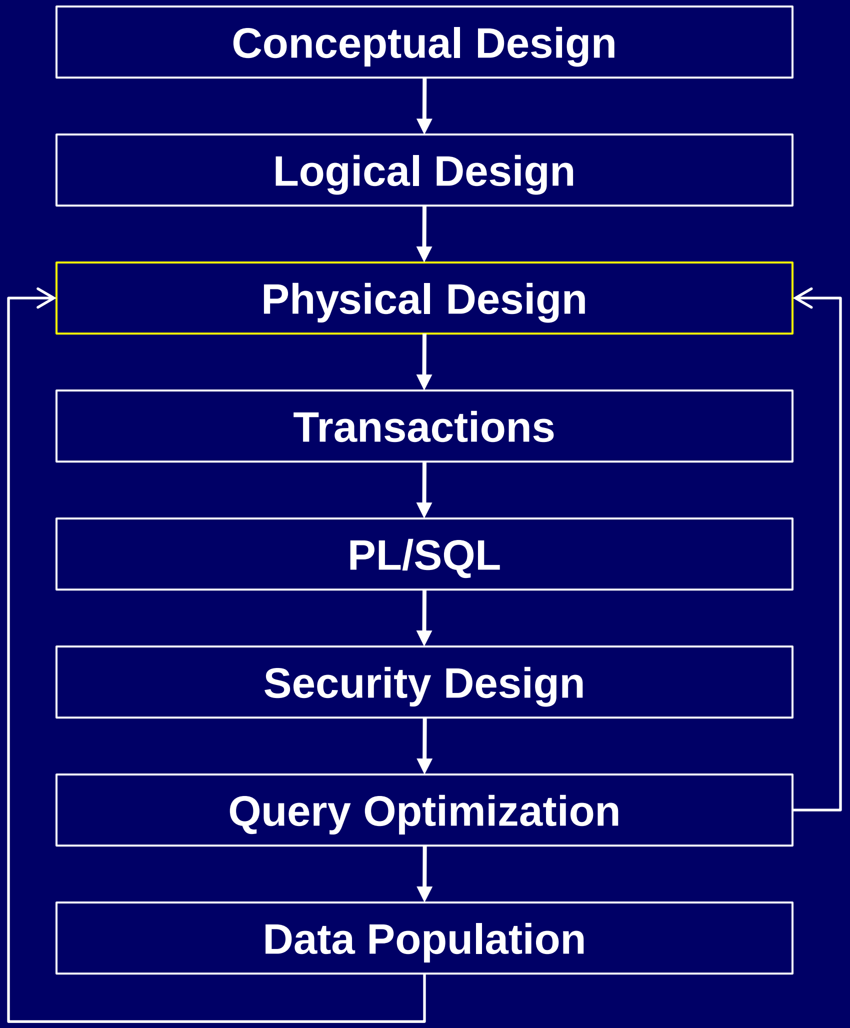
CSCD43: Database Systems Technology

Lecture 3

Wael Aboulsaadat

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

Steps in Database Design



Physical Design

A. Specify Storage parameters

B. Specify Indices

Storage and Indexing

Motivation

- DBMS stores vast quantities of data
- Data is stored on external storage devices and fetched into main memory as needed for processing
- Page is unit of information read from or written to disk. (in DBMS, a page may have size 8KB or more).
- Data on external storage devices :
  - Disks: Can retrieve random page at fixed cost

