CS 61:
Database Systems

Data persistence, file organization, indexing

Big picture: find a needle in a big data haystack quickly

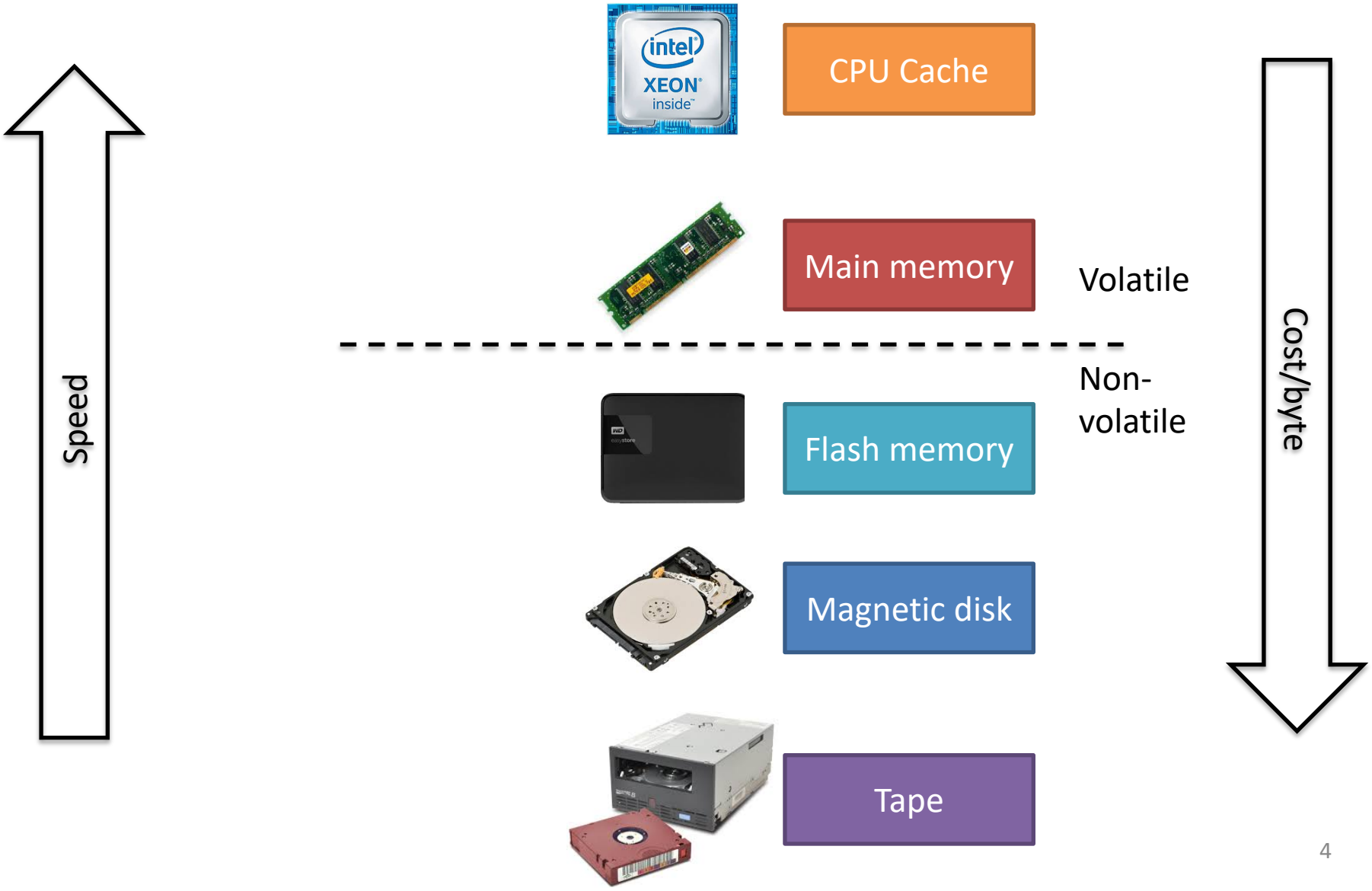
Find data on Morris Park Bake Shop in large database quickly…

But Morris Park Bake Shop is just one entry in large data set

Agenda

1. Data persistence
2. Database file organization
3. Indexing

There are many different types of data storage used by databases



CPU cache is fast, but small and volatile

CPU cache

- Fast but expensive
- Holds small amount of data (normally megabytes of recently used data)
- Volatile (loose if power fails)

CPU Cache

Main memory

Volatile

Non-volatile

Flash memory

Magnetic disk

Tape

Main memory is larger, but still small and volatile

Main memory

- Larger than CPU cache (gigabytes)
- Volatile (loose if power fails)
- Some databases can be entirely contained in memory
- Use CS10 data structures to access data quickly if stored in memory

CPU Cache

Main memory

Volatile

Non-volatile

Flash memory

Magnetic disk

Tape

Flash (SSDs) are larger and non-volatile, but much slower than main memory

Flash memory

- Solid State Drives (SSDs)
- Often larger than main memory (up to terabytes)
- Non-volatile (do *not* loose if power fails)
- Faster data access than magnetic disks
- Read data in blocks (pages) of about 4 KB

CPU Cache

Main memory

Volatile

Non-volatile

Flash memory

Magnetic disk

Tape

Magnetic disk have been the mainstay of data storage for decades

Magnetic disk

- Mainstay of data storage
- Large (up to dozens of terabytes)
- Made up of (perhaps many) spinning platters
- Slower than SSDs
  - Seek track/ sector
  - Rotational latency
- Read data in blocks (page) of roughly 8 KB

CPU Cache

Main memory

Volatile

Non-volatile

Flash memory

Magnetic disk

Tape

Tape is large, but slow; mainly used for back ups

Tape

- Very large capacity (terabytes)
- Normally used for off-line backups
- Do you need back up if replicate database?
- YES!!!!
- A rouge process that writes garbage writes it to all replicas!

