Query Optimization

CS 327E
Feb 26, 2018

Announcements

- Midterm next class in **PAI 2.48** instead of our regular classroom.
- Review session Thursday at 5pm in WAG 420.
- After Spring Break: guest lecture, BigQuery.

1) What is the **key** benefit of having index structures in a database?

A) They speed up read queries
B) They compress column data
C) They improve write throughput
D) They make the database resilient to crashes

2) Which of the following are tradeoffs associated with indexes?

A) Slower updates.
B) Slower inserts.
C) Slower deletes.
D) All of the above.

3) What is the SQL command for creating an index?

A) `CREATE INDEX table_name (column_name);`
B) `CREATE INDEX index_name ON table_name (column_name);`
C) `CREATE B-TREE index_name ON column_name;`
D) None of the above

4) A B-tree can index only a single column of a table.

A) True
B) False

5) Which column(s) of a table does the database engine automatically index?

A) Integer columns
B) Varchar columns
C) Primary key columns
D) All of the above

Simplest Database System

```
1  import os
2
3  def set(key, val):
4      db_file = open("/usr/local/database", "a")
5      db_file.write(key + "," val)
6      db_file.close()
7
8  def get(search_key):
9      db_file = open("/usr/local/database", "r")
10     for line in reversed(db_file):
11         key, val = line.split(",")
12         if key == search_key:
13             break
14     db_file.close()
15     return key, val
```

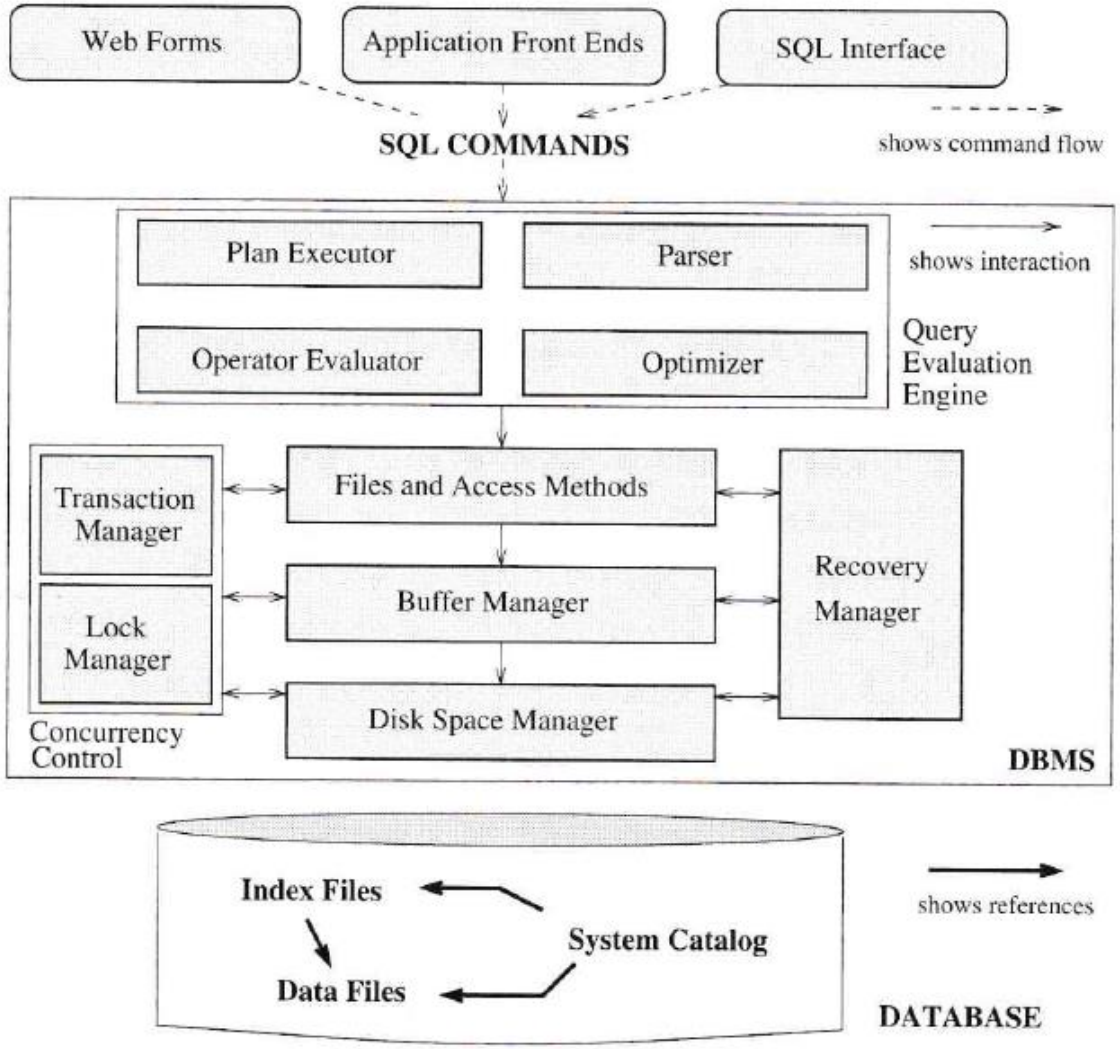
Simplest Database System

```python
import os

def set(key, val):
    db_file = open("/usr/local/database", "a")
    db_file.write(key + "," val)
    db_file.close()

def get(search_key):
    db_file = open("/usr/local/database", "r")
    for line in reversed(db_file):
        key, val = line.split(",")
        if key == search_key:
            break
    db_file.close()
    return key, val
```