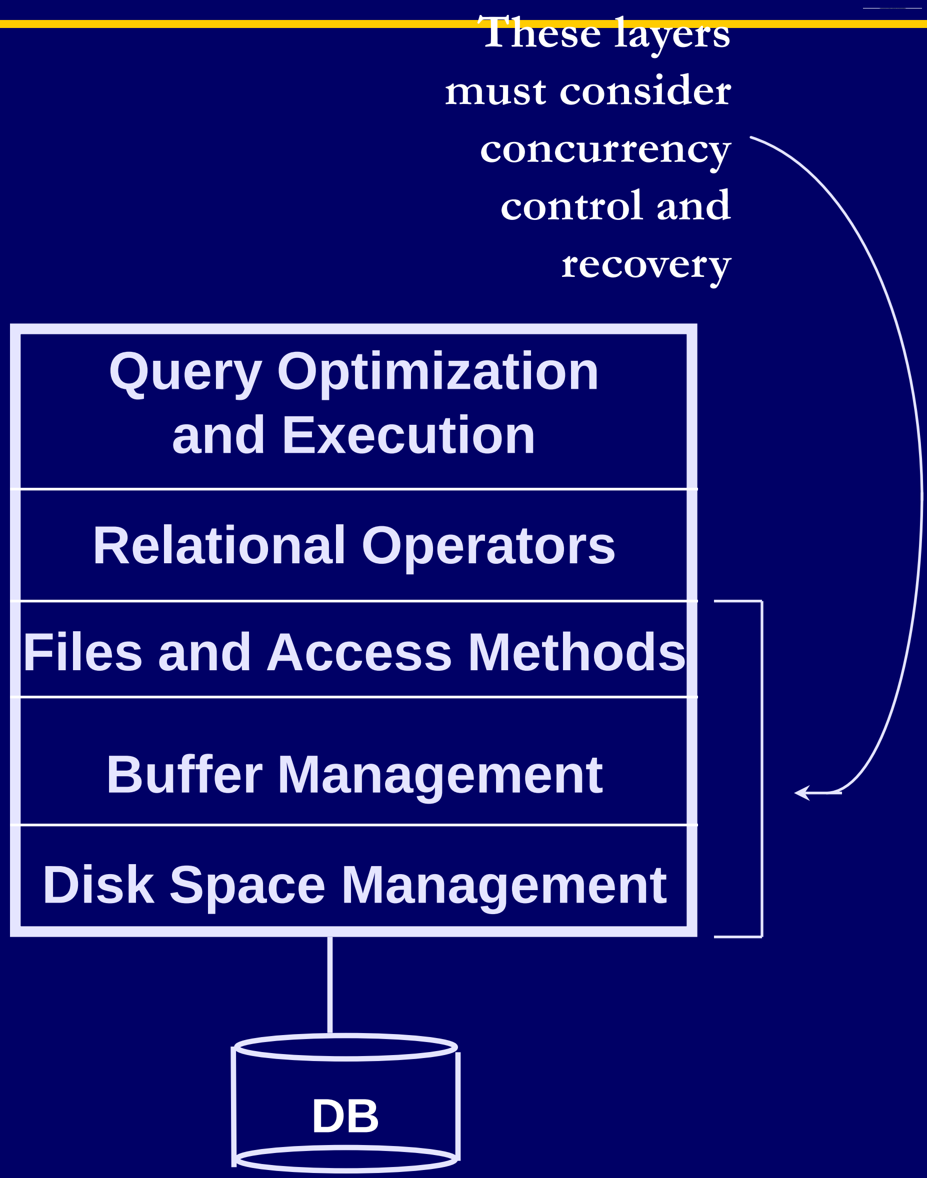
    But reading several consecutive pages is much cheaper than reading them in random order
  - Tapes: Can only read pages in sequence

