1 Disorder is a friend of scaling

1.1 Streaming through RAM

- Instead of reading and writing a single item at a time, batch things up
- When input buffer consumed, read another chunk. When out buffer fills, write to output using f(x) in the middle
- We can simply partition the input and parallelize

1.2 Rendezvous

- Streaming one chunk at a time is easy, but some algorithms need certain items to be co-resident in memory (not guaranteed to appear in the same input chunk)
- **Time-space rendezvous**: in the same place (RAM) at the same time - most of computing is about this
- Divide and Conquer: you have B chunks, use 1 for read into and one for write into, leaving B-2 chinks of RAM left as space for rendezvous

2 SQL

The standard for database queries - has been around since the 70s and remains the top language

2.1 Pros and Cons

Pros: SQL is a declarative language and is widely implemented (varying levels of efficiency and completeness). It is also general-purpose and feature-rich (many years of added features, extensible: callouts to other languages, data sources)
Cons: Constained (Core SQL is not a Turing-complete language - extensions make it Turing complete)

2.2 Relation Terminology

- Database: set of relations
- Relation (Table): Schema (description, schema of database is set of schema of its relations), Instance (data satisfying the schema)
- Attribute (Column)
- Tuple (Record, Row)

- Schema is fixed, set up in the beginning. Attribute names, atomic types. Populated with data that changes over time
- Instance can change: a multi-set of rows. Tables are unordered and there can be repeats
- DDL: Data Definition Langue is where you declare hat you want your tables to be and look like.
- DML: Data Manipulation language is where you say what info you actually want in the table
- RDBMS is responsible for efficient evaluation

Foreign Key (sid) References Sailors: this means that the sid will be tied to the sid in the Sailors table - you can't put something in the new table that didn't exist in the other table

2.3 Single Relation Queries

DISTINCT means remove duplicates SELECT [DISTINCT] what you want
FROM table
WHERE condition
GROUP BY column list
HAVING predicate
ORDER BY column list

Aggregates compute a summary of some arithmetic expression and produce one row of output
Group by partitions the table into seperate areas and produce aggregate result per group - cardinality of output is the number of distinct group values
Having predicate is applied after grouping and aggregation - hence can contain anything that could go in the SELECT list - only used in aggregate queries

Order: FROM comes first - gives us the table. Then we look at the table data and stream it through the WHERE clause. In the SELECT clause we see which columns we actually care about. Then we do GROUP BY and create buckets and collect tuples for each bucket. Then we look at the HAVING clause and throw out any bucket that doesn't satisfy. Then we eliminate duplicates if DISTINCT is on and then we do that AGGREGATE

2.4 Querying Multiple Relations - Join Queries

FROM Sailors S, Reserves R - we have multiple tables in this clause
WHERE S.sid = R.sid - otherwise we get nonsense

Range Variabels: needed when ambiguity could arise - same table used multiple times in FROM

Where S.name LIKE 'P(underscor)p(percent sign)' means name starts with a P then some stuff then a lower case p
EXCEPT can remove sailors who have reserved a boat - removing some group - also eliminates duplicates

2.5 Nested Queries

WHERE S.sid IN
For each S.sid check if it is in the nested table (can also do NOT IN)

WHERE EXISTS (subtable) - subtable will include WHERE S.sid = R.sid

WHERE S.rating ¿ ANY - where the rating is bigger than some selected rating value in the subtable

2.6 NULL Values

Field values that are sometimes unknown or inapplicable - SQL provides a special value null for such situations
In a truth table - ? or True = True, ? and True = ?, etc. - these are unknown values

2.7 Inner Join and Outer Join

FROM Sailors S
INNER JOIN Reserves r
ON s.sid = r.sid
ON is called the join condition

FROM Sailors S
LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
We return all the matching rows, plus all unmatched rows from the table on the left of the join clause - use nulls in fields of non-matching tuples
Full Outer Join will use nulls to fill in blanks from both tables

