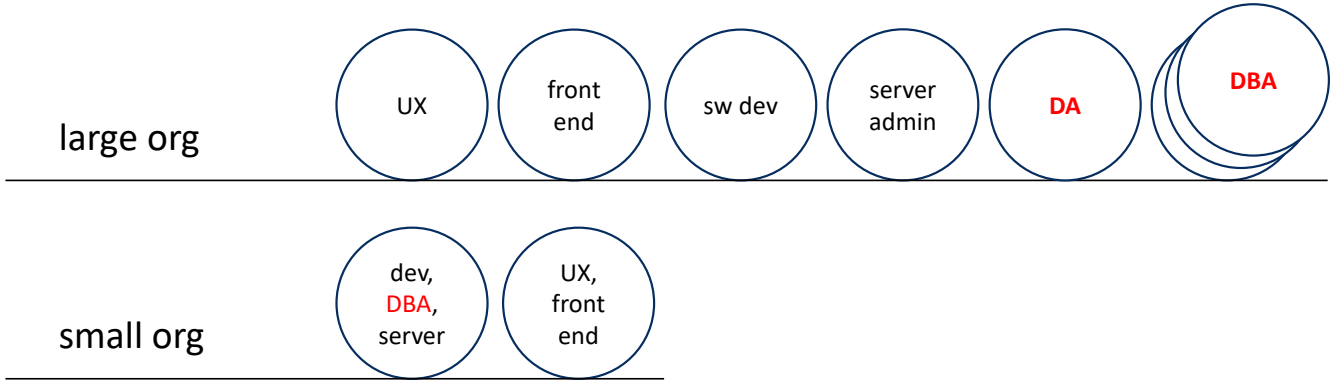
“Database Administrator” or “DBA” often framed as a “job” or a “person”

Large companies – many DBAs

Small company developer is the DBA

DBA role can be made redundant by Cloud-based DBMS or “data as a service” DAAS (often IAAS or PAAS)





Data and Database Administration

Data Administrator (CDO / CSO) (management role)

- Data policies, procedures and standards
- Compliance with legislation (EU GDPR, AUS Privacy Act)
- Compliance with company policy (e.g. Uni of Melbourne privacy policy)
- Planning
- Data conflict resolution
- Managing information repository
- Internal marketing and education

Database Administrator (technical role)

- analyze and design DB
- select DBMS / tools / vendor
- install and upgrade DBMS
- tune DBMS performance
- manage security, privacy, integrity
- backup and recovery

Data Administration

- Data conflict resolution
  - Conflict resolution is often possible in customer management systems. For example, salespeople can maintain customer address information at different databases in a replication environment. Should a conflict arise, the system can resolve the conflicting updates by applying the most recent update to a record
- Managing information repository
  - A corporate repository is a place where a large amount of a **corporation's information** is stored, typically for the long term. Corporate repositories are **not made public**. Only users with specific roles and permissions granted may access documents
- Internal marketing and education
  - Internal marketing is the **promotion of a company's objectives, products and services** to employees within the organization. The purpose is to increase employee engagement with the company's goals and foster brand advocacy.

DA Versus DBA

DATA ADMINISTRATOR (DA)	DATABASE ADMINISTRATOR (DBA)
Strategic planning	Control and supervision
Sets long-term goals	Executes plans to reach goals
Sets policies and standards	Enforces policies and procedures Enforces programming standards
Broad scope	Narrow scope
Long term	Short term (focus on daily operations)
Managerial orientation	Technical orientation
DBMS-independent	DBMS-specific

Architecture of a Database Management System (DBMS)

Database Systems Architecture

A Database Management System (DBMS) exists as one entity in two places

- In Memory
- Physically on disk

programs sitting on the hard disk, but run in the memory. (RAM, CPU)

Both places manage

- Data (the reason we have the DBMS)
- Performance (how it performs as it is used and grows)
- Concurrency (manages high volumes of users)
- Recoverability (assist in recovery and availability)

how quickly your database, work the program process data.

And how to find data in disc

consistance is important

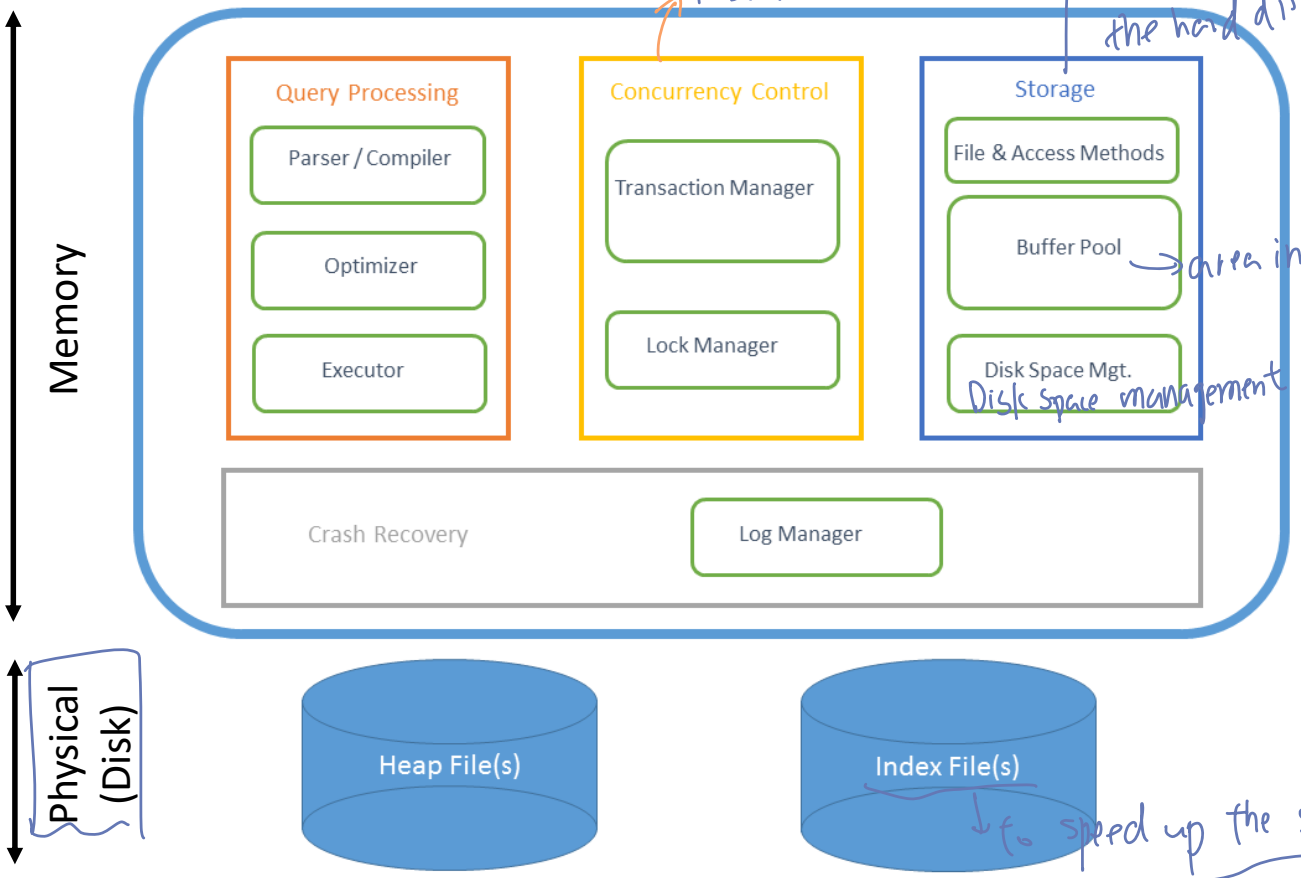
database might crush

One place is persistent, the other - transient

- Disk representation is always present (persistent)
- Memory – transient – only exists when DBMS is running (transient)