    Cheaper than disks; used for archival storage

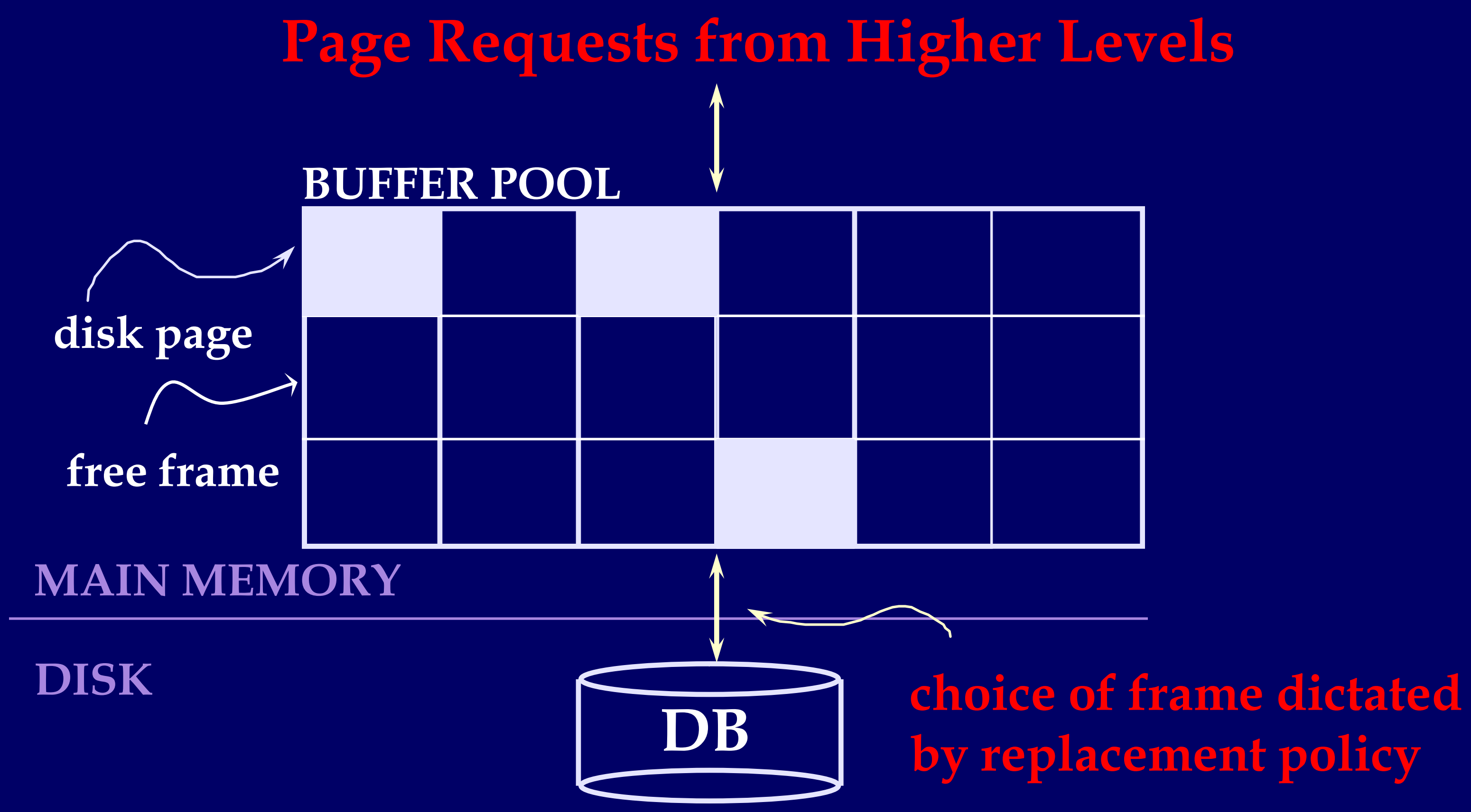
- Cost of page I/O dominates cost of typical database operations

Structure of a DBMS: Layered Architecture

- external storage access
  - **Disk space manager** manages persistent data
  - **Buffer manager** stages pages from **external** storage to **main memory** buffer pool.
  - **File and index** layers make calls to buffer manager.



Buffer Manager



- *Data must be in RAM for DBMS to operate on it!*
- *Buffer Mgr hides the fact that not all data is in RAM*

Buffer Manager

- Similar to *virtual memory manager*
- Buffer replacement policies
  - What page to evict ?
  - LRU: Least Recently Used
    - Throw out the page that was not used in a long time
  - MRU: Most Recently Used
    - The opposite
    - Why ?

Buffer Manager

- *Pinning* a block
  - Not allowed to write back to the disk
- *Force-output (force-write)*
  - Force the contents of a block to be written to disk
- *Order the writes*
  - This block must be written to disk before this block
- Critical for fault tolerant guarantees
  - Otherwise the database has no control over whats on disk and whats not on disk

File Organization for DB ?

- How are the relations mapped to the disk blocks ?
  - Use a standard file system ?
    - High-end systems have their own OS/file systems (e.g. Oracle)
    - OS interferes more than helps in many cases
  - Mapping of relations to file ?
    - One-to-one ?
    - Advantages in storing multiple relations clustered together
  - A *file* is essentially a *collection of disk blocks*
    - How are the tuples mapped to the disk blocks ?
    - How are they stored within each block

File Organization for DB

- Goals:
  - Allow insertion/deletions of tuples/records
  - Fetch a particular record (specified by record id)
  - Find all tuples that match a condition (say SSN = 123) ?
- Simplest case
  - Each relation is mapped to a file
  - A file contains a sequence of records
  - Each record corresponds to a logical tuple
- Next:
  - How are tuples/records stored within a block ?