CPU Cache

Main memory

Volatile

Non-volatile

Flash memory

Magnetic disk

Tape

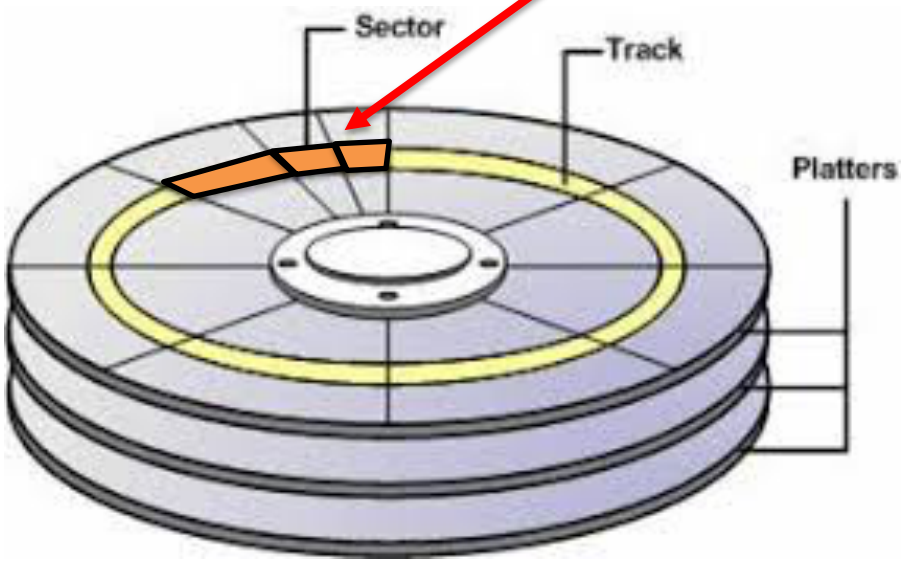
Most online data is stored on a magnetic Hard Drive (HD) or Solid-State Disk (SSD)

Data storage on magnetic disk

Time to read/write:

- **Seek time to track**
- **Rotational delay for sector**
- **Transfer speed**
- **Controller overhead**

1 disk block
4KB in size



RAID – Redundant Array of Independent Disks stores data across multiple disks

- Striping to increase throughput (RAID 0)
- Mirroring to reduce failures (RAID 1)

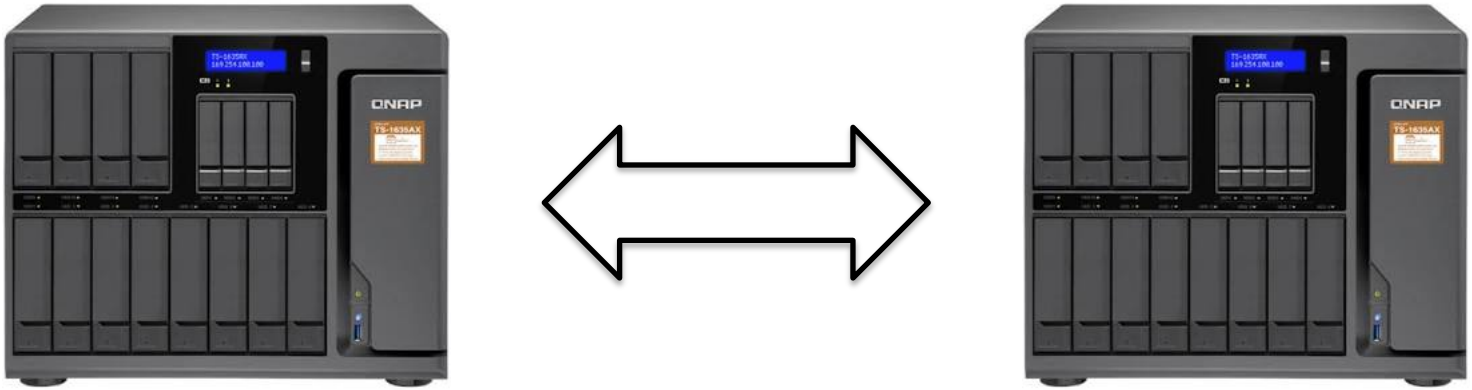
Hard drive read/write

- A hard disk often has multiple spinning *platters*, each with a read/write head
- Each platter has several concentric *tracks*
- Each track is divided into multiple *sectors*
- The read/write head can address data on at a track/sector location
- To read or write, move arm to correct track, and wait for sector to spin underneath head
- Disk itself cannot address smaller amounts of data than one sector (normally 512 bytes)

Operating system addresses blocks of data

- A block spans several sectors
- OS cannot address smaller than block
- Normally around 4 KB today (8 sectors)
- Files written across multiple blocks

Disks are often grouped into Storage Area Networks (SANs)



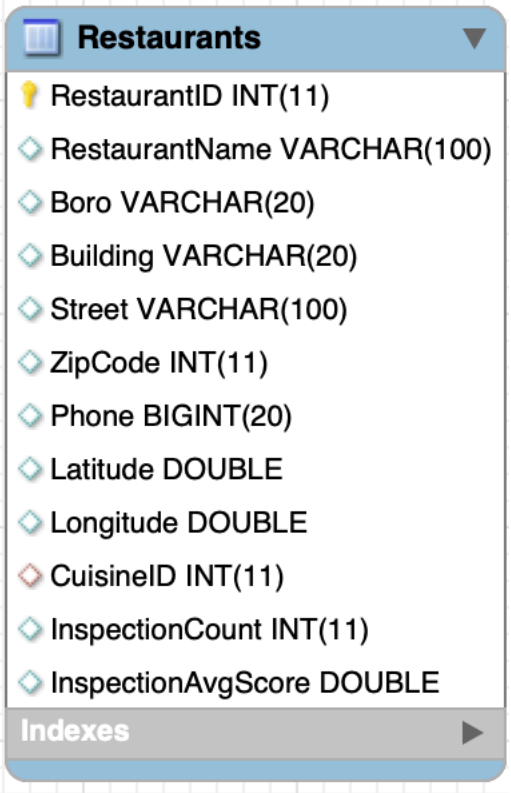
Storage Area Network (SAN)