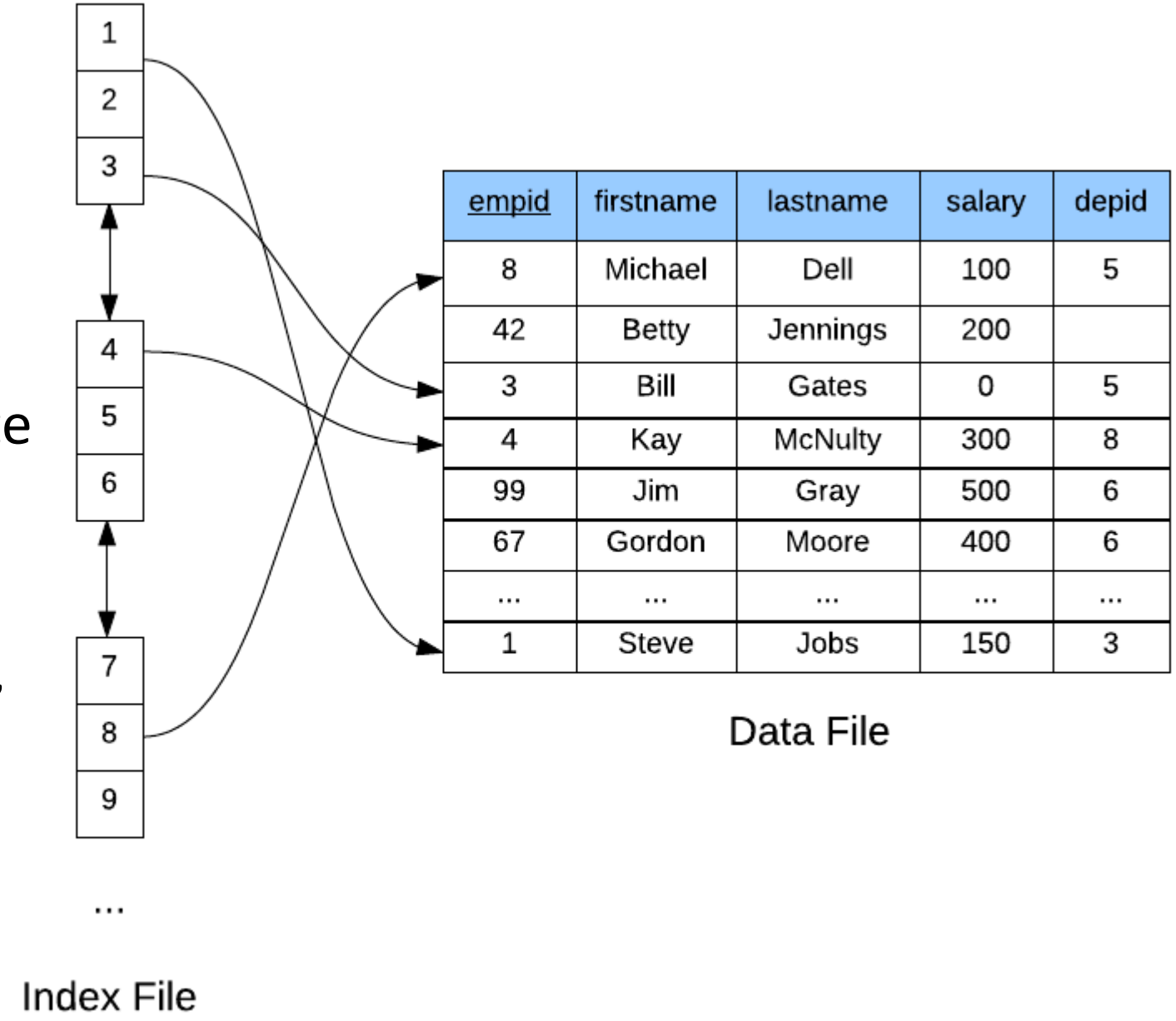
Realistic Database System



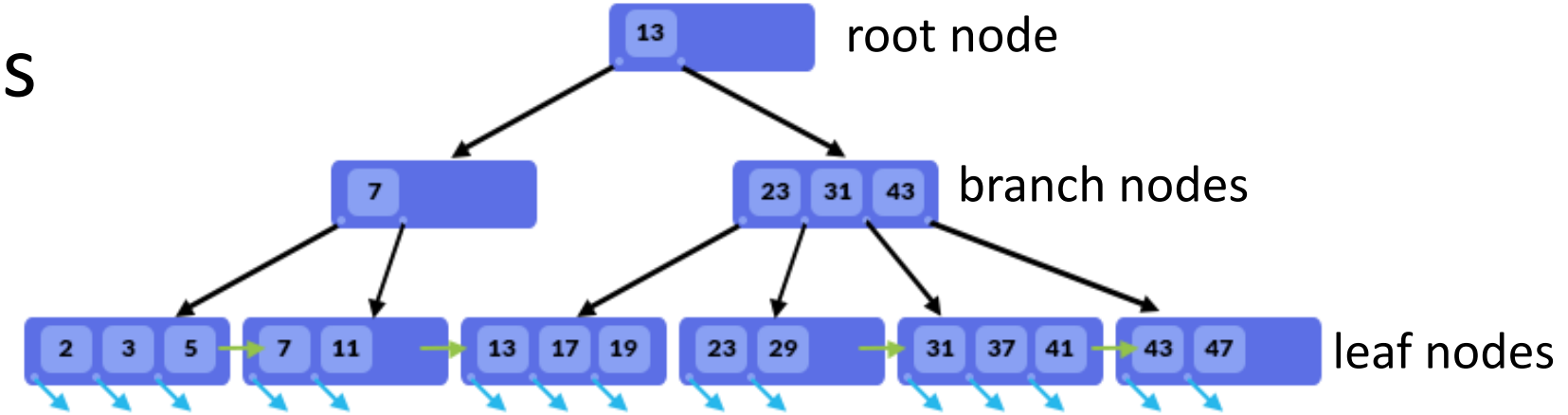
Source: Ramakrishnan and Gehrke, DBMS Systems, 3rd edition, 2003.

Database Indexes

- **Critical** to database systems
- At least one index per table
- DBA analyzes workload and chooses which indexes to create (no easy answers)
- Creating indexes can be an expensive operation
- They work “behind the scenes”
- Query optimizer decides which indexes to use during query execution