2.8 Views: Named Queries

CREATE VIEW viewName
AS selectStatement
Views make development simpler, used for security, not "materialized" (you just store the select statement - you don't store the actual table)

GRANT privileges ON object TO users
Object can be a Table or a view, Privileges talk about operations allowed on

those table - Select, insert, delete, reference (create foreign key that references specified columns), all - can later be revoked

2.9 Subqeiers in FROM

Akin to a view
FROM Boads2 b,
(SELECT b.bid, COUNT(*)
FROM ... WHERE ...) AS Reds(Bid, scount)
Now you can refer to Reds later - this is a view created on the fly

2.10 Constrains

Integrity Constrains: conditions that every legal instance of a relation must satisfy (similar to assertions in other languages)
Inserts/deletes/updates that violate IC's are disallowed - can ensure application semantics (sid is a key) or prevent inconsistensies (sname must be a string)

Domain Constrians: field values must be of the right type (always enforced)

Primary Key and Foreign Key Constraints: keys are way to associate tuples in different relations - one form of ICn. A foreign key is when a key references a table where it isn't the primary key (sid primary key in students table but foreign key in classes table)
FOREIGN KEY (sid) REFERENCES Students - can use to prevent not inserting a student ID that doesn't actually correspond to a student
Cascading delete means that the delete propagates to the tables which refer to the original table
Superkey: no two distinct tuples can have the same value in all key fields
General Constaint: Check (rating between one AND ten) - they can be more complicated and actually be select from where queries

3 Disks and Files

3.1 Database Management System (DBMS) Architecture

First you do query parsing and check validity. Then optimization turns this query into something that can actually be handled by the system. This takes us to a relational operators which access something lower down - the files and index management. The layer belwo this is the buffer management which plays the role of essentially virtual memory. Below this level is the disk/flash drive - each layer abstracts the layer below, allowing us to manage complexity and make performance instructions

3.2 Disk Space and Buffer Management

Going back and forth between disk and ram: program operates on data that is actually in memory - there is some interface with the disk that puts this world together and this is the buffer pool or buffer manager - manages a relatively small amount of pages of memory that deal directly with disk.

How are tables stored on disk? - usually stored as a logical file of some sort that can be broken up into pages, where each page contains a collection of records.
Pages are managed in memory, by the buffer manager - higher levels of database only operate in memory and on disk by the disk space manager which reads and writes pages to physical disk/files

3.3 Storage Media

A lot of databases still use magnetic disks - a mechanical anachronism - API exists to actually deal with physical memory when you read and write - these API calls are expensive
Economics: RAM is much more expensive than Magnetic disk, which is also more expensive than solid state disk
Storage Hierarchy: Registers, on-chip cache (very fast, small), on-board cache (slower, larger), RAM (currently used data), SSD (varies by deployment - sometimes database/cache) , Disk (database and backups/logs)

3.4 Components of a Disk

Platters spin (maybe 15000 rpm) - Arm assembly moved in or out to position a head on a desired track - each platter made up of tracks which are concentric rings
Only one head reads/writes at a time - block/page size is a multiple of fixed sector size

Key to lower cost is reducing seek and rotational delays
Seek time: how long it takes to ove the arm to position disk head on track
Rotational delay: how long it takes for the block to rotate under head
Transfer time: actually moving data to/from disk surface

3.5 Flash (SSD)

The cost of flash has come down dramatically recently - the current generation of flash devices have differences between reads and writes (small reads, big writes) - flash also gets worn out over time - tries to wear level and spread things out so one area isn't overused
Write amplication: big units, so to even write something small you still have to write a lot

Random reads are fast and predictable with no seek penalty - sequential reads also perform similarly (unlike hard drive). But writes are slower for random than sequential writes.

