Database

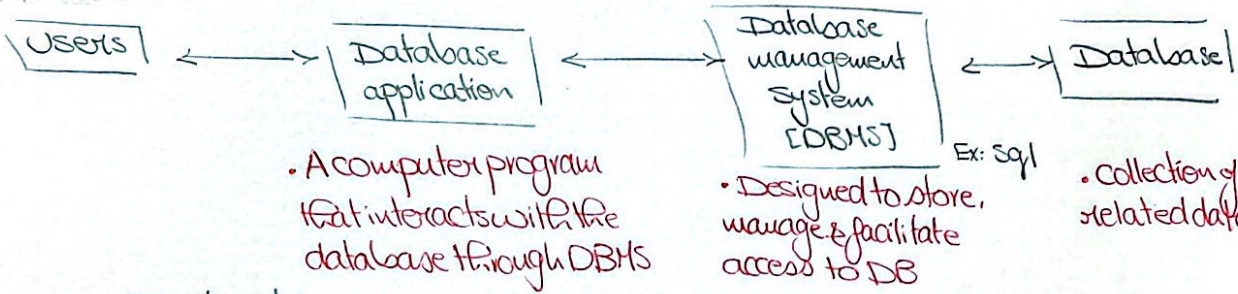
Lecture 1: *Database examples: hospital / hotel / bank / airline / business

*Data models:

① Structured data: fixed format Ex: Relational model

② Semi-structured data: some structure but not fixed Ex: XML

③ unstructured data: No structure Ex: NoSQL – text / image / audio / video



* SQL: Standard Structured query language

→ DDL: Data Definition language ⇒ create / alter / delete → drop tables

DML: Data manipulation language ⇒ insert / delete / modify tuples

DCL: Data control language ⇒ specify user permissions

• English → SQL → RA → Query tree

* why DBMS is important?
- core aspect of most science
- core need in industry
- mix of theory & systems

Lecture 2:

* heap file ⇒ ملف كومة

Sequential ⇒ ملف تسلسلي

* Advantages: speed up search

* Disadvantages: ① changes in data ↔ changes in index
② slows down insert, update, delete,

* Index is an auxiliary file (usually those in where)
↳ can be created on one or more columns
↳ is the principle technique used to efficiently answering a given query.

* index file occupies less disk blocks

* index is always sorted / less disk blocks

* Dense ⇒ has an index entry for every search key / one for every record

Sparse ⇒ has an index entry for only some search key (ordered files ولازم تكون)
↳ one for each block

Types of indexes:
- ① Single level indexes
  1- primary index
  2- clustering index
  3- secondary index
- ② Multilevel indexes

→ Main Memory

B tree & B+ tree

- Primary index: ① ordered file — one primary index for Data file
  ② primary key is the search key * sparse
- Clustering index: ① ordered file * بيشاور على أول block ظهرت فيه الـ data
  ② Not necessarily unique, but must be not null
  * sparse
- Secondary index: Any data file type
  * file can have many secondary indexes * dense
  Less efficient than primary indexes [has more space]
  * with key or with non key
- Multilevel index: faster search — ordered بيكون index الـ level في
  (search tree) slower updates [insertion & deletion problem]
  ↳ used when the index file is too large to fit in memory.
- B- Tree: * Balanced tree * 3 level
  * n key → n+1 pointers

* Primary key is a unique identifier of records in a table
* join is a relational operation that causes two or more tables with a common domain to be combined into a single table or view.
* RA expression → a sequence of relational algebra operations
* π- degree: number of select attributes.
* join ⇒ combine tables
  Union ⇒ combine tuples
* Union & intersection & minus:
  1. Binary operation (two operands)
  2. Must be compatible
  3. Duplicates are removed
* create index [index name] on [relation name]
* drop index [indexname]

* join:
  ① cross-join = cartesian product
  ② Equi-join → common colums appear redundantly. (=) / Theta-join (>, <, ≤, ≥)
  ③ Natural join → one of the duplicate columns is eliminated.
  ④ outer join → some rows not satisfying the condition appears in the result set.
  ⑤ Inner join → only rows satisfying the condition appears in the result set.

	ordered file	unordered file
key	primary key	secondary with key
Non key	clustering key	Secondary with non-key

Lecture 3:

* SQL: specify what is wanted, not how it is obtained.
* Select operator: produce table containing subset of rows
* Selection condition:
  1. Simple selection condition → ① Equality comparison ⇒ =
     → ② Non equality comparison ⇒ >, <, !=
  2. Conjunction [AND]
  3. Disjunction [OR]
  4. Not