- Provides block-addressable high-performance non-volatile storage
- Often connected to other SANs
- SAN software replicates data across SAN devices to eliminate single point of failure (e.g., fire in data center)
- Not JBOD, often use striping and mirroring within one SAN

Agenda

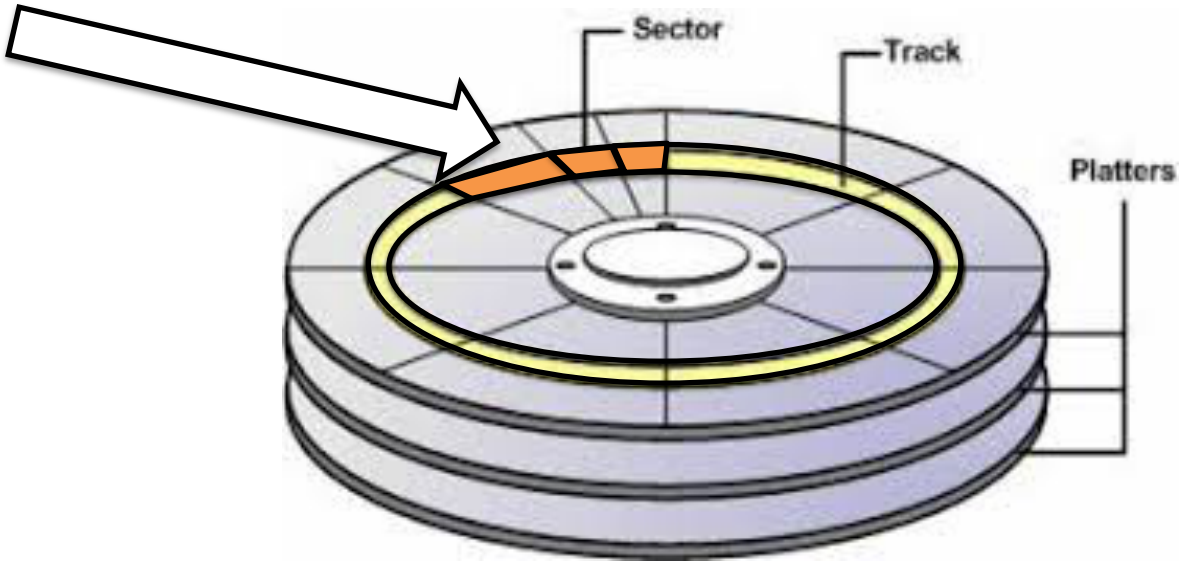
1. Data persistence
2. Database file organization
3. Indexing

Databases persist data onto disk in files that normally span multiple blocks



Two approaches

- **Fixed length rows**
- **Variable length rows**



- Each relation generally stored in one file
- Each file is a sequence of table rows made up of attributes mapped onto disk blocks roughly 4KB in size
- One row assumed to be smaller than block size (book gives solution if not)

Fixed length rows are easy to implement, but can waste disk space

Simplified fixed length Restaurant records

ID Int	Name Varchar(100)	Boro Varchar(20)	AvgGrade Double
30075445	Morris Park Bake Shop	Bronx	10.6
30075445	Wendy's	Brooklyn	19.8
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8

4 byte integers

100 bytes for varchar

Other bytes not used if Restaurant name < 100 characters long

20 bytes for varchar

Other bytes not used if Boro name < 20 characters long

8 bytes double

Row *size* = 132 bytes, find record *i* at *file start + i*size* bytes

Most likely some bytes wasted (Restaurant name is probably not 100 characters long)

Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Deleting rows can be tricky, could copy records to fill hold, but inefficient!

Simplified fixed length Restaurant records

ID Int	Name Varchar(100)	Boro Varchar(20)	AvgGrade Double
30075445	Morris Park Bake Shop	Bronx	10.6
Deleted			
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
40356018	Riviera Caterers	Brooklyn	11.1

Could fill gap but might involve many copies if record near start of file and many entries This would be slow!!!

Row *size* = 132 bytes, find record *i* at *file start + i*size* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

A better way to handle deletes is to keep a list of free spaces

Simplified fixed length Restaurant records

ID Int	Name Varchar(100)	Boro Varchar(20)	AvgGrade Double
30075445	Morris Park Bake Shop	Bronx	10.6
Deleted			
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
Deleted			

Normally more inserts than deletes
Keep list of free spaces, insert record into space on free list on next insert

Row *size* = 132 bytes, find record *i* at *file start + i*size* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Inserting new row, add onto end if no entries in free list

Simplified fixed length Restaurant records

ID Int	Name Varchar(100)	Boro Varchar(20)	AvgGrade Double
30075445	Morris Park Bake Shop	Bronx	10.6
30075445	Wendy's	Brooklyn	19.8
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
40356018	Riviera Caterers	Brooklyn	11.1