3.6 Storage Pragmatics and Trends

Many significant DBs are not that big - but data sizes grow faster than Moore's Law - minimize seek and rotational delay because we are faced with hard disks. The role of flash and RAM has increased recently

4 File Storage

R = the number of records on a page
B = the number of data blocks/pages
D = average time to read/write disk block

4.1 Sorted Files

Files are sorted according to search key
Scan All Records: BD
Find Specific Key: $\log_2(B)$ in worst case and avg case
Find keys between two values (in a range): $(\log_2(B) + range_len)D$
Insert: $(\log_2(B) + B)D$
Delete: $(\log_2(B) + B)D$

4.2 Heap Files

Inserts always append to the end
Scan All Records: BD
Find Specific Key: $\frac{BD}{2}$
Find keys between two values (in a range): BD
Insert: 2D
Delete: $(\frac{B}{2} + 1)D$

5 Indexes

5.1 Indexes Overview

An index is a data structure that enables fast lookup of data entries by search key
Search key: any subset of columns in the relation - doesn't have to be a key of the relation. **Composite Key** is more than one column
Data Entries: items stored in the index - can be actual data or pointers

How is data stored in the file?

- By value: actual data record with key value k
- By reference: < k, rid of matching data record >
- By list of references: < k, list of rids matching data records >

Clustered: index data entries stored in approximately order by value of search keys in data records - a file can be clustered on at most one search key - more expensive to maintain/update
The Fan out is F - relatively large (like 1000) in practice F is 2 or 3
Scan All Records: 1.5 * (BD)
Find Specific Key: $(\log_F(1.5 * B) + 1)D$ in worst case and avg case
Find keys between two values (in a range): $(\log_F(1.5 * B) + range_len)D$
Insert: $(\log_F(1.5 * B) + 1)D$
Delete: $(\log_F(1.5 * B) + 1)D$

5.2 Page Basic

Page header - number of records, free space
Record length can be fixed or variable - if they are fixed you can pack records densely by appending each insert. If you delete in the middle - use bitmap and mark the deleted whole in the map. For variable length record - slot directory with the length and pointer of each record and a pointer that denotes free space

5.3 Record Formats

You can move the variable length fields to the end and use a record header

5.4 High fan-out search tree - ISAM

Binary search key file - only look at the first key in the page along with pointers to the page - if you need to break this up across pages do the same thing recursively and build another layer on top until there is only one page
This means lookup is $\log_F$ - insert is also $\log_F$ for non overflow inserts and keeps a linked list for overflow pages
Lots of insertions lead to more overflow pages which slows down ISAM
Pros: sequentially means you can scan all records without touching index = good for static data with big scans
Cons: doesn't handle insertion/deletions well and degrades to linear search

5.5 B+ Trees

There are pointers both ways at the leaf level from the greatest in entry i to the smallest in entry i + 1.
Occupancy invariant: every node is at least half full (exception of root) - d is order of tree - each node has at least d entries and can hold max 2d entries - max fanout is 2d + 1 (+1 because 1 more pointer then entry possible) - not

held for the right most branch in a B+ tree
Data pages at bottom no longer in logical order - a tree can store $F^h * 2d$ records total **Inserts**: either simply insert into appropriate leaf in log time - or it's full you split the leaf L in half and insert the new entry where it goes - you then copy the middle key along with a pointer to new page and put it in the level above
Delete: start at root - find leaf - delete - if it has d-1 (less than fill factor) then you re-distribute from sibling (adjacent node with same parent as L) - if this fails merge L and sibling
Fill factor how full each node is - you want it higher for fewer insertions
Variable length keys and records:we must redefine occupancy invariant - the page is half full measured in bytes

5.6 Prefix Compress Keys

: store the prefix to distinguish them from the next key - doesn't quite work so we must change to compressing starting from the leaf

5.7 Suffix Key Compression

: all keys have common prefix and we make this prefix the header and use the suffixes in the actual page

6 Buffer Management

Data must be in RAM for DBMS to work on it and the buffer manager makes sure this is a similar process
Pinning: prevent buffer manager from evicting page by pinning it if it's being currently used

6.1 When a Page is requested

The requester indicates whether page was modified via dirty bit and also must unpin the page when done using - different requestors can request the same page so a **pin count** is used