Sequential File Organization

- Early databases
- Influenced by tape storage

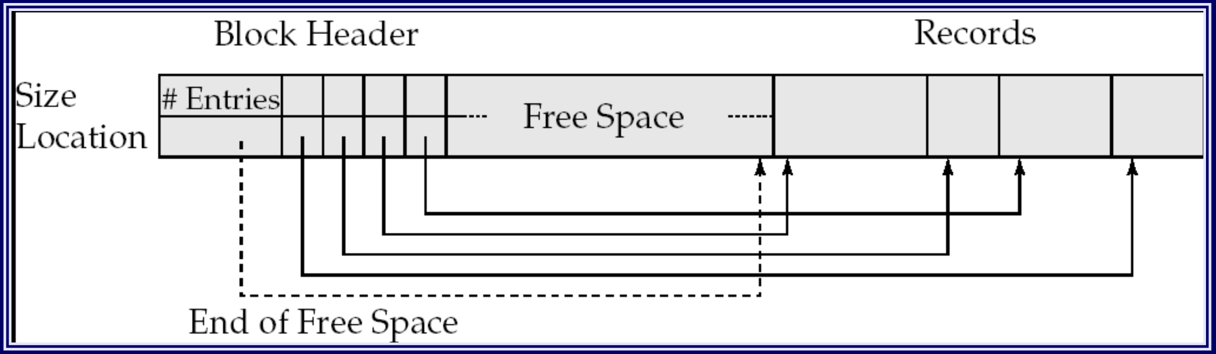
Fixed Length Records

- n = number of bytes per record
- Store record *i* at position:
  - n * (i – 1)
- Records may cross blocks
  - Not desirable
  - Stagger so that that doesn't happen
- Inserting a tuple ?
  - Depends on the policy used
  - One option: Simply append at the end of the record
- Deletions ?
  - Option 1: Rearrange
  - Option 2: Keep a *free list* and use for next insert

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Variable-length Records

Slotted page structure



- *Indirection:*
  - The records may move inside the page, but the outside world is oblivious to it
  - Why ?
    - The headers are used as a indirection mechanism
    - *Record ID 1000 is the 5th entry in the page number X*

Sequential File Organization

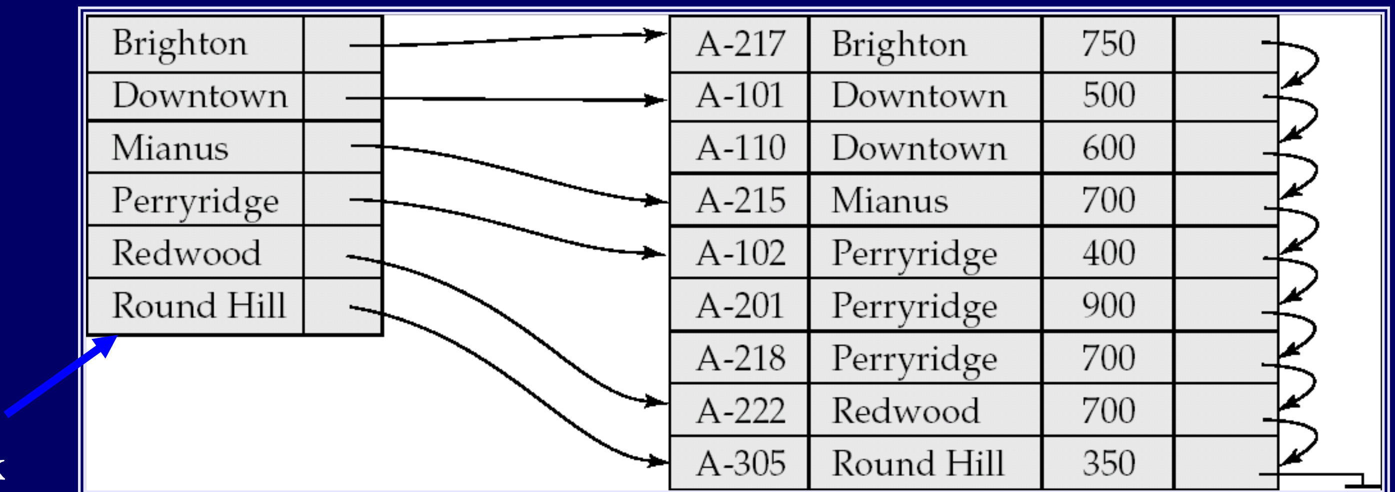
- Keep sorted by some search key
- Insertion
  - Find the block in which the tuple should be
  - If there is free space, insert it
  - Otherwise, must create overflow pages
- Deletions
  - Delete and keep the free space
  - Databases tend to be insert heavy, so free space gets used fast
- Can become *fragmented*
  - Must reorganize once in a while
- What if we want to find a particular record by value ?