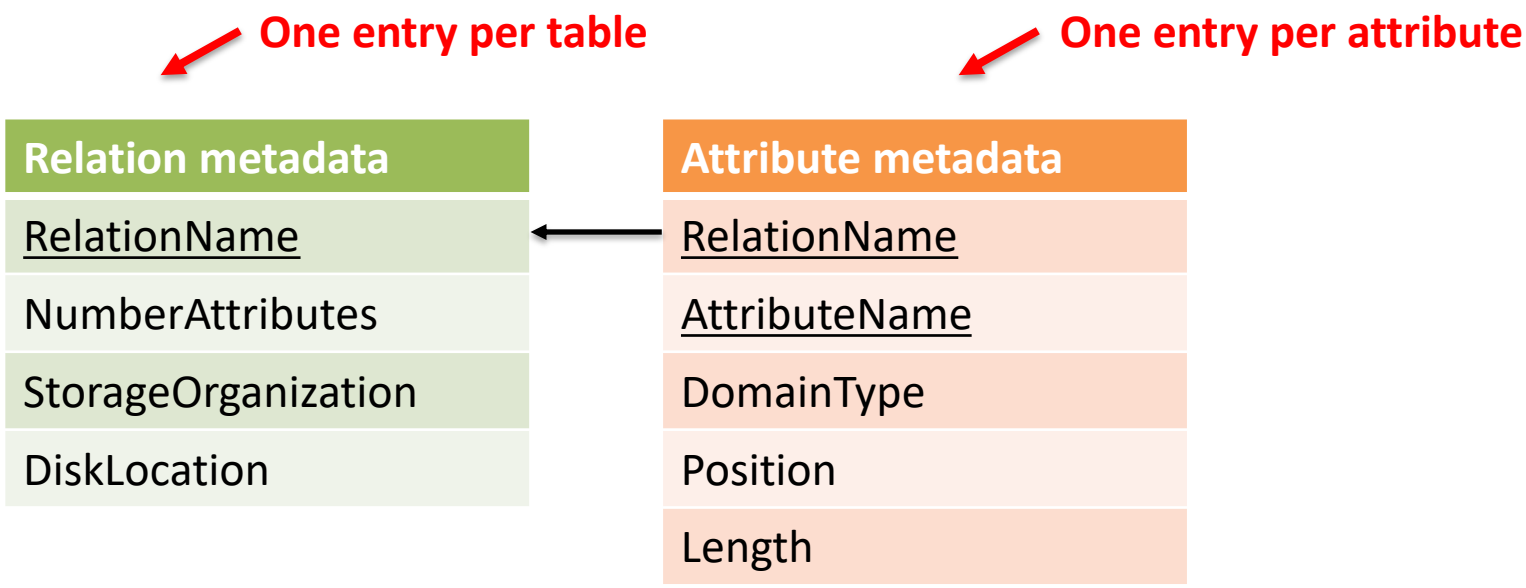
Insert new record into free space or add to end of file

Row *size* = 132 bytes, find record *i* at *file start + i*size* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Store *block size/row size* records in each block, do not use the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Database keeps track of file layouts in data dictionary (system catalog)

Data dictionary



Data dictionary tracks relations and attributes

Relation metadata table has entry for each table

Attribute metadata uses RelationName in PK and has entry for each attribute

- Lists domain type for each attribute (e.g., INT, VARCHAR, DOUBLE)
- Position in record layout on disk
- Length of attribute

Variable length records are more complicated to implement but save space

Variable length records

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 42	43 47
0000	30075445	24, 19	43, 5	10.6	Morris Park Bake Shop	Bronx

To track NULLs, records have a byte array where each bit represents one attribute

- Set bit to 1 to indicate the value is NULL
- e.g., Boro is in 3rd position, set 3rd bit to 1 if Boro is NULL
- I did not show this field in the fixed length record, but often used there too

Some domain types are always of fixed length such as INT and DOUBLE

Variable length records

Fixed length 4 bytes

Fixed length 8 bytes

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 42	43 47
0000	30075445	24, 19	43, 5	10.6	Morris Park Bake Shop	Bronx

Look up attribute types in data dictionary, get order and length for each attribute

Attribute metadata
RelationName
AttributeName
DomainType
Position
MaxLength

Relation Name	Attribute Name	Domain Type	Position	Max Length
T_1	ID	INT	1	4
T_1	Name	VARCHAR	2	100
T_1	Boro	VARCHAR	3	20
T_1	AvgScore	DOUBLE	4	8

Use start and length pointers to track variable length attributes such as VARCHAR

Variable length records

Variable length have start and length integers
Start and end are fixed length, 2 bytes each

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 42	43 47
0000	30075445	24, 19	43, 5	10.6	Morris Park Bake Shop	Bronx

Look up attribute types in data dictionary, get order and length for each attribute

Attribute metadata
RelationName
AttributeName
DomainType
Position
MaxLength

Relation Name	Attribute Name	Domain Type	Position	Max Length
T_1	ID	INT	1	4
T_1	**Name**	**VARCHAR**	**2**	**100**
T_1	**Boro**	**VARCHAR**	**3**	**20**
T_1	AvgScore	DOUBLE	4	8

Name starts at byte 24 and is 19 characters long
Boro starts at byte 43 and is 5 characters long

You can see this data by querying the INFROMATION_SCHEMA table

Variable length records

MySQL tracks some other attributes in addition to our somewhat simplified model

```
6 •   SELECT * FROM INFORMATION_SCHEMA.COLUMNS
7     WHERE TABLE_SCHEMA = 'nyc_inspections' AND TABLE_NAME = 'Restaurants';
```