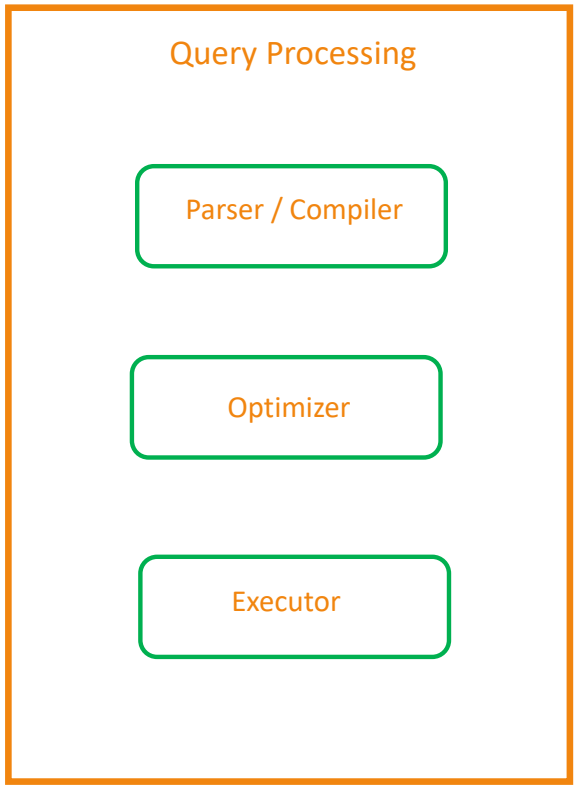
need to know

DBMS Overview



A heap file is **an unordered set of records, stored on a set of pages**.

Query Processing



get our data.

not syntax error
but can not run
→ don't have permisson

syntax analysising
↑
can be execute

Parsing

- Syntax is correct and can "compile"
- DBMS User Permissions
- Resources (Data, Code, ability to Record/Change /Retrieve results) Permission to the access

Optimising

- Execution Plan and Execution Cost
- Evaluate indexes, table scans, hashing
- Eliminate worst, consider best options
- Lowest cost theoretically "best"
  - which algorithm able to use

how to run program faster

This will check if the table is too big. does it need index or not.

Execution

- Meet the ACID test (atomicity, consistency, isolation, and durability),
- Atomic: All rows succeed, or all fail
- Ensure resources are available
  - Data, Log changes, Memory, Cursor to do the work for the USER

Query Processing

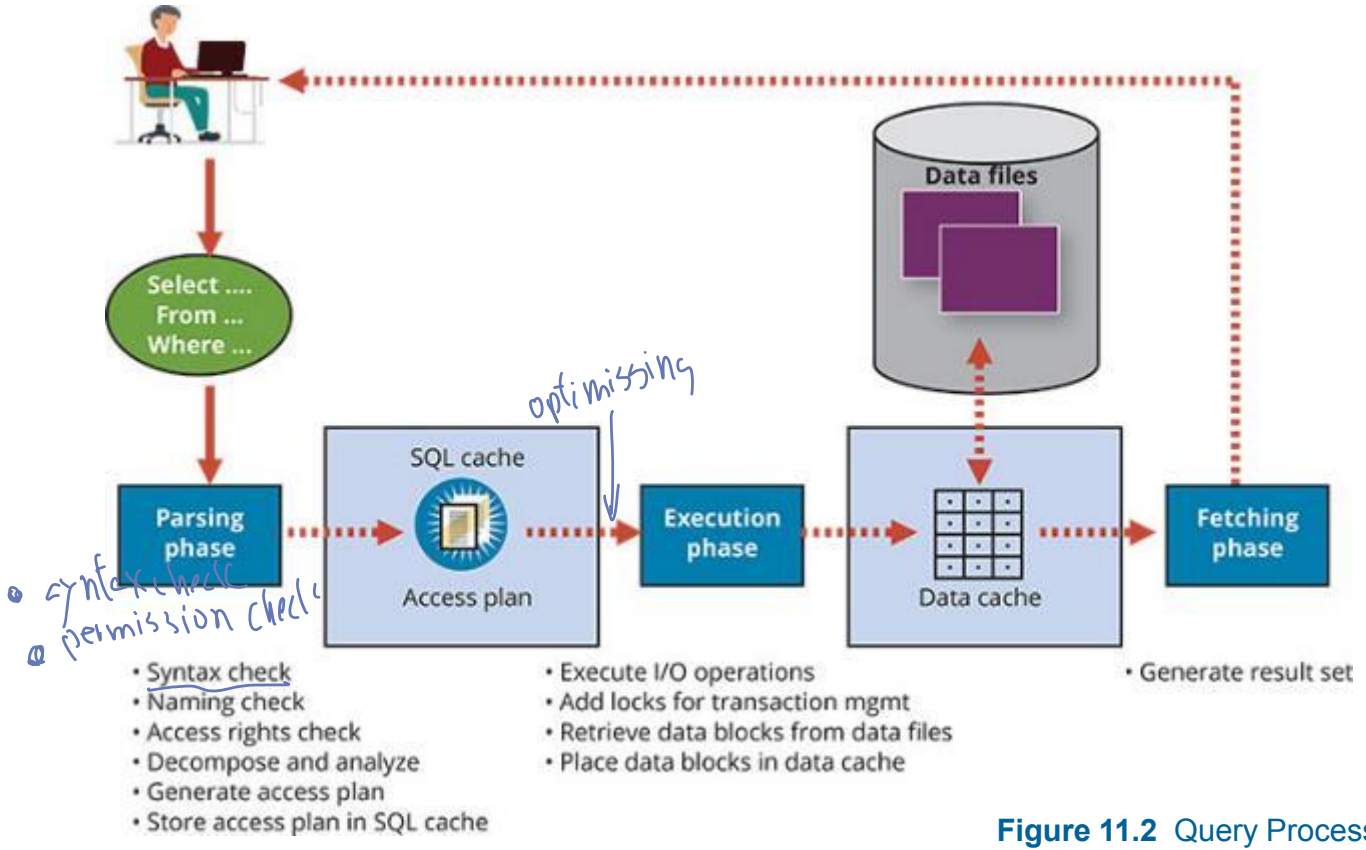


Figure 11.2 Query Processing (Coronel & Morris, 2023)