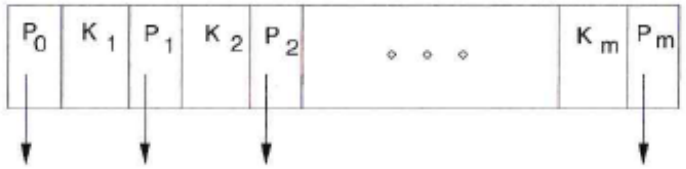
Properties of B Trees



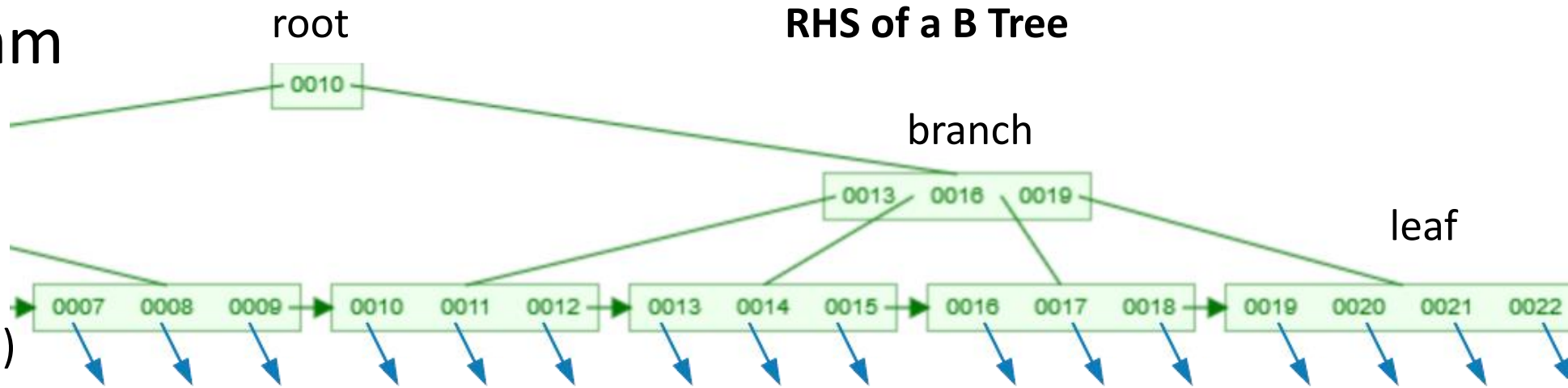
Structure of a B Tree

- height is balanced
- have several children
- data stored in the leaf nodes
- leaf nodes are ordered
- leaf nodes are connected (doubly linked list)
- each node stores several index entries
- index entry = (key value, pointer)
- search speed ≈ height of tree

Format of a Node



Search Algorithm



- Let S = Search Key
- Let K = Key Value
- An Index Entry = (K, P)
- Begin at root:
  - If S < K, follow K's left pointer
  - If S = K, follow K's right pointer
  - If S > K and K is not in last entry, scan forward to next entry
  - Repeat for each entry until last entry is reached:
    - If S < K, follow K's left pointer
    - If S ≥ K, follow K's right pointer
- Repeat until leaf node is reached
- Scan forward leaf node until K = S
- Follow K's pointer to row id in data file

Equality Query:

```
select *
from T1
where c1 = x;
```

Range Query:

```
select *
from T1
where c1 > x and c1 < y;
```

Choosing B Trees

Common use cases:

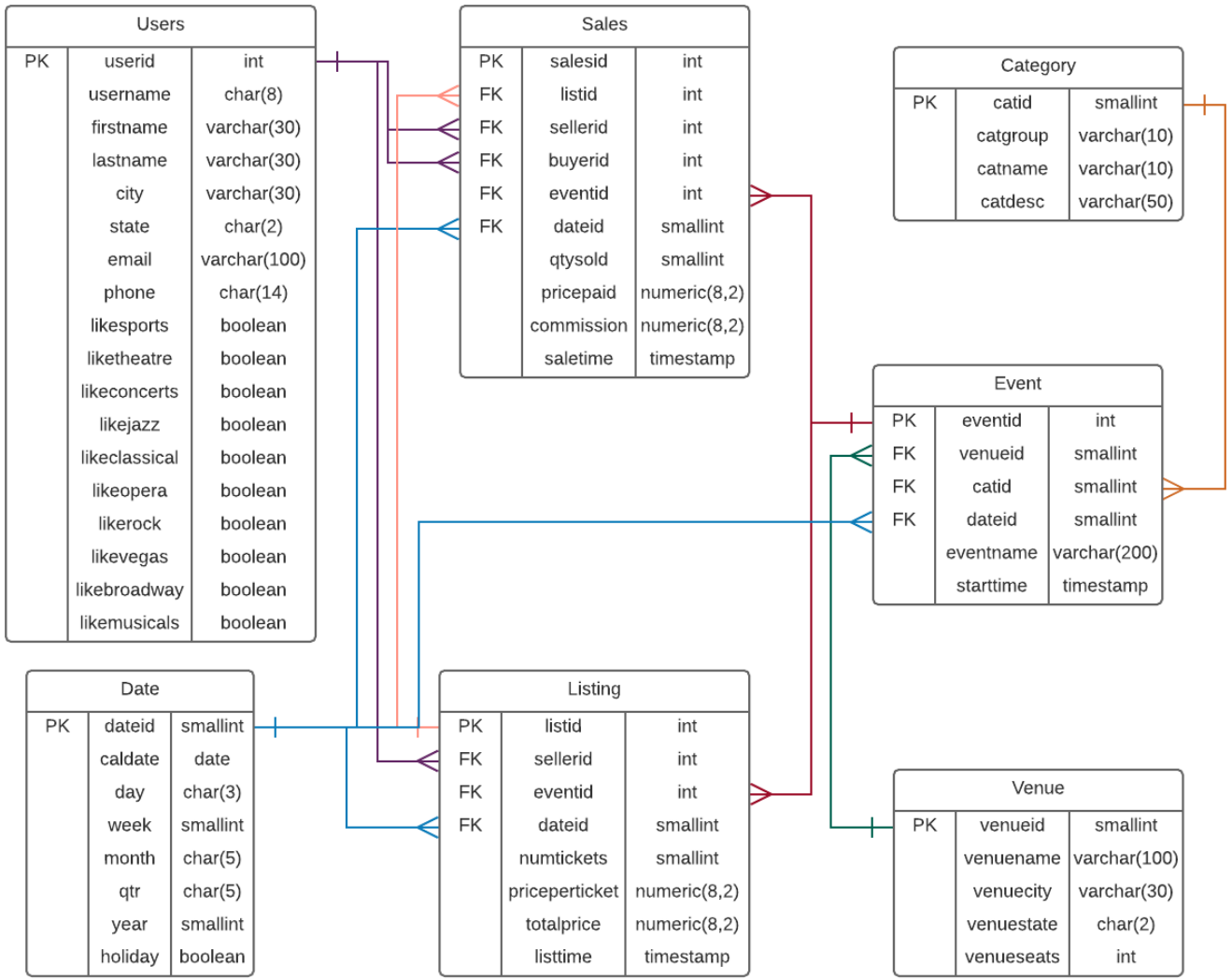
- Columns in WHERE clause
- Columns in JOINs

Other use cases:

- Columns in GROUP BY and ORDER BY clause
- Columns in SELECT clause