100% 1:2

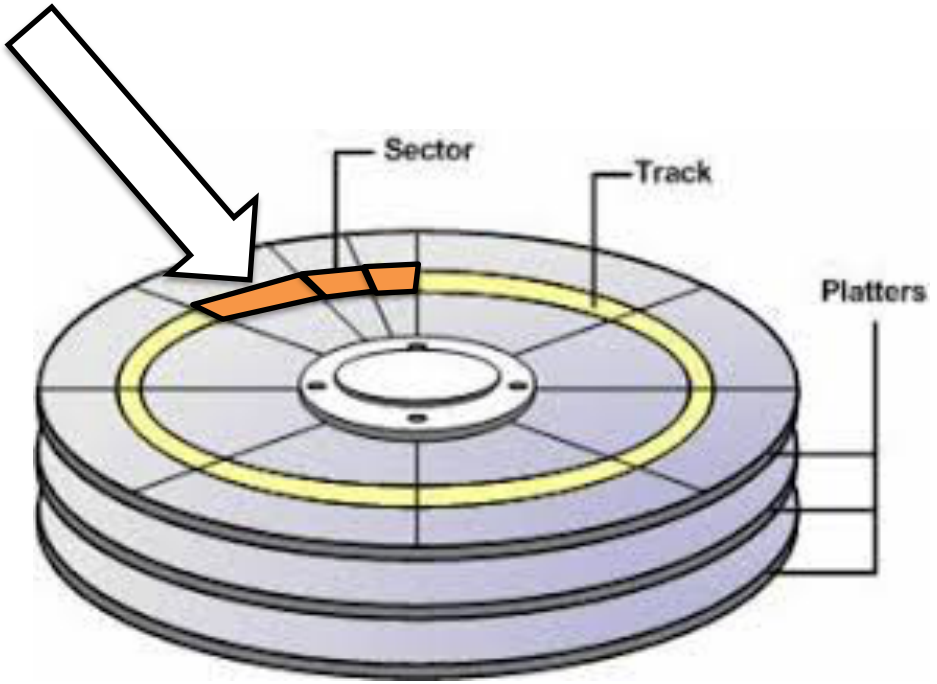
Result Grid Filter Rows: Search Export:

T...	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_M
def	nyc_inspections	Restaurants	RestaurantID	1	NULL	NO	int	NULL
def	nyc_inspections	Restaurants	RestaurantName	2	NULL	YES	varchar	100
def	nyc_inspections	Restaurants	Boro	3	NULL	YES	varchar	20
def	nyc_inspections	Restaurants	Building	4	NULL	YES	varchar	20
def	nyc_inspections	Restaurants	Street	5	NULL	YES	varchar	100
def	nyc_inspections	Restaurants	ZipCode	6	NULL	YES	int	NULL
def	nyc_inspections	Restaurants	Phone	7	NULL	YES	bigint	NULL
def	nyc_inspections	Restaurants	Latitude	8	NULL	YES	double	NULL
def	nyc_inspections	Restaurants	Longitude	9	NULL	YES	double	NULL
def	nyc_inspections	Restaurants	CuisineID	10	NULL	NO	int	NULL
def	nyc_inspections	Restaurants	InspectionCount	11	NULL	YES	int	NULL
def	nyc_inspections	Restaurants	InspectionAvgS...	12	NULL	YES	double	NULL

Multiple records are typically stored in one disk block

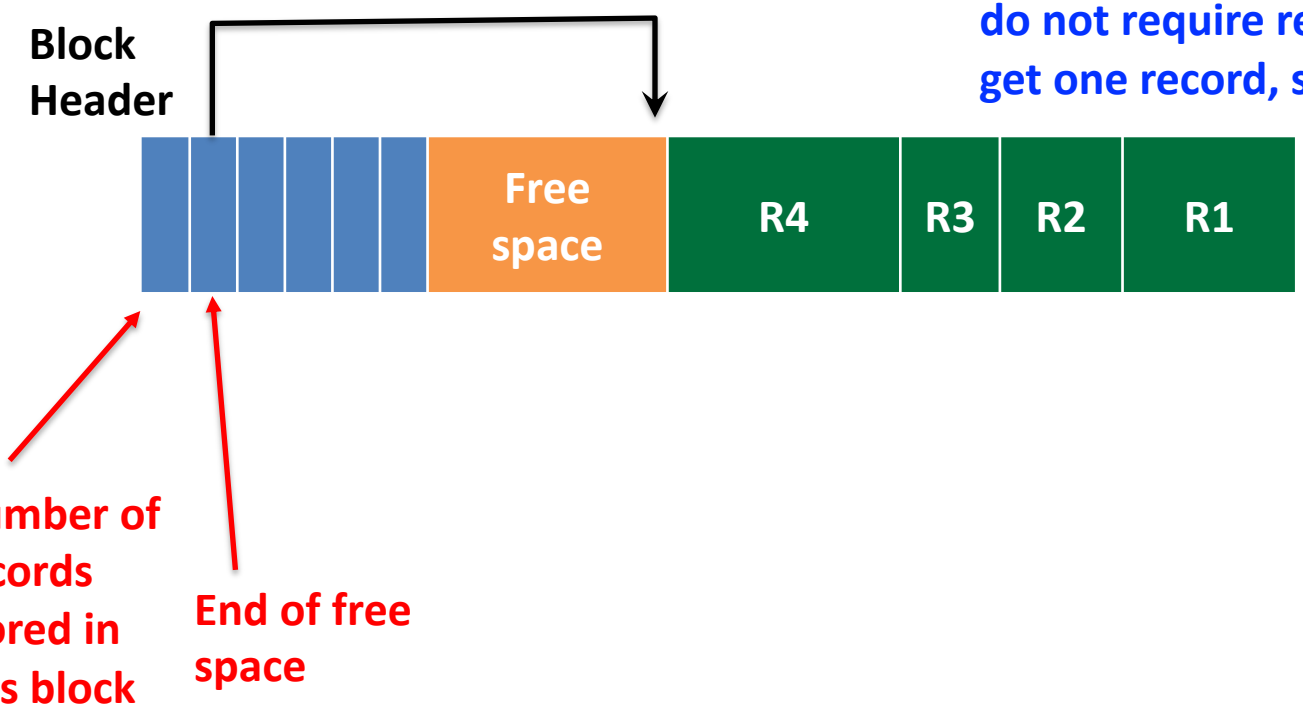
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30075445	MORRIS PARK BAKE SHOP	Bronx	1007	MORRIS PARK AVE	10462	7188924968
30112340	WENDY'S	Brooklyn	469	FLATBUSH AVENUE	11225	7182875005
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40356018	RIVIERA CATERERS	Brooklyn	2780	STILLWELL AVENUE	11224	7183723031
40356151	BRUNOS ON THE BOULEVARD	Queens	8825	ASTORIA BOULEVARD	11369	7183350505
40356483	WILKEN'S FINE FOOD	Brooklyn	7114	AVENUE U	11234	7184443838
40356731	TASTE THE TROPICS ICE CREAM	Brooklyn	1839	NOSTRAND AVENUE	11226	7188560821
40357217	WILD ASIA	Bronx	2300	SOUTHERN BOULEVA…	10460	7182207846