Query Processing - SQL Parsing Phase

The optimization process includes breaking down the query into smaller units

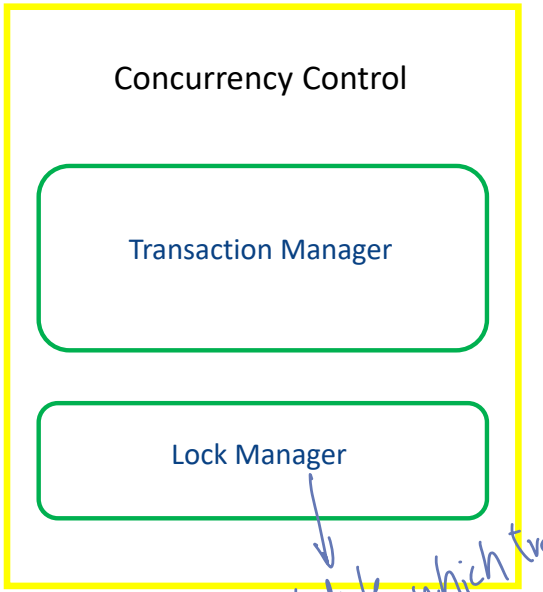
- The original SQL query is transformed into slightly different version of original SQL code which is fully equivalent and more efficient

SQL parsing is performed by the **query optimizer**, which analyzes the SQL query and finds the most efficient way to access data

An **access plan** is the result of parsing a SQL statement; it contains a series of steps the DBMS will use to execute the query and return the result set in the most efficient way

- If an access plan exists for the query in SQL cache, the DBMS reuses it
- If there is no access plan, the optimizer evaluates various plans and chooses one to be placed in SQL cache for use

Concurrency Control



Manages the work of the DBMS.

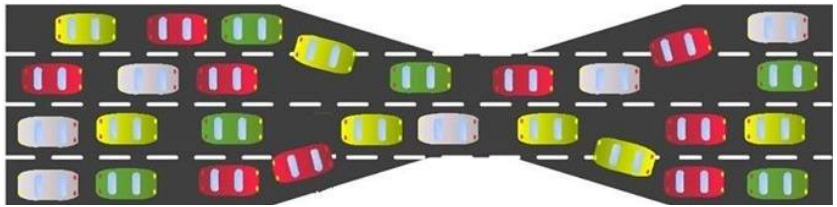
Transaction Manager handles all aspects of the SQL transaction - which DBMS user / process wants *which* resource

Lock Manager maintains a list of what resources are locked and by which user at what level (and who is waiting)

Not only tables, indexes

- buffers, cursor, memory addresses of resources

Concurrency Control



Essential to manage large scalable DBMS

Enables 1,000,000s of concurrent users

Like a Traffic Policemen controlling the flow of traffic

- Who can do what (allowed to do what they need to do)
- Who has to wait (queue)
- Who can travel through the intersection concurrently
  - Usually *readers* of data

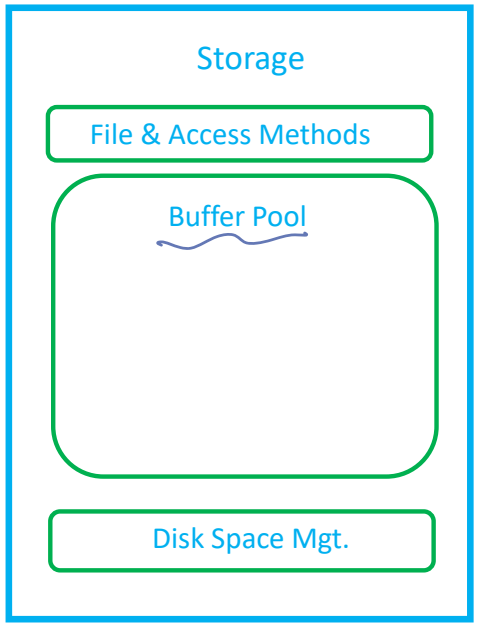
Which transactions have completed, in progress, compiled.

- What resources are involved with that transaction
- Who last used, is using and wants those resources
  - SQL, Cursor, Index, Table, Rows, File Access, Recovery Logs

a record exclusive will slow down the database

you control whether it is exclusive or not

Storage



File and Access Methods

- Disk to Memory to Disk
- Read a buffer (Database processes request pages from the buffer manager)

Buffer Pool

- a main-memory area used to cache database pages
- Data in memory
  - Row data
  - Index data
- Buffer manager minimizes the number of secondary memory accesses by keeping needed pages in the buffer

Disk Space Management

- How to organise growth of data on disk efficiently by writing efficiently.

Paging

A technique for memory organisation

Programs that are too big to fit on a single page or take too much memory as a whole are split into equal size blocks – pages

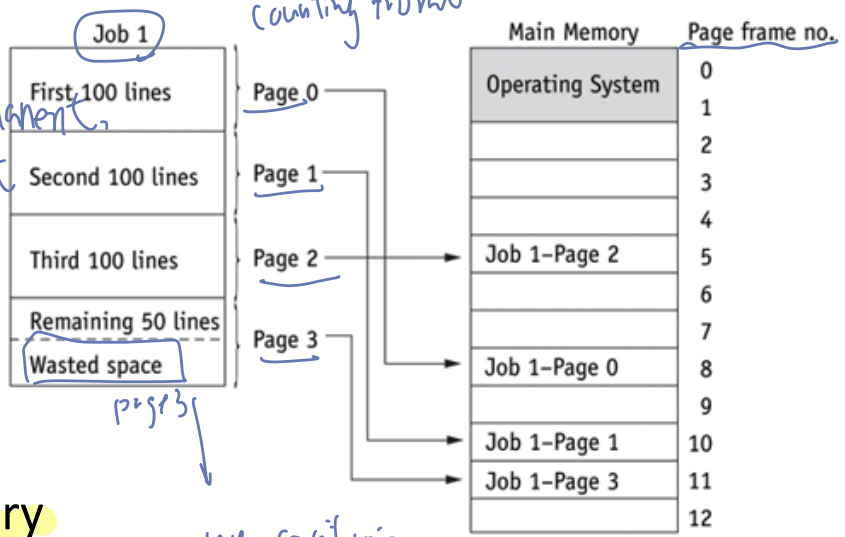
4K and 2K are the most commonly used sizes

Paging allows programs to use non-contiguous memory

It is not necessary for the whole program to be in memory

To use paging, the OS memory manager:

- Breaks physical memory into Frames
- Breaks logical memory into Pages of same size
- Uses a Page Table to link Pages to Frames