* AND: B2 B3 AND B1 B2 → B2
  B2 B3 AND B1 B2 AND B1 → ∅
* OR: B1 B3 OR B2 B3 OR B4
  ↳ B1 B2 B3 B4 [with duplication]
* Linear search [brute force]:
  Scan the data file, block by block looking for records
* Binary search → ordered file

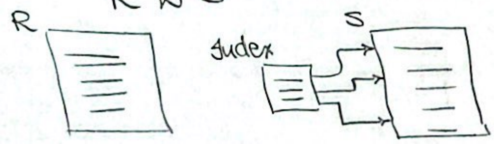
Lecture 4.

* project operation: produces table containing subset of columns. [no duplication] → unique
  ↳ using distinct
* if we want to remove duplicates, we should sort first
* Set operations: ∪ | ∩ | − | ×
* Cost of cartesian product → R has n records & j attributes
  S has m records & k attributes
  ↳ n×m & j+k [very expensive]
* compatible relations:
  1. Same number of attributes
  2. Same name of attributes
  3. Same domain of attributes
* Tuples at a time unary operators: ① selection & projection
  ② No need to bring entire relation into memory
* Full relation unary operators: ① Duplicate elimination & grouping
  ② Require seeing all or most of the tuples into memory
* Full relation Binary operators: ① Set operators: ∪ | ∩ | − | × | ⋈
  ② Require seeing the tuples of both relations in memory
* Aggregate operations: COUNT | SUM | AVG | MAX | MIN - Not RA
* clustering index is easier to use in group by operation

Lecture 5:

- Condition selectivity ⇒ percentage of the records that satisfy the condition [0,1]
  1. Equality condition on a key attribute: Selectivity 1/Relation size
  2. Equality condition on a non-key attribute: Selectivity $1/i$ → distinct values
- Algorithms for join operations:
  1. Sort merge join: files are <u>ordered</u> on join attribute
  2. Single loop join: one table has <u>index</u> on its join attributes

R ⋈ S



خلي الـ index جوه

  3. Nested-loop join: linear search
     ↳ no indexes

الي فيها block اقل تبقى بره احسن
records

- Factors affecting join performance:
  1. Available buffer space
  2. Choice of inner <u>vs</u> outer join relation
- Block → The smallest unit to read from disk.
- Buffer/page → A memory region with the same size of a block
  - read number of <u>blocks</u> = number of available <u>buffer</u>

Lecture 6:

- Left outer join:
  1. $T_1 \leftarrow R \bowtie S$
  2. $T_2 \leftarrow R - T_1$ [apply projection as they are not union compatible]
     - + sorting
  3. $T_3 \leftarrow T_2 \times Null$
  4. $Result \leftarrow T_1 \cup T_3$
- SQL → declaritive [does not specify how to execute]
- RA → procedural [there is an associated query execution plan]

↳ Re-writing a query to produce an equivalent but better query
  ↳ cost less: ① Disk access ② computation ③ memory usage ④ communication

- Heuristic rules:
  ↳ rules for ordering operations in query optimization.
- Cost estimation:
  ↳ estimate cost of different strategies & chooses the lowest execution cost.
- SELECT → reduce the # of tuples
- PROJECT → reduce the # of attributes

- Steps in converting a query tree:
  ① Moving SELECT operations down
  ② Applying the most restrictive SELECT operation first
  ③ Replacing cartesian product with join & join condition
  ④ Moving PROJECT operation down the query tree.

Lecture 7:

- cost components:
  ① I/O ② Storage ③ CPU ④ RAM ⑤ communication
- Histogram:
  number of values / tuples in each of a number of intervals
- Disk access: finding the first block + reading all blocks with matching records.
- ordering index with non-equality → retrieve preceding or subsequent records
- clustering index → only retrive subsequent blocks.

Lecture 8:

- cost of query plan = sum of operator costs
- conjunctive selection → use either linear search or index to solve.