If disk block is 4KB and each row is roughly 500 bytes, then there are around 8 records (4,000/500) per disk block



Data stored in blocks on disk so that records do not span multiple blocks

Disk block organization



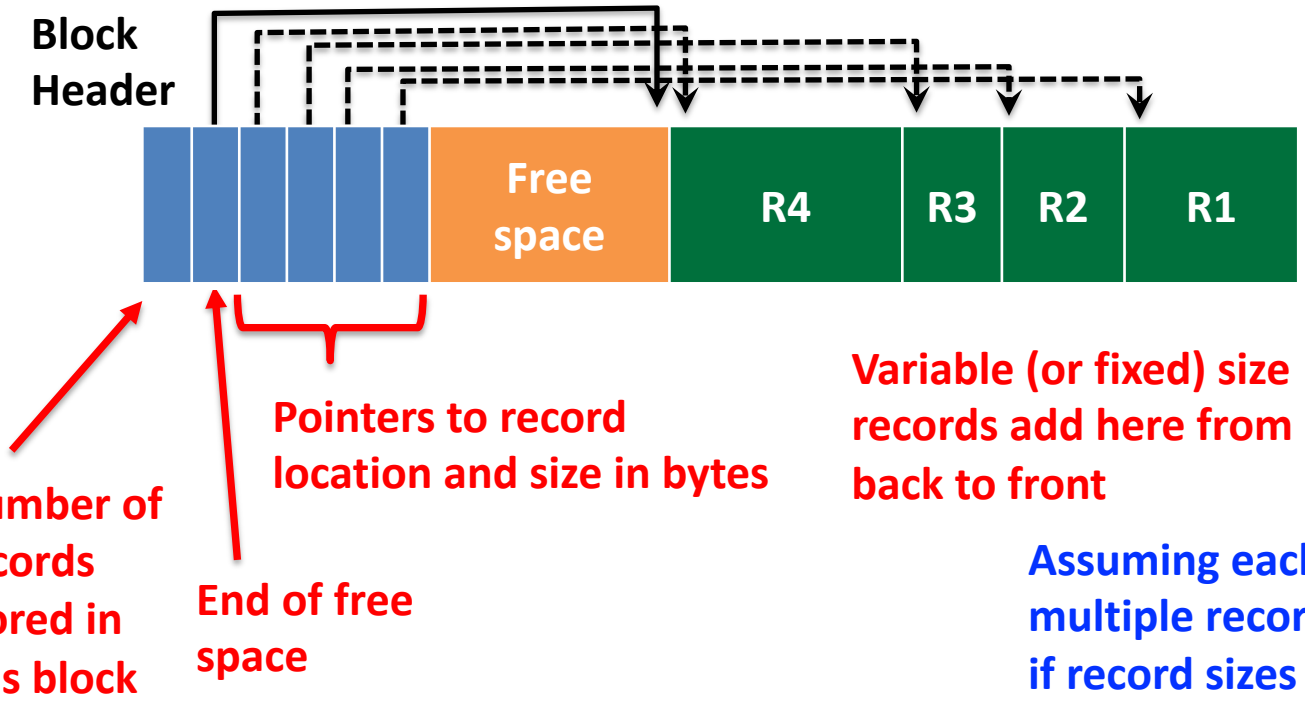
Remember reading from disk is slow, do not require reading two blocks to get one record, so do not span blocks!

Storing records in a disk block

- Header gives number of records stored in block
- Records added from back to front
- Free space in between header and records
- New records added to end of free space

Data stored in blocks on disk so that records do not span multiple blocks

Disk block organization



Storing records in a disk block

- Header gives number of records stored in block
- Records added from back to front
- Free space in between header and records
- New records added to end of free space

Agenda

1. Data persistence
2. Database file organization
3. Indexing

Without indices, database must do a full table scan to find rows meeting criteria

```
1 • use nyc_inspections;
2 • SELECT * FROM Restaurants
3   WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Must look over entire table for restaurants in Manhattan
Slow if table is large!
Would prefer fast way to find by boro

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR...	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

Notice results are sorted by RestaurantID, why?
MySQL automatically creates clustered index for PK (concatenate for composite key PKs)
Clustered index means rows are sorted on disk by index key, secondary indices are not
MySQL uses index to return rows

Each table has at least one index; normally based on primary key

```
1 • use nyc_inspections;
2 • SELECT * FROM Restaurants
3   WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Must look over entire table for restaurants in Manhattan
Slow if table is large!
Would prefer fast way to find by boro

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR...	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

There can be only one (and only one) clustered index per table
All tables have at least one index
If a primary key is not declared, MySQL uses the first UNIQUE index
If no UNIQUE index, MySQL creates a synthetic column with RowID

Add new rows to table, update index with location of each new row

```
1  use nyc_inspections;
2  SELECT * FROM Restaurants
3  WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Must look over entire table for restaurants in Manhattan
Slow if table is large!
Would prefer fast way to find by boro

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

If new rows are inserted into table:

- **Add at space indicated by free space list or end of file**
- **Update all indices to show where new row is located (indices cause increased overhead)**
- **Database will move records to keep clustered index sorted by index when not busy**

Add deleted rows to free list and update index for each row removed

```
1  use nyc_inspections;
2  SELECT * FROM Restaurants
3  WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Must look over entire table for restaurants in Manhattan
Slow if table is large!
Would prefer fast way to find by boro

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

If rows are deleted:

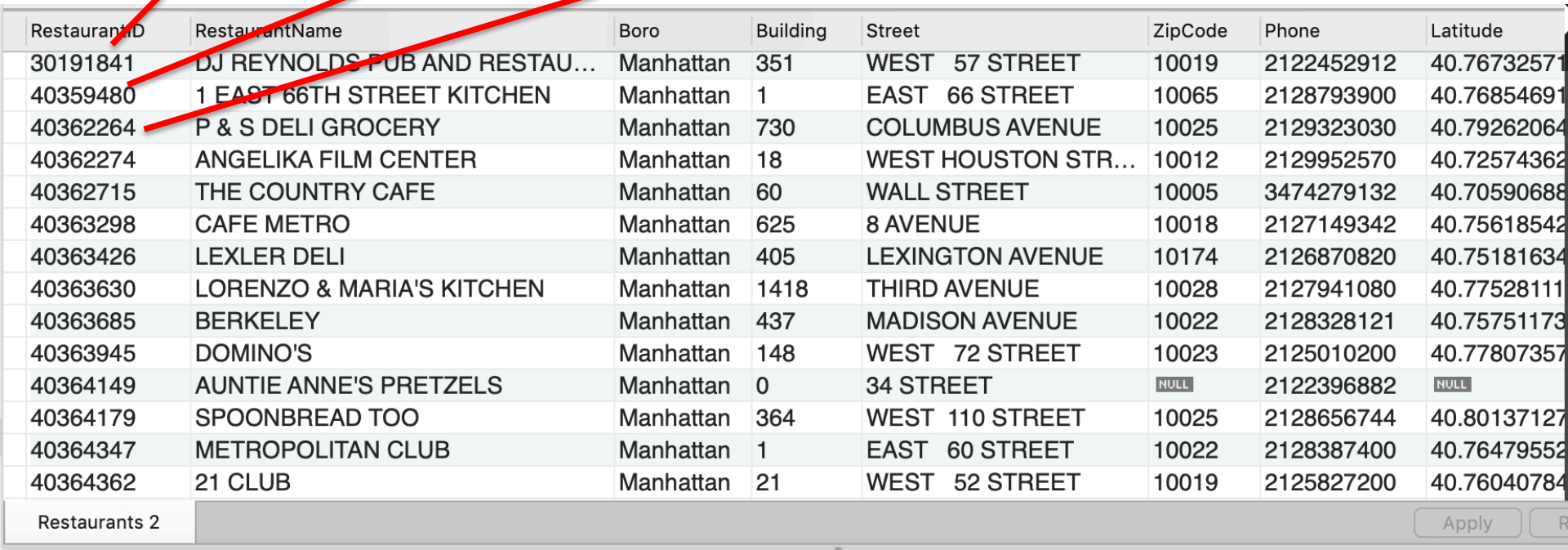
- **Add to row to free space list**
- **Update all indices by removing entry (indices cause increased overhead)**
- **Database will move records to keep clustered index sorted by index when not busy**

B+ trees normally used for indices, work like 2-3-4 trees from CS10

Creating B+ tree index

On insert add search keys to node until node is full
Non-leaf nodes (except root) have between $\lceil m/2 \rceil$ *and m* children (4 here)
Each node has pointer to where row is located on disk (not shown)
When a node is full, split and promote

3019141	40359480	40362264



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Restaurants 2

Apply

B+ trees normally used for indices, work like 2-3-4 trees from CS10

Creating B+ tree index

On insert add search keys to node until node is full

Non-leaf nodes (except root) have between $\lceil m/2 \rceil$ *and m* children (4 here)

Each node has pointer to where row is located on disk (not shown)

When a node is full, split and promote

3019141	40359480	40362264

Full, split and promote

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

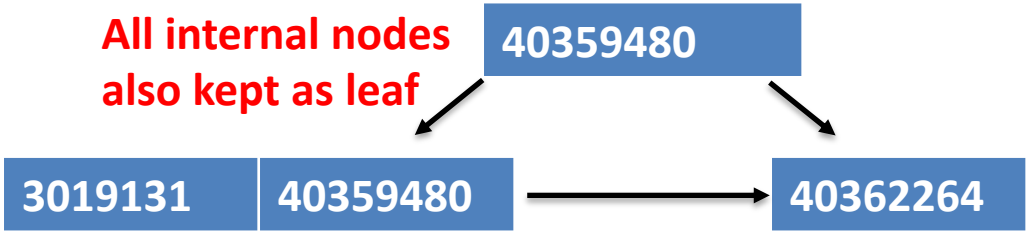
Restaurants 2

Apply

B+ tree works like 2-3-4 trees from CS10 (2-3-4 is special case of B+ tree)

Creating B+ tree index

Insert new node by traversing from root



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

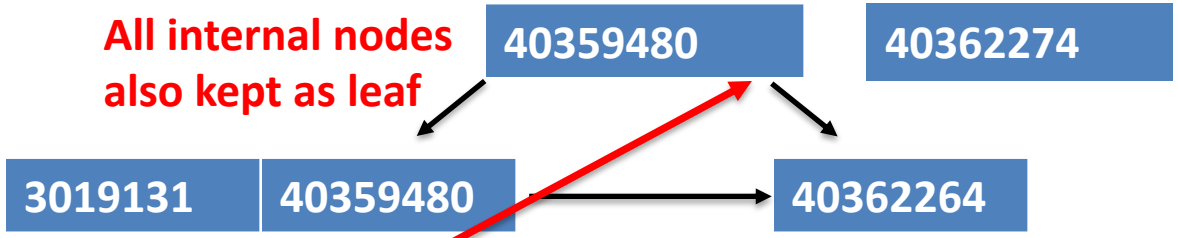
Restaurants 2

Apply

B+ tree works like 2-3-4 trees from CS10 (2-3-4 is special case of B+ tree)