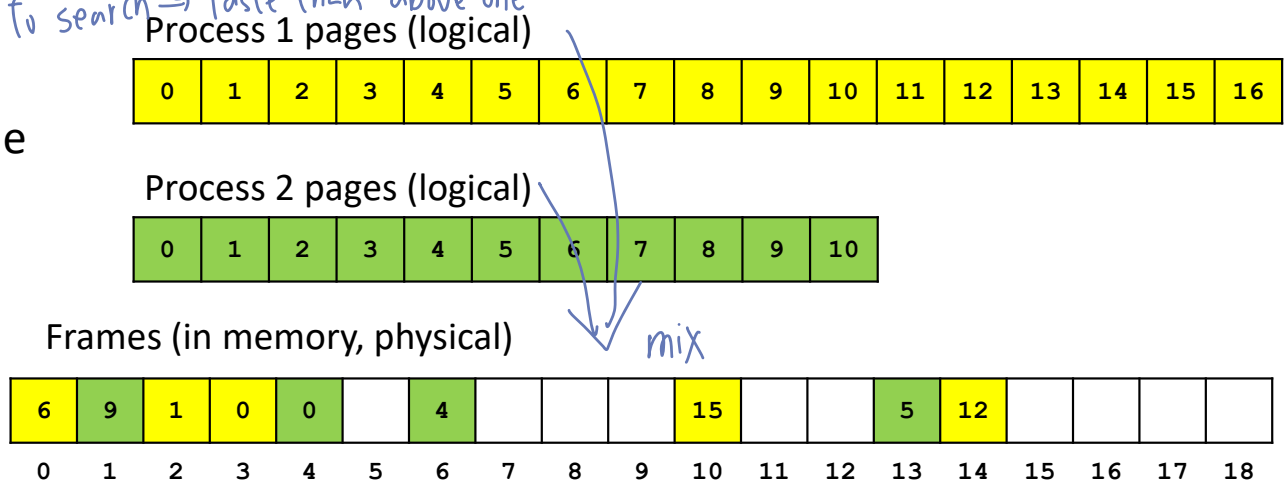


Files, Pages and Frames

A process consists of a program, associated data and the process context

How to access the file → the are different algorithm, programme know how to deal with it.

- Full Scan (full table scan)
  - Reads the entire table sequentially, from the first row to the last, one row at a time (slowest)
- Partial Scan (index range) index use to search → faste than above one
  - Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan)
  - Using an index to scan a range of values, e.g. (2 < deptno < 9)
- Page only (file header and page)
- Read the index and data file



Storage - Buffer Pool and Disk Space

Buffer Pool → is a part of memory allocated to database system. management

- Table data and indexes are read from disk into the buffer (dedicated area of memory)
- May contain multiple copies of the same data → like dirty data of transcation, you commit but not finished it can roll back and not save.
  – Need to know which copy is the current committed version

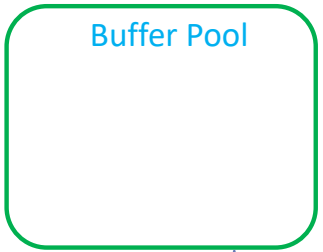
Disk Space Management

- How to allow files to grow on disk (and set max growth size) where data / index need to be save
- File organisation
  – e.g. index reorganisation;
  – varchar growth, e.g. Brown (5) grows to Nicholson (9) for last name (algorithm work)

5 byte ↑ 9 byte ↑ algorithm

Buffer Pool

what is keep in the buffer, is what is actually being working.



Many Object Types (Tables, Indexes, recent SQL, Undo)

Each buffer contains rows, pages, etc.

the data being process

Each buffer can have one of four status types:

example: change sb from deparment ID 8 to 9

1. • Current
   - In use current committed version of data (row) — DepartmentID=8
2. • Active
   - Most recent change (may not be committed) — DepartmentID=9 copy chang to 9

     COMMIT (Current: DepartmentID=9)
3. • Stale
   - An old version of the data — *DepartmentID=8*
4. • Aged
   - Old and about to be removed from buffer pool — *DepartmentID=8*

It'll be delete

Certainly! In the context of buffer pool management within a database system, the terms **current**, **active**, **stale**, and **aged** refer to specific statuses or states of pages in the buffer pool that describe their usage and relevance. These states help the database manage its cache efficiently, ensuring data integrity and optimal performance. Let's explore each status in detail:

1. Current

- **Definition**: A "current" page in the buffer pool is one that is actively being used or has been very recently used in a transaction or query. This status indicates that the data on the page is up-to-date with the latest changes in the database.
- **Usage**: Current pages are typically locked in the buffer pool while in use to prevent other transactions from modifying them concurrently. They are the highest priority for staying in the buffer cache due to their immediate relevance to ongoing operations.

2. Active

- **Definition**: An "active" page is one that is not presently being used but is still considered important because it was recently used or is likely to be used soon. Active pages may contain changes that have not yet been written to disk (i.e., they might be dirty).
- **Usage**: Active pages are kept in the buffer pool to avoid the cost of reloading them from disk should they be needed again shortly. They are usually protected against removal from the cache until they become less likely to be accessed soon.

3. Stale

- **Definition**: A "stale" page is one that holds data that has not been accessed for some time and does not reflect recent changes made to the database. Its contents are still correct according to the last read operation but are not guaranteed to be updated.
- **Usage**: Stale pages may be kept in the buffer as lower priority compared to current and active pages. They are candidates for eviction if their data is not requested after a certain period, especially under memory pressure, but they might be refreshed or reloaded if accessed again.

4. Aged

- **Definition**: An "aged" page is one that has been in the buffer pool for a long time without being accessed. This status indicates that the page is least likely to be used in the immediate future.
- **Usage**: Aged pages are the primary candidates for eviction from the buffer pool when new pages need to be loaded from disk, and there is no free buffer space available. These pages are often removed first to make room for more relevant data.

Managing Buffer Pool States

The buffer pool manager in a database system often uses algorithms to transition pages between these states based on usage patterns, frequency of access, and the nature of transactions. Common strategies like LRU (Least Recently Used), MRU (Most Recently Used), and LFU (Least Frequently Used) might be employed to decide which pages should be aged out or promoted to more active statuses.

Understanding these states helps in optimizing database cache management, ensuring that frequently accessed data is kept close at hand while less important data is cycled out, thus balancing memory use and access speed efficiently.