Lecture 9:

- Transaction is a sequence of database actions that is considered as a unit of work
- In "ad-hoc" sql → each statement is one transaction
- ACID properties:
  ① Atomicity → performed or not performed at all
  ② Consistency preservation → user specified integrity constraints
  ③ Isolation
  ④ Durability or permanency

* DBMS ensures atomicity by undoing the actions of partial transactions.
  ↳ maintain record called log → responsible by the recovery manager
* Recoverability → safe from system crashes
* concurrency → Multi-user database access
* concurrency interleaving → time slice [serial] → ① poor system performance
  ② Low throughput
  ③ long response time
  ④ poor resource utilization
* parallel processing → several users same DB
* Advantages of concurrency:
  ① waiting time ↓
  ② Response time ↓
  ③ Resource utilization ↑
  ④ Efficiency ↑

* Concurrency problems:

① Lost update problem: an update is done to data item by transaction is lost as it is overwritten by the update done by another transaction

② Dirty read problem: one transaction updates an item of the database & somehow the transaction fails and before the data gets rollback, the updated data item is accessed already by another transaction

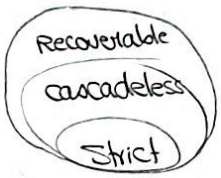
③ Unrepeatable read problem: two different values are read for the same data item

④ Incorrect summary: interleaving execution of multiple transactions that are working on same data

Lecture 10:

* Two operations are conflicting, iff: ① Different transactions ② Same data ③ one is write
* Recoverability: if transaction fails we need to undo the effect of this transaction to ensure the atomicity property.

* Recoverable schedule: write then read [commit/abort] (this is)
* cascadeless schedule: write [commit/abort] read
* Strict schedule: write [commit/abort] read
  write [commit/abort] write



Lecture 11:

* Result equivalence ~> same final database state
* conflict equivalence ~> the order of any two conflicting operations is the same in both schedules
  L~> iff the serialization graph has no cycles

- **Equations:**
  - Blocking Factor (bfr): $\frac{\text{Block size}}{\text{Record size}}$ (record/block)
  - Number of blocks: $\frac{\text{number of records}}{\text{number of records/block}}$
  - (Selection cardinality): $S = SL \times r$ (selection, selectivity) or $SL = \frac{S}{r}$ (r ← percentage)
  - Nested-loop join: $B_x + [B_x / \text{Buffers}] * B_y$
  - Linear search:
    1. worst case = b ⇝ cost of non-key
    2. Best case = 1
    3. Average = $b/2$ ⇝ cost of key
  - Binary search: cost of finding the first needed block = $\log_2 b$ [key]
    cost of finding all needed blocks = $\log_2 b + (\lceil S/bfr \rceil - 1)$ [non-key]
  - Primary index: cost = $x + 1$ (clustering)
  - ordering index with non-equality: cost rough = $x + b/2$
  - clustering index: cost = $x + \lceil S/bfr \rceil$
  - Secondary index: index on key = $x + 1$
    index on non-key = $x + 1 + S$ → unordered
    Linear search > Non-equality comparison = $x + (b_{i1}/2) + (r/2)$
    (b_{i1} ← first level index block; block of record pointers: عدد الـ pointers في الـ block)
  - join selectivity (js) = $\frac{|R \bowtie_{A=B} S|}{|R| \times |S|}$ $(0 < js < 1)$
  - Nested loop join (linear) = $\underbrace{b_R}_{r} + \underbrace{\lceil b_r / n_b - 2 \rceil \times b_S}_{\text{reading s}} + \underbrace{(js \times |R| \times |S| / bfr_{RS})}_{\text{resulting block}}$ (R ② Result)
  - Single-loop join:
    1. primary index = $b_R + (js \times |R| \times |S| / bfr_{RS}) + (|R| \times (x_B + 1))$
    2. clustering index = $b_R + (js \times |R| \times |S| / bfr_{RS}) + (|R| \times (x_B + (S_B / bf_B)))$
    3. Secondary index = $b_R + (js \times |R| \times |S| / bfr_{RS}) + (|R| \times (x_B + 1 + S_B))$
  - Sort-merge join: $b_R + b_S + ((js \times |R| \times |S| \times bfr_{RS}))$ ⟹ sorted (مش هتقرأ اكتر من مرة)

Part 2:

Lecture 1

- The two main files of a database LDF (log data file) & MDF (master data file)
- Transaction : log data file —committed / confirmed→ Master data file
- A lock is a flag that is associated with a data item.
- Lock types:
  - Binary → general lock
  - shared → several peers [read]
  - exclusive → one peer [write]

- incorrect summary → lost update + dirty read
- unrepeatable read problem ↳ lost update

- Lock manager that manage the table & grants access to data item to locking transacti
- Lock conversion : upgrade [read → write]
  downgrade [write → read]
- Two-phase locking protocol (2PL): [guaranteed to be serializable]
  → growing phase: making locks
  shrinking phase: making any operation except locking
  CAUTION: one you start unlocking, you cannot and is not allowed to request lock

Types:

① Basic 2PL:
→ you can't lock after you unlocked an item. على مستوى نفس ال transaction
→ interleaving operations
→ guarantees serializability
→ deadlock problem
→ doesn't guarantee to be recoverable, cascadeless or strict.