Indexed File Organization

- A data structure for efficient search through large databases
- Two key ideas:
  1) The records are mapped to the disk blocks in specific ways
     - Sorted, or hash-based
  2) Auxiliary data structures are maintained that allow quick search
- Think library index/catalogue
- Search key:
  - Attribute or set of attributes used to look up records
  - E.g. SSN for a persons table
- Two types of indexes
  - Ordered indexes
  - Hash-based indexes

Ordered Indexes

- Primary index
  - The relation is sorted on the key of the index
- Secondary index
  - Not supported
- Can have only one primary index on a relation

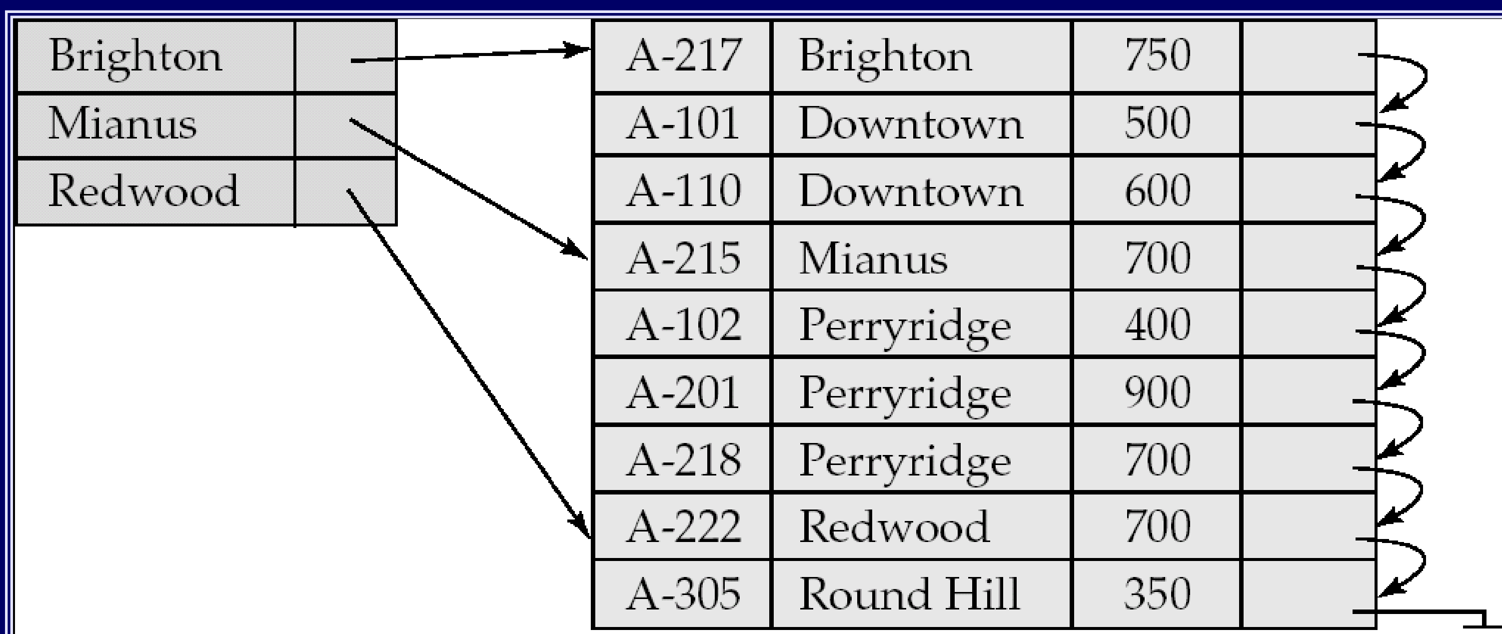


Index

Relation

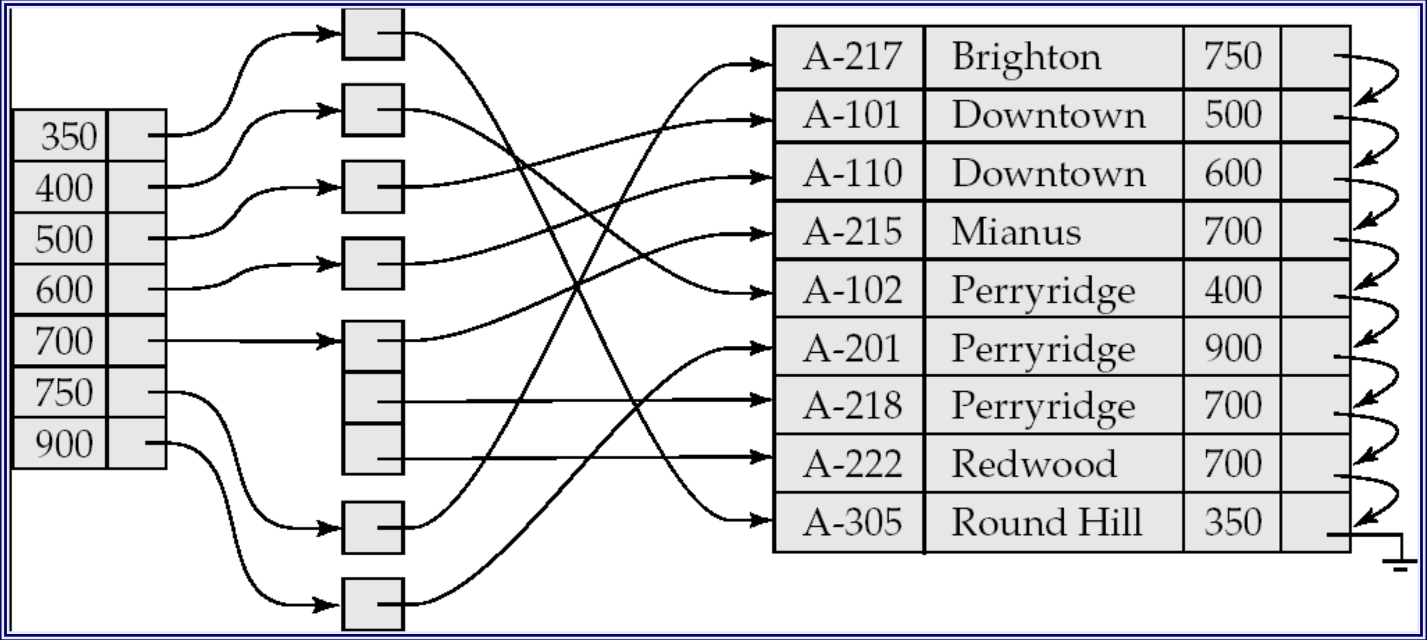
Primary *Sparse* Index

- Every key doesn't have to appear in the index
- Allows for very small indexes
  - Better chance of fitting in memory
  - Tradeoff: Must access the relation file even if the record is not present



Secondary Index

- E.g.
  - Relation sorted on *branch* but we want an index on *balance (why?)*
- Must be dense
  - Every search key must appear in the index



How to create an index in SQL ?

- Syntax

```
CREATE INDEX  Index-Name on Table-Name(Columns…);
```

- Example:

```
TABLE Customer     (First_Name char(50),
                    Last_Name char(50),
                    Address char(50),
                    City char(50),
                    Country char(25),
                    Birth_Date date)
```

```
CREATE INDEX  IDX_CUSTOMER_LAST_NAME on CUSTOMER (Last_Name)

CREATE INDEX IDX_CUSTOMER_LOCATION on CUSTOMER (City, Country)
```

How to drop an index in SQL ?

- Syntax

```
DROP INDEX  Index-Name;
```

- Example:

```
DROP INDEX  IDX_CUSTOMER_LAST_NAME;

DROP INDEX IDX_CUSTOMER_LOCATION;
```

- PostgreSQL tables

```
select * from pg_index;
```