Log Manager (done automatically)

Recovery

Log Manager records ALL changes

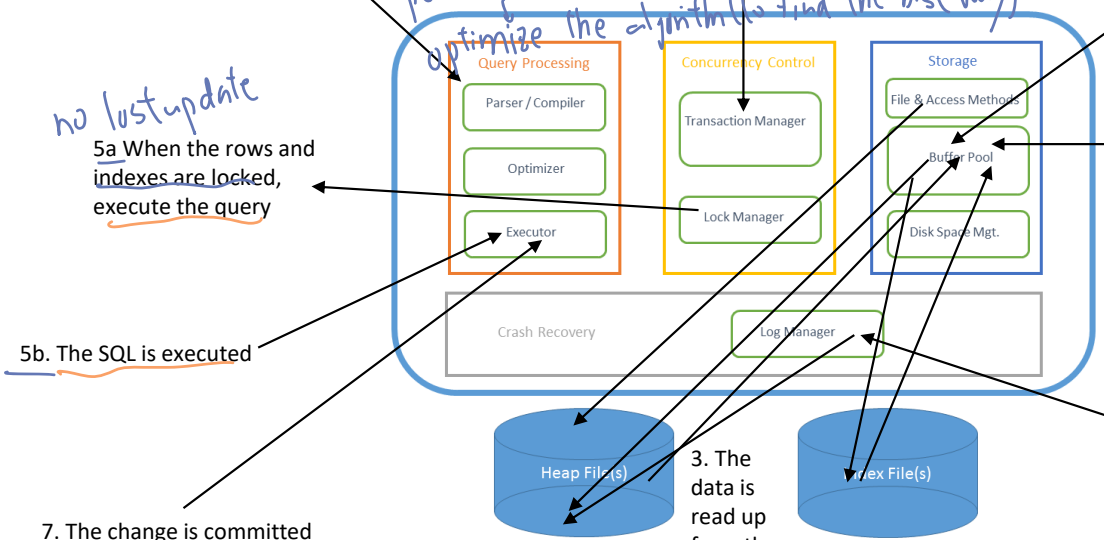
- Statement
- Transaction
  - Statement (update, select ...)
  - Rollback values (there is way to rollback)
  - Before and After values (original value and after excute value)
  - Timestamp begin (when commit happen)

    transaction, savepoint* and commit timestamps
- Database
  - Data Dictionary Changes

* Savepoint - a point in the request that specifies the state of the database at that time

DBMS – How a transaction works

```
UPDATE employee
SET departmentid = 9
WHERE firstname = 'Nancy'
AND departmentid = 8;
```



1. The code is parsed and optimised Permissions for data access are checked

2. DBMS checks if the table and index data (rows) are in the buffer pool. If it is, skip to 4. If not, go to 3.

3. The data is read up from the table and index on disk

4. The Transaction manager ensures all the tables and index data is in the buffer pool and ready to be used in the execution of the SQL.

5a When the rows and indexes are locked, execute the query

5b. The SQL is executed

6a In the buffer pool record the before value, the after value and the change to the value

6b Record a timestamp and the before value, the after value and the change to the value for all transactions in the Log Manager

7. The change is committed to make it permanent

8. The changed data and transaction log are written to disk

What affects database performance?

1. Caching data in memory, e.g. data buffers
2. Placement of data files across disc drives
3. Fast reliable storage (e.g. SSD, RAID - redundant array of independent disks)
4. Database replication and server clustering
5. Use of indexes to speed up searches and joins
6. Good choice of data types (especially PKs)
7. Good program logic (no long running CRUD)
8. Good code (no deadlocks)

Remember

```
SELECT *
FROM Buyer
WHERE BuyerID IN
  (SELECT BuyerID
   FROM Offer
   WHERE ArtefactID = 1)
```

VS

```
SELECT *
FROM Buyer b
INNER JOIN Offer o
ON b.ID=o.BuyerID
WHERE ArtefactID = 1
```

Caching Data in Memory

our goal is to minimise reads and write.

Data and Code found in memory

Avoids a read from Disk (better to read in buffer pools.)

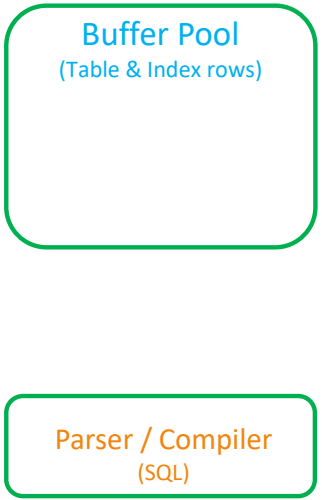
Reads are expensive

Goal in to minimize reads (and writes)

- Writes (also expensive) are necessary (recovery logs, changed data)

“in memory databases”

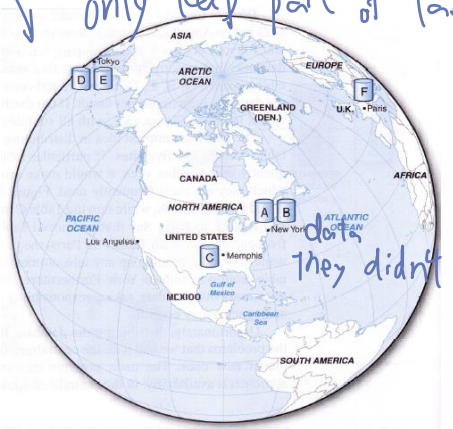
- All code and all data loaded into memory on database start and stays until shutdown or crashes
- Help with increasing performance demands of modern database applications

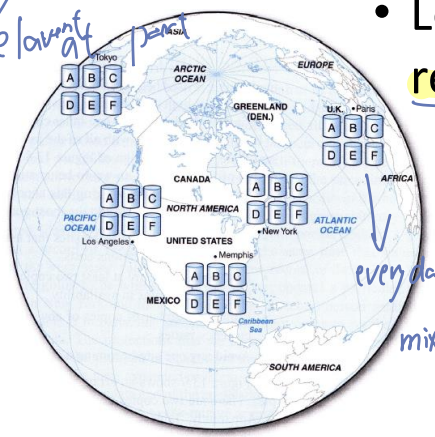


Distribution and Replication

Distributed data

- Spreads the load
- Data kept only where it is needed
- Less work per physical server – faster response times





Replicated Data

- Database replication is the frequent electronic copying of data from a database in one server to a database in another
- Spreads the load
- Less work per physical server – faster response times

Good Execution Plans

best plan

⇒ less cost

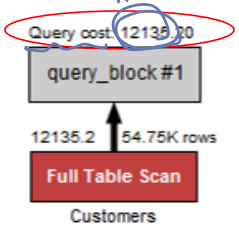
The best execution plan has the lowest "cost"

Known as Cost Based Optimization (CBO)

```
SELECT cust_first_name, cust_last_name,
cust_marital_status, cust_city, cust_income_level
FROM Customers
WHERE cust_last_name = 'Parkburg'
AND cust_first_name = 'Peter'
AND cust_city = 'Trafford';
```

This one take a while because customer is a big table

	cust_first_name	cust_last_name	cust_marital_status	cust_city	cust_income_level
	Peter	Parkburg	single	Trafford	H: 150.000 - 169.999



have 5 digit

position of where clause also important like put where the city first is faster. because city has less rows. then search the subset in city