② conservative 2PL [static]
→ All locks are done at the beginning [no interleaving operations]
→ guarantees serializability
→ no deadlock problem
→ doesn't guarantee to be recoverable, cascadeless or strict.

③ Strict 2PL
→ Doesn't allow unlock of exclusive locks unless a commit or abort happens
→ guarantees serializability but have deadlock problem
→ guarantees to be recoverable, cascadeless and strict
→ interleaving opererations

④ Regorous 2PL
→ Doesn't allow unlock of exclusive / shared locks unless a commit or abort happen
→ guarantees serializability but have deadlock problem
→ guarantees to be recoverable, cascadeless & strict.

• لو regorous اكيد Strict • لازم اى حاجة تبقى basic

Lecture 2:

- Lost update & unrepeatable read solved by Basic / Static 2PL
- 4 problems solved by the strict / rigorous
- Deadlock: occurs when transaction is waiting for some item that is locked by some other transaction in the set.

- **Wait - Die**
  → smaller timestamp: wait
  bigger timestamp: die

- **Wound wait**
  → smaller timestamp: wound
  bigger timestamp: wait

  ] avoid starvation

- **No - wait**
  → transactions abort & restart needlessly [deadlock free]

- **cautious - wait**
  → no transaction is allowed to wait for another (waiting) blocked transactions

Lecture 3

- Timestamp : a monotonically increasing variable (integer) indicating the age of a transaction.
  → A larger time stamp value indicates a more recent event or operation

- Basic timestamp ordering algorithm:

  ⇝ if T issues W(x):
  - write_TS(x) or read_TS(x) > TS(T) ⇝ Abort T
  - write_TS(x) and read_TS(x) <= TS(T) ⇝ Execute write_TS(x) & set write_TS(x) = TS(T)

  ⇝ if T issues R(x):
  - write_TS(x) > TS(T) ⇝ Abort T
  - write_TS(x) <= TS(T) ⇝ Execute read_TS(x)
    if TS(T) > read_TS(x)
    set read_TS(x) = TS(T)
    else continue

  * هل في حد قبلي احدث مني عمل write/read لنفس الـ item ؟

- Strict timestamp ordering algorithm

  ▵ TS(T) > write_TS(x) ⇝ Delay R/W(x) until T' commit/abort
  ▵ No deadlock.

  هل في حد قبلي اقدم مني عمل write لنفس الـ item ؟

- Thomas's write rule + Basic time ordering

  ▵ It rejects fewer write operations by modifying the checks for the write_item(x) operation while read check remains the same
  ▵ Not conflict/serializable

  * هل في حد قبلي احدث مني عمل write لنفس الـ item ؟

Lecture 4:

- Failure types:
  - ▸ Transaction failure
  - ▸ System failure
  - ▸ Media failure
- caching is the process of bringing a DB block from disk to the DBMS main memory
- flushing is the process of writing a modified page from the "in memory buffer" to the disk
- Write Ahead Logging [WAL]:
  Is the recovery mechanism used to ensure that BFIM is recorded in the appropriate log entry and the log entry is flushed to disk before the BFIM is overwritten with the AFIM on disk
- Steal → permits data to be flushed anytime
  No steal → prohibits changed data be written to disk before the transaction commits
- Force → forces all pages updated by a transaction to be immediately written to disk before the transaction commits
  No Force → flushing the updated pages of a transaction before commit are not mandatory
- In-place update: writes a modified buffer to the same original file block.
  Shadow update: writes a modified buffer to a different disk block.
- when transaction fails during execution, transaction rollback / undo
  when transaction fails after execution, transaction rollforward / redo

* Deferred update:
- ▸ NO-undo / Redo
  No-steal / No-force
  ▸ postpones (defers) all the updates to the database on disk till after the transaction commits
- ▸ Redo? → committed transaction after the last checkpoint

* Immediate update:
- ▸ undo / Redo
  Steal / No-force
  ▸ transactions are allowed to commit before their changes are recorded to disk
- ▸ undo / No-redo
  Steal / force
  ▸ all updates of a transaction are recorded on disk before the transaction commits