Creating B+ tree index

Insert new node by traversing from root



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

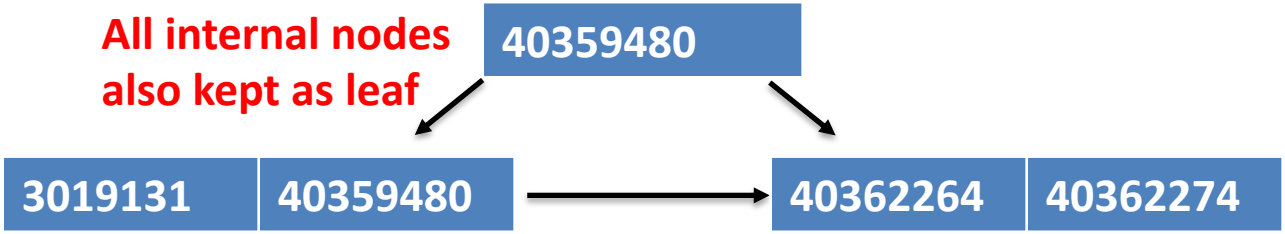
Restaurants 2

Apply

B+ tree works like 2-3-4 trees from CS10 (2-3-4 is special case of B+ tree)

Creating B+ tree index

Insert new node by traversing from root



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Restaurants 2

Apply

B+ tree works like 2-3-4 trees from CS10 (2-3-4 is special case of B+ tree)

Creating B+ tree index

Insert new node by traversing from root

Unlike 2-3-4 trees, B+ tree leaves keep pointer to next leaf

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Restaurants 2

Apply

The height of a B+ tree is normally small, only a few levels

Creating B+ tree index

In practice:

- **m is normally 100 or more**
- **Height is usually around 3 or 4, even for large tables**

$\log_{\lceil m/2 \rceil} n \leq \text{height} \leq \log_m n$

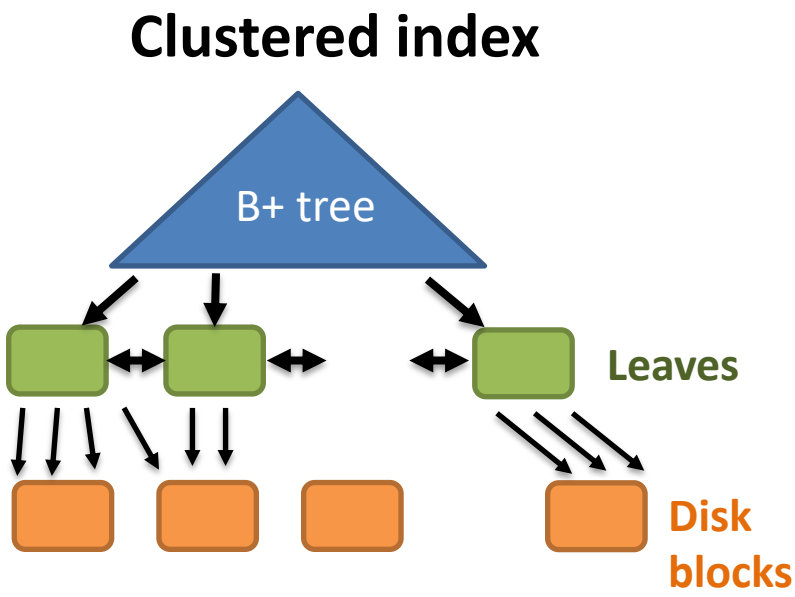
Where:
m = children/node
n = nodes in B+ tree

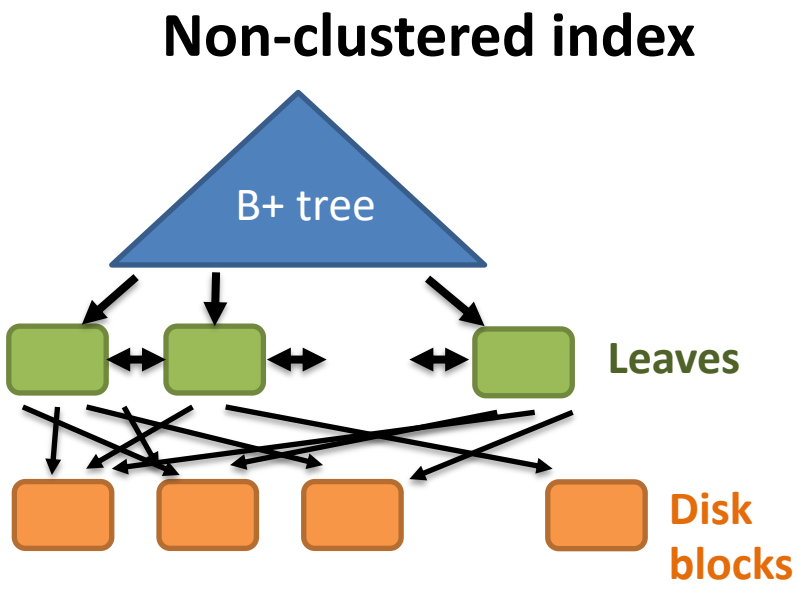
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Restaurants 2

Apply

Clustered indices make range queries fast, non-clustered mean more reads





Range query: SELECT * **FROM** T_1 **WHERE** col >= lb **AND** col <= ub

Fast on clustered indices

- Traverse to find lower bound at leaf
- Read many rows in one disk block read
- Traverse right until find upper bound

Require more disk reads on non-clustered (secondary) indices

Create indices on tables using the CREATE INDEX command

CREATE INDEX idx_boro **ON** Restaurants(Boro);

- If index is secondary index (non-clustered), index must be dense because rows are sorted by clustered index
- Now have pointers to rows with all unique values (for each Boro here)
- Result is that we can find all restaurants in each Boro (say Queens) without scanning the entire table
- Cardinality of the index is the number of unique items (6 here)
- MySQL creates a separate BTREE for each index on table
- Index downsides:
  - More storage space (not practical to put index on each attribute)
  - Must keep indices up to date on insert, update, delete operations
  - Database reorders clustered indices when not busy