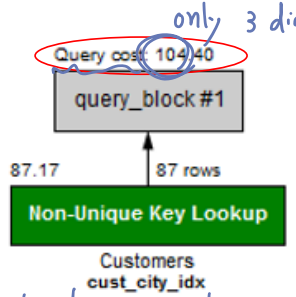
Index on where condition (cust_city) improves cost:

```
CREATE INDEX cust_city_idx
ON Customers(cust_city);

SELECT cust_first_name, cust_last_name,
cust_marital_status, cust_city, cust_income_level
FROM Customers
WHERE cust_last_name = 'Parkburg'
AND cust_first_name = 'Peter'
AND cust_city = 'Trafford';
```

create the index of city

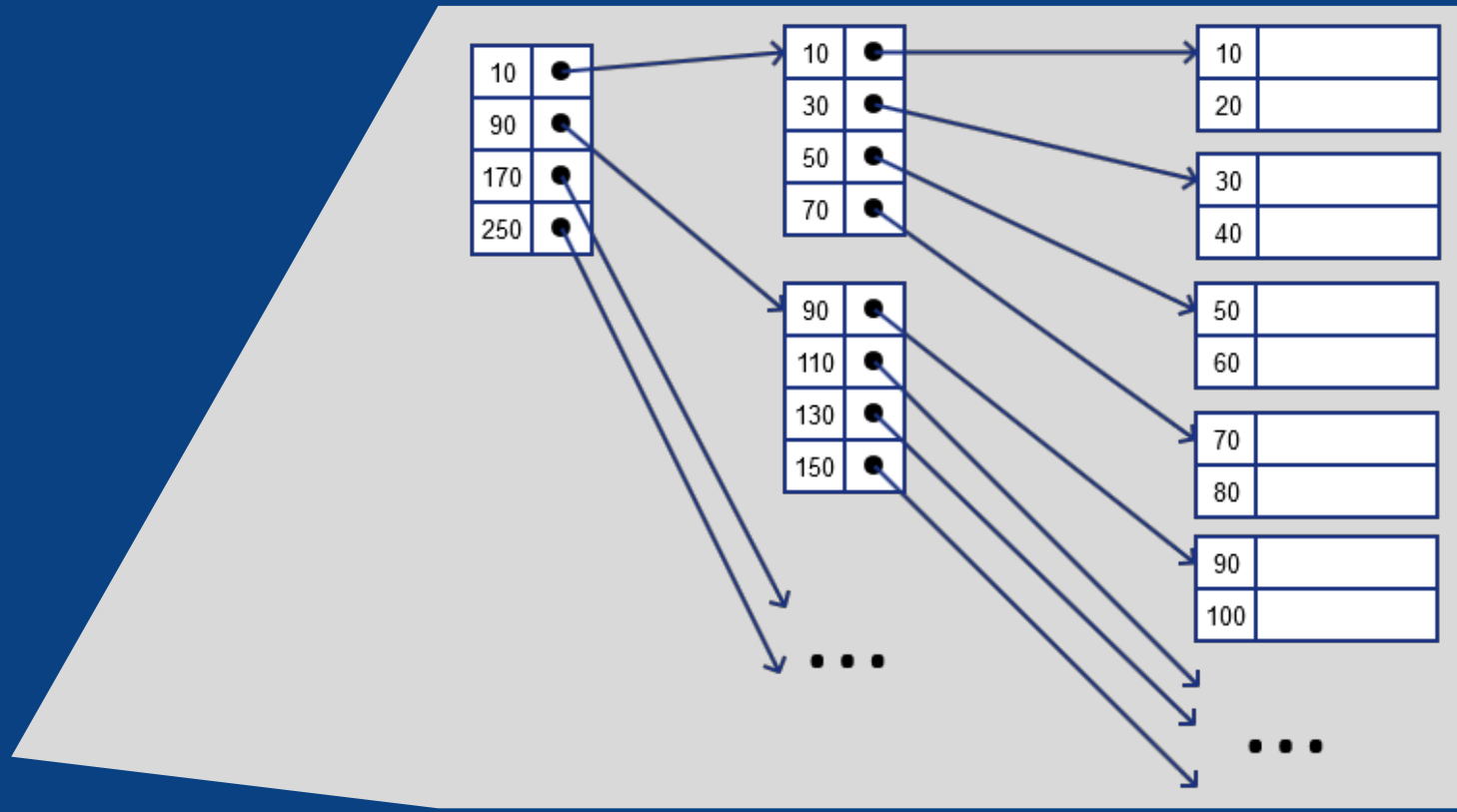
find the city and then search within that city for the customer surname.



only 3 digit

THE UNIVERSITY OF MELBOURNE

Indexing



Indexes

Imagine that this is a list of employee ids.

Do these employees exist?

- 502
- 91

It is hard and slow work to **sequentially** view **every** value.

247	128	117	391	404	533	313	519	421	105	335	291	597	369	279
204	327	308	544	338	166	38	299	35	76	250	332	237	40	446
119	85	141	466	124	352	581	478	364	590	149	272	593	424	9
229	399	95	17	256	282	569	137	226	129	532	241	146	235	213
261	188	540	151	429	206	219	216	592	62	576	599	56	528	554
314	414	316	527	81	288	74	575	197	531	578	97	458	351	99
12	409	281	507	566	329	349	134	427	198	212	118	555	47	202
570	304	167	66	448	114	111	260	72	264	562	183	23	476	136
559	365	377	498	456	143	543	172	78	302	596	534	334	195	51
469	385	238	67	549	169	267	127	315	20	189	270	106	2	560
488	357	552	236	435	395	14	443	140	159	463	587	1	70	411
64	7	402	513	179	485	346	298	171	152	186	225	25	86	21
384	438	295	522	482	185	269	461	44	471	493	294	432	158	274
474	516	310	336	564	232	415	495	447	122	115	451	537	480	553
222	341	387	41	491	462	92	253	48	165	323	203	32	155	328
211	381	354	305	57	572	244	31	325	200	557	500	396	356	380
150	209	192	389	28	510	126	457	181	376	297	401	584	502	162
83	545	130	431	430	367	147	524	284	108	358	394	504	406	177
321	454	156	547	440	287	405	89	373	588	444	419	343	370	132
318	37	4	175	317	276	54	417	59	71	368	259	102	164	361

Indexes

Do these employees exist?

sorted table

- 91
- 502

The search is **much easier** when the values are stored in **numeric sequence**.

before search good to sort

1	44	85	127	159	197	237	282	318	357	395	432	474	524	560
2	47	86	128	162	198	238	284	321	358	396	435	476	527	562
4	48	89	129	164	200	241	287	323	361	399	438	478	528	564
7	51	92	130	165	202	244	288	325	364	401	440	480	531	566
9	54	95	132	166	203	247	291	327	365	402	443	482	532	569
12	56	97	134	167	204	250	294	328	367	404	444	485	533	570
14	57	99	136	169	206	253	295	329	368	405	446	488	534	572
17	59	102	137	171	209	256	297	332	369	406	447	491	537	575
20	62	105	140	172	211	259	298	334	370	409	448	493	540	576
21	64	106	141	175	212	260	299	335	373	411	451	495	543	578
23	66	108	143	177	213	261	302	336	376	414	454	498	544	581
25	67	111	146	179	216	264	304	338	377	415	456	500	545	584
28	70	114	147	181	219	267	305	341	380	417	457	502	547	587
31	71	115	149	183	222	269	308	343	381	419	458	504	549	588
32	72	117	150	185	225	270	310	346	384	421	461	507	552	590
35	74	118	151	186	226	272	313	349	385	424	462	510	553	592
37	76	119	152	188	229	274	314	351	387	427	463	513	554	593
38	78	122	155	189	232	276	315	352	389	429	466	516	555	596
40	81	124	156	192	235	279	316	354	391	430	469	519	557	597
41	83	126	158	195	236	281	317	356	394	431	471	522	559	599

Sequential Record Access

Read each record one by one from the beginning of the table until the required record is found.