- **Not** low-cardinality columns (e.g. boolean columns)
- **Not** aggregated columns
- **Not** columns from multiple tables

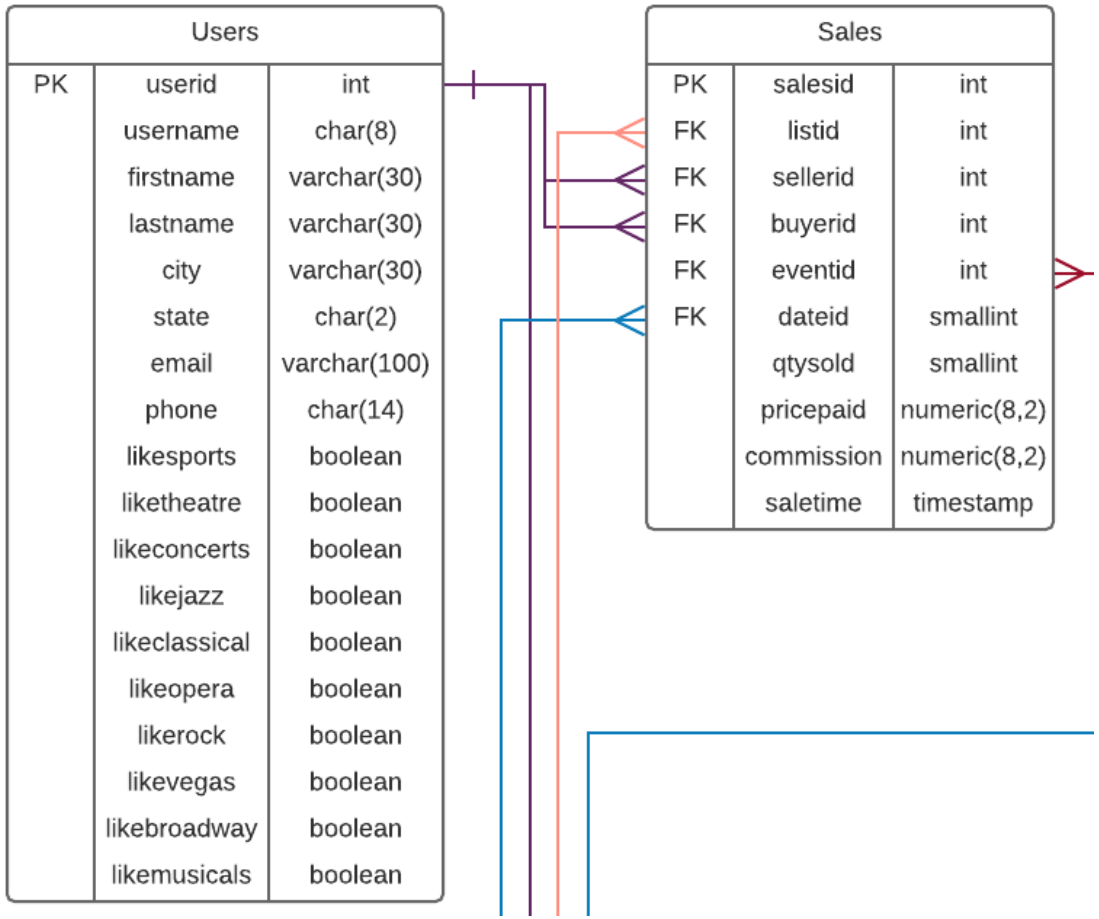
Demo: Optimizing Queries on TICKIT Database