6.2 Page Replacement Policy

LRU, Clock, MRU - right policy depends on access patterns

- Least Recently Used (LRU): keep track of last time frame was unpinned and replace the least recently unpinned - this can be costly because you must maintain heap data structure - very common, intuitive, simple - bad with repeated scans of big files (**sequential flooding**)

- Clock Replacement Policy: keep frames in a logical cycle and have a reference bit that says and clock arm advances until we find the page to evict by skipping pinned pages or if reference bit is set - when you get a cache hit or insert a page you set its reference bit
- Most Recently Used (MRU): reverse of LRU - good for sequential scans

Prefetching: in the background prefetching the next needed pages in a sequential scan
Double Buffering main thread funs on one pair of I/O buffs and 2nd I/O thread fills/drains unused I/O bufs. When the main thread is ready for a new buf, swap.

7 Sorting

helps eliminate duplicates and summarize groups of items - additionally output sometimes must be ordered
Given a file F, containing multiset of records R consuming N blocks of storage. We have two scratch disks (plenty of space), but we have a fixed amount of RAM equivalent to B blocks of disk (where $B << N$)
We want to produce to files S and H where S is sorted and H is arranged so that no two records are

7.1 External Sorting: 2 way - a strawman

on pass 0 we read the page, sort it, and write - each page in sort order. In subsequent passes we merge the files 2 at a time recursively - the number of passes is $ceiling(log_2(N)) + 1$ - each pass we read and write each page in the file 2N since there are N pages in the file, so the total IO cost is $2Nceiling(log_2(N)) + 1$
If we have B blocks, we can use B-1 blocks in the merging process - $\frac{N}{B}$ sorted runs in pass 0, which means total cost is $2N(1 + log_{B-1}\frac{N}{B})$

7.2 Internal Sort

tournament sort/ heap sort - we keep two heaps in memory H1 and H2.
We read B-2 pages or rages, inserting into H1 - then we iterate over the heap until it is empty taking out m from H1 - we read in another record r and if $r < m$ we insert r into H1 else into H2
Average length of a run is 2(B-2)

7.3 Hashing

We first do a streaming partition using a hash function h_p to stream records to disk partitions

Two Phases: first partition (divide) - then rehash by reading partitions into RAM hash table one at a time, using a different hash function h_r to conquer
How big of a table can we hash in two passes- B-1 partitions from pass 1 - each should be no more than B pages in size - so B(B-1) total.

8 Relational Algebra

Uses relation calculus - $\in$, $\wedge$, etc. - there is exact equivalence between relational calculus and relational algebra
Relational algebra has operators that take relations and produce more relations (**closed** - you can impose them together as the output is another relation) and **typed**, meaning input schema determines output

Aditionally it has set semantics, which means no duplicate tuples

8.1 Uniary Operators

Takes a single relation instance as input and produce one as output

- Projection (π): select a subset of columns (vertical) - corresponds to select clause

$$\pi_{\text{sname, rating}}(S2)$$

- Selection (σ): select a subset of rows (horizontal) - corresponds to where clause

$$\sigma_{\text{rating}>8}(S2)$$

- Renaming (ρ): give a name to results or output of queries, returns of selection statements, and views of queries that we would like to view at some other point in time

$$\rho(Temp1, 1 \rightarrow sid1, 4 \rightarrow sid2, R1xS1)$$

  The first argument is the output relation name and the last is the input relation. The middle argument renames individual columns

8.2 Binary Operators

Takes pairs of relation instances as input and produce one as output

- Union ($\cup$): takes the union of two set - they must be compatible - corresponds to union clause
- Intersection ($\cap$): takes intersection - corresponds to intersect
- Set difference (-): remove things from S2 from S1 if S1-S2 - corresponds to except

- Cross Product (x): each row of R1 paired with each row of S1 (R1 x S1) size is the size multiplied
- Joins ():

Monotone: increasing the amount of inputs can only increase or keep the same the amount of outputs - union, intersect, etc. are but set difference is not