- The DBMS may have to evaluate 100,000 separate records.
  - Assume that each record is very large.
  - Assume each evaluation requires one disk I/O. *
  - Disk I/O is the slowest computer activity.

- On average, a search for an employee will require 50,000 disk I/Os.

- ** - In reality, a DBMS will read records as **blocks** of data (e.g. 2K). Therefore many records can be read into memory with a single disk I/O*

Indexes

If the index did not exist, DBMS would have to **view each record** in the employee table **sequentially** to find a matching employee id.

With 100,000 records, that could be a long time (a second or two).

If there were 10 million rows, the delay would increase.

The **more rows** we have…. the **slower the sequential search.**

read on row by one row

Suppose we create an index on the **Employee Id** of the Employee table

An **index** stores

Index will have details about every row (small table)

- The **index value** (the employee id) but only two columns
- The **location** of the record within the database

1. index value: record number
   meaning like PK
2. where it's on the disc

When DBMS **searches** for an employee id

- It quickly finds the employee id in the index (if it exists)
- Gets the location of the record
- Retrieves the record

Inserting data into a simple Index

When a table has a **Primary Key**, an **Index** based on that Primary Key is **automatically** created.

done by default.

As each row is **added**, the DBMS automatically **inserts** the search **key** and the **row location** information into the **index file**.

Insert 1st row. Employee 109

A **row is inserted** into the Employee table
The **search key** and **row location** is added to the PK index

Table

primary key

RowLoc	EmpId	FirstName	Surname	Dept
1	109	Bronwyn	Harris	Mktg

Index

Key	RowLoc
109	1

Inserting data into a simple Index

Insert 2nd row. Employee 13

A row is **inserted** into the Employee table - The row is **added** to the table.

The **search key** and **row location** is added to the PK index
The PK Index is **automatically maintained** in PK sequence

don't need to suit.
It's suted at the time of insertion

Table

RowLoc	EmpId	FirstName	Surname	Dept
1	109	Bronwyn	Harris	Mktg
2	13	Dave	Rigg	IT

pic sequence

Index

Key	RowLoc
13	2
109	1

Inserting data into a simple Index

Insert 3rd row. Employee 27

A row is **inserted** into the Employee table - The row is **added** to the table.

The **search key** and **row location** is added to the PK index
The PK Index is **automatically maintained** in PK sequence

Insert 4th row. Employee …

Table

RowLoc	EmpId	FirstName	Surname	Dept
1	109	Bronwyn	Harris	Mktg
2	13	Dave	Rigg	IT
3	27	Debra	Shaddock	Admin

In the index you can sort the table in the order you want.

numeric

like PK is numeric sort from small to large.

Index

Key	RowLoc
13	2
27	3
109	1

The Index

Obviously, the Index is **much smaller** than the table data

Searching the index is **quicker** than sequentially searching the table

The Index is in key sequence. This makes searching quick.

index

in primary key

in increasing order

and p/c location of the record on the disc

Key	RowLoc
13	2
23	13
27	3
47	12
56	10
87	8
88	4
109	1
136	7
145	6
215	9
289	11
340	5

RowLoc	EmpId	FirstName	Surname	Dept
1	109	Bronwyn	Harris	Mktg
2	13	Dave	Rigg	IT
3	112	Debra	Shaddock	Admin
4	88	Jenny	Black	Mktg
5	340	Karen	Charles	Admin
6	145	Lisa	Church	IT
7	136	Naomi	Allen	Admin
8	87	Nick	Taylor	Admin
9	215	Peter	Mills	Mktg
10	56	Steve	Avery	IT
11	289	Susan	Armstrong	Admin
12	47	Terry	Ymer	Admin
13	23	Tim	Wade	Admin

Searching the Index

Locating the row for employee 289 is **quicker** using the index.
A RDBMS may use a search method on the index such as a
Binary Search

A Binary Search continues to **split the index in half** until a match is/is not found

Start at the middleth position

The value being searched for is either
The current key value OR
A key value higher in the list OR
A key value lower in the list

example 289

greater than 88

Key	RowLoc
13	2
23	13
27	3
47	12
56	10
87	8
88	4
109	1
136	7
145	6
215	9
289	11
340	5

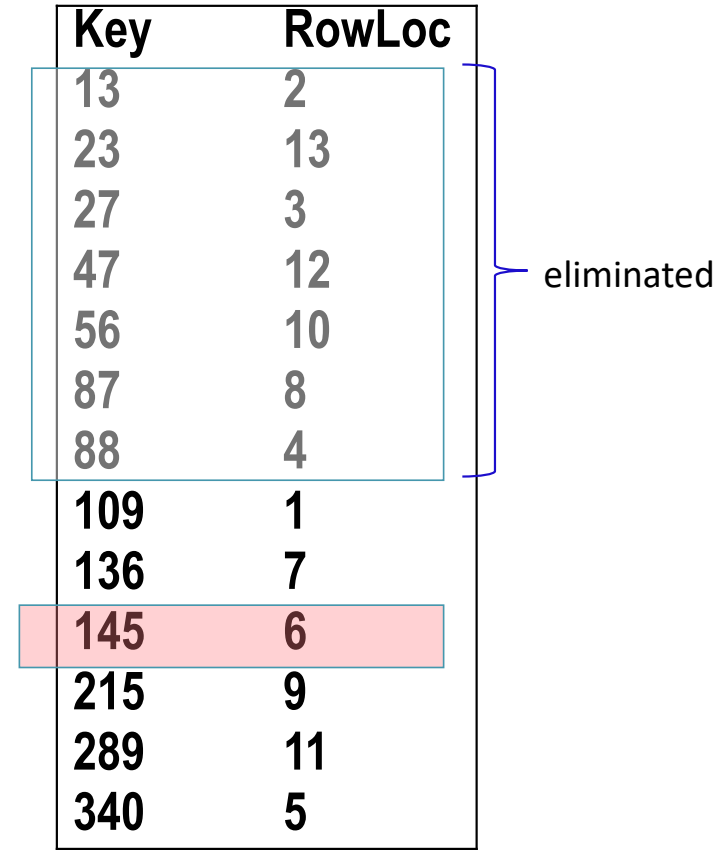
At least **half of the key values are eliminated.**
The eliminated values do not need to be searched