TICKIT ERD: see [snippets] repo for dataset and code.

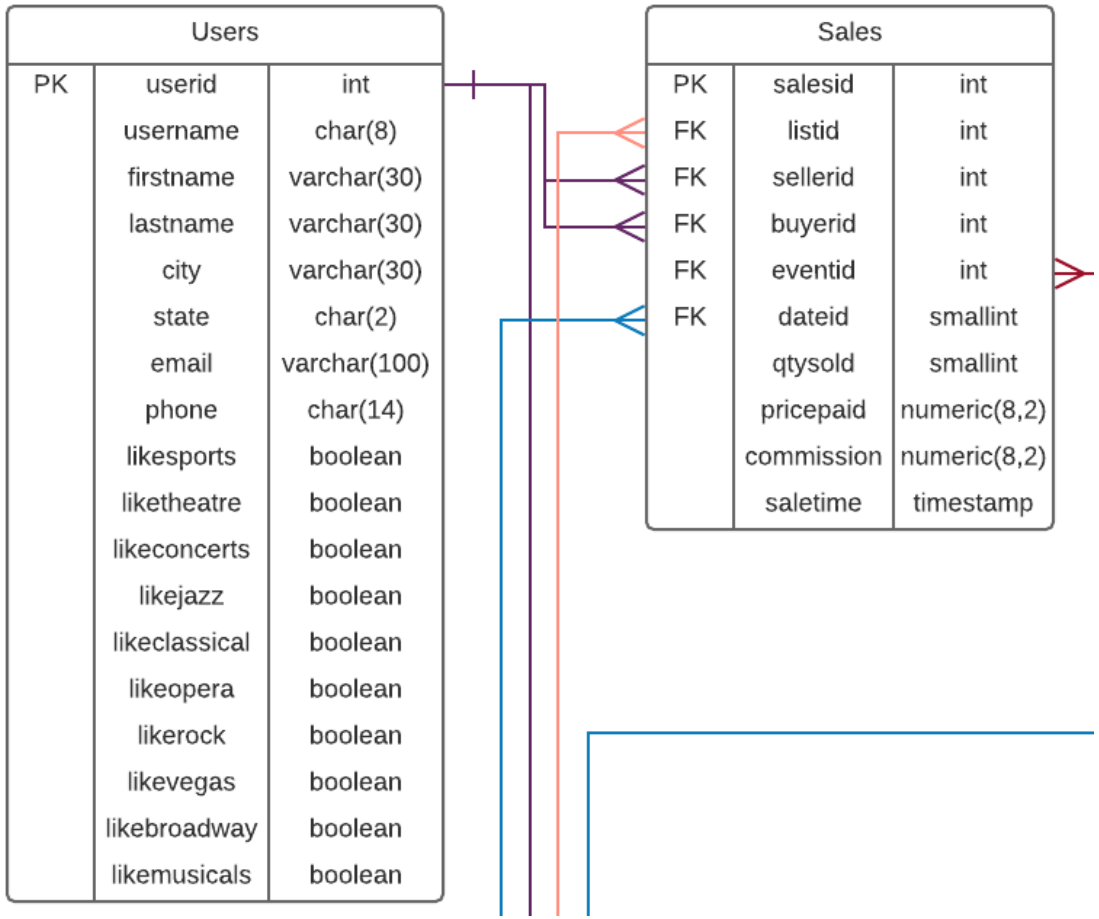
Practice Problem 1: Construct an index on the appropriate column(s) of the Sales table to optimize this query:

```
select s.sellerid, u.username, u.email, sum(qtysold)
from Sales s join Users u on s.sellerid = u.userid
group by s.sellerid, u.username,
order by sum(qtysold) desc;
```