Searching the Index

Of the remaining key values,
examine the new middleth position

The value being searched for is either
- The current key value OR
- A key value higher in the list OR
- A key value lower in the list OR

Again, **half of the keys are eliminated.**



Searching the Index

Once a match has been found, the RDBMS can **quickly retrieve** the appropriate record

Key	RowLoc
13	2
23	13
27	3
47	12
56	10
87	8
88	4
109	1
136	7
145	6
215	9
289	11
340	5

RowLoc	EmpId	FirstName	Surname	Dept
1	109	Bronwyn	Harris	Mktg
2	13	Dave	Rigg	IT
3	112	Debra	Shaddock	Admin
4	88	Jenny	Black	Mktg
5	340	Karen	Charles	Admin
6	145	Lisa	Church	IT
7	136	Naomi	Allen	Admin
8	87	Nick	Taylor	Admin
9	215	Peter	Mills	Mktg
10	56	Steve	Avery	IT
11	289	Susan	Armstrong	Admin
12	47	Terry	Ymer	Admin
13	23	Tim	Wade	Admin

Searching the Index

Imagine that you want to **publish** all 100,000 employees in **Employee ID sequence**

The database will jump from location to location to retrieve the data in the correct sequence.

- Employee 4 is at location 97,491
- Employee 5 is at location 18
- Employee 7 is at location 43,292 …

The RDBMS is doing a large amount of work

If the table was **not indexed**:

- How would the RDBMS find the employee with the lowest ID?
- How would the RDBMS find the next lowest ID? …
- It would be very, very slow.

Key	RowLoc
4	97,491
5	18
7	43,292
14	6,390
17	431
18	72,103
23	83,242
27	12,378
36	70,231
45	383
58	2,834
79	27,844
124	84,234

(In reality, RDBMS are very quick at processing records. If a table only has a few thousand rows, the speed of using an index or using a sequential search will almost be identical.)

Advantages of using an Index

1. Using an index is a **quick method** to locate rows.
2. Large portions of the index (or the entire index) may be read into the computer's **memory** at once before searching for rows in a table.
3. **Fewer disk I/Os** (input/outputs – the slowest function of a computer).
   - This reduces the number of disk I/Os required to read table rows
   - This **speeds up retrieval time**

In our example, finding employee 289 took **2 disk I/Os.**

- One to read the index into RAM
- One to read the 11th row in the table

Additional information

- A **binary search** is only one type of search method available to a DBMS
  - **In reality, far more complex indexing options exist**
  - **Most RDBMS may have multiple options / methods of indexing tables**
    - The option selected often depends on an understanding of the data values that will be stored.

- It is **not** possible to **display** the contents of an index
  - **Indexes may be stored in a non-human readable format and/or a compressed format to reduce their size and improve search speeds**

Additional information

The index does **not** contain a **record number**.

- The record number we see in the datasheet doesn't really exist!

- Fred Blogs data may be stored in Record 3.
  If Record 1 & 2 are deleted, then Fred becomes Record 1.
- So the Record Number is a poor way of identifying a record
- (That's one reason we use Primary Keys).

Instead, the RDBMS uses a **physical address** on disk to specify the location of the record.
This is the value of RowLoc in the previous slides

address on hard disc

- In **Oracle RDBMS** the address is something like AAAYUzAAEAABIjLAAJ
- MS Access makes it almost **impossible** for you to view the physical address.

Indexes speed up report generation

There are many IT database stories involving reports that

- Use many tables with Foreign Key lookups
- Involve millions or billions of database records
- Take hours to produce

A daily report that takes more than **10 hours to run** may be next to useless.

- Such reports do exist!

Creating indexes on fields central to the report (e.g. used in WHERE and HAVING clauses) have often reduced report generation time from many hours to a few minutes.

The cost of creating Indexes

Why not index **every** field?:

- **Disk Space** - a minor consideration. Disk space is cheap.
- **Update-Time** - a **major** consideration.
  - When records in a table are inserted, changed, deleted:
    - **Every index must be updated**
      so that the index remains in key sequence
    - **Each change takes time to complete**

Too many indexes can make **unacceptable** delays

- Imagine each **Subject** that you **enrol** in, causes a few seconds delay
  - **as all of the indexes are updated**
- Imagine each time **Australia Census** data is recorded there is a delay as all indexes are updated
  - Users will **not be happy**

Deletion / Recreation of Indexes? cost little time

Database management in an organisation often involves compromises

An organisation may run a once-a-week report.

The report requires many indexes to run acceptably

It may make sense to

- **Create** the indexes prior to running the report
    - **it make take a few minutes for each index to be created**
- **Run** the report
- **Drop** the indexes

Often such tasks can be **scheduled** to automatically take place

primary index is very long → split into segment

A multi-level index

Some large indexes may be split into smaller index segments.

An index entry is created for all the segments.

It's an index of indexes.

Task: Find employee 145

LastKey	IndexSeg
56	1
112	2
215	3
340	4

from 112 to 215

Index Seg	Key	Row
1	13	2
	23	13
	45	14
	47	12
	56	10
2	85	15
	87	8
	88	4
	109	17
	112	3
3	136	7
	145	6
	171	1
	201	16
	215	9
4	289	11
	291	23...

Row	EmpId	FirstName
1	171	Bronwyn
2	13	Dave
3	112	Debra
4	88	Jenny
5	340	Karen
6	145	Lisa
7	136	Naomi
8	87	Nick
9	215	Peter
10	56	Steve
11	289	Susan
12	47	Terry
13	23	Tim
14	45	Simon
15	85	Bruce
16	201	Debra
17	109	Belinda
18	333	Michael
19	401	Dave

DBMS Optimisers

DBMS systems have **optimisers**.

Optimisers are designed to decide the **quickest** and most **efficient** way of accessing data.

Multiple indexes may exist, but it is the **DBMS optimiser that decides** if it is of benefit to use the indexes.

Imagine we have a table with a **small number of rows.**
We have indexes based on the Employee Id and Surname.

Imagine that it only takes one or two reads to load the entire table into memory,

It may be quicker to read/load all rows into memory
compared to reading / loading the Indexes into memory and then retrieving the appropriate row(s)

Optimisers attempt to **reduce the time** taken to process the query.

When to create indexes

For each table, choose the columns you will index:

- *queried frequently* (used in WHERE clauses)
- used for *joins (PK to FK)*
- primary *keys* (automatic in most DBMS)
- foreign *keys* (automatic in MySQL)
- unique columns (automatic in most DBMS)

Large tables only - small tables do not require indexes

- if you frequently retrieve less than about 15% of the rows often don't need index

Wide range of values (good for regular indexes).

Small range of values (good for bitmap/hash indexes).

What's examinable

- What a DBA and Data Administrator do
  - And the difference in each role
- Database Architecture
  - Label all memory structures & know their role
- What affects database performance
  - Caching; Datafile placement; Indexes; Data types; Query Execution plans; Efficient code;
- When to create an index

THE UNIVERSITY OF MELBOURNE

Thank you