Practice Problem 1: Construct an index on the appropriate column(s) of the Sales table to optimize this query:

```
select s.sellerid, u.username, u.email, sum(qtysold)
from Sales s join Users u on s.sellerid = u.userid
group by s.sellerid, u.username,
order by sum(qtysold) desc;
```



Which columns are contained in the index?

A) `sellerid`

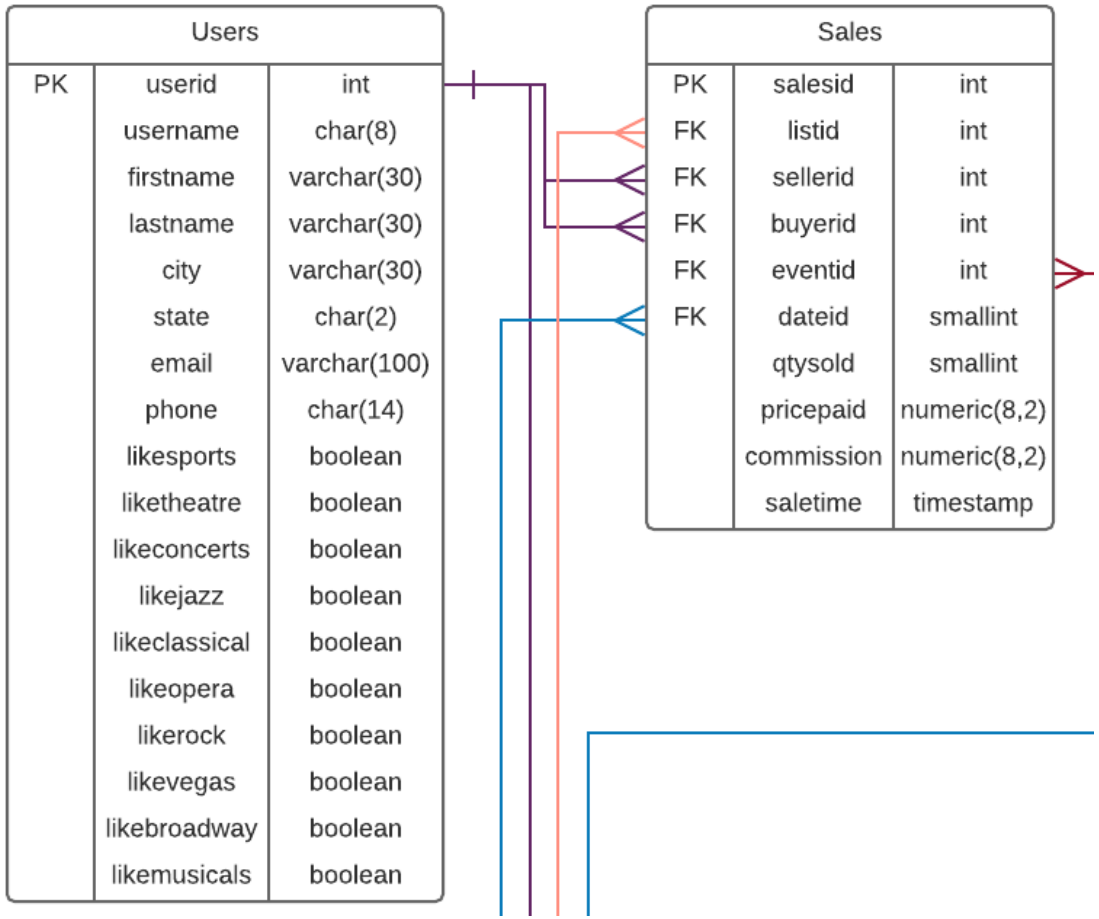
B) `qtysold`

C) `sellerid`, qtysold

D) None of the above

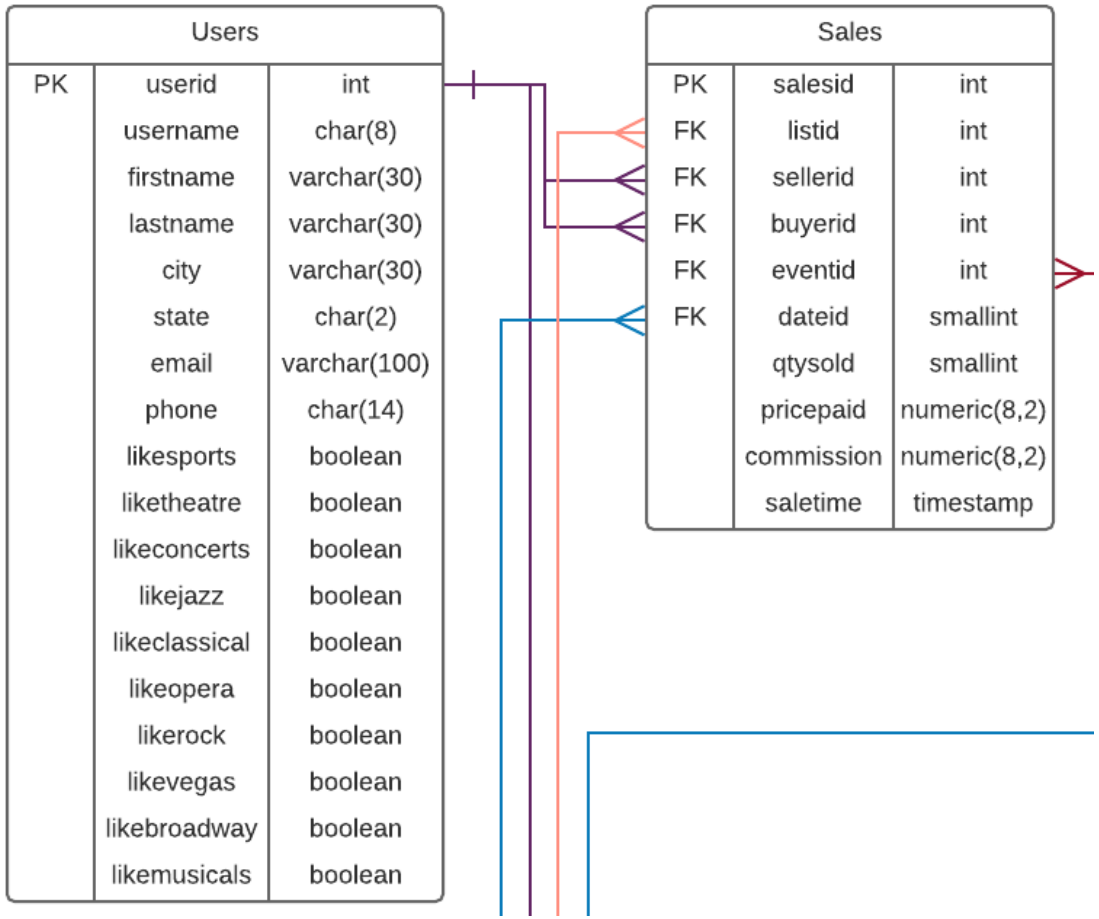
Practice Problem 2: Construct an index on the appropriate column(s) of the Users table to optimize this query:

```
select s.sellerid, u.username, u.email, sum(qtysold)
from Sales s join Users u on s.sellerid = u.userid
where u.city = 'Houston'
group by s.sellerid, u.username,
order by sum(qtysold) desc;
```



Practice Problem 2: Construct an index on the appropriate column(s) of the Users table to optimize this query:

```
select s.sellerid, u.username, u.email, sum(qtysold)
from Sales s join Users u on s.sellerid = u.userid
where u.city = 'Houston'
group by s.sellerid, u.username,
order by sum(qtysold) desc;
```



Which columns are contained in the index?

A) city
B) userid, username, email
C) All of the above
D) None of the above