Data Modelling

Sancheeta Kaushal | Ex-EM Blinkit

Airtribe

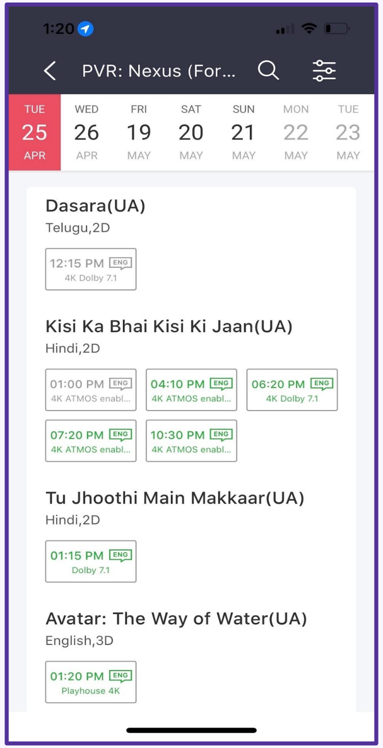
Problem Statement

Problem Solving Case: Bookmyshow

Bookmyshow is a ticketing platform where you can book tickets for a movie show.

As part of this assignment, we need to build API's for the following feature. As a user, I can select any theatre in the city. On selecting the theatre, I should be able to see the dates of next 7 days. I can click on any date and the page should load to give me all the movies in that theatre on that given date. Movies should contain details of all the showtimes.

Problem Statement


1:20

PVR: Nexus (For...

TUE	WED	FRI	SAT	SUN	MON	TUE
25	26	19	20	21	22	23
APR	APR	MAY	MAY	MAY	MAY	MAY

Dasara(UA)
Telugu,2D
12:15 PM ENG 4K Dolby 7.1

Kisi Ka Bhai Kisi Ki Jaan(UA)
Hindi,2D
01:00 PM ENG 4K ATMOS enabl...
04:10 PM ENG 4K ATMOS enabl...
06:20 PM ENG 4K Dolby 7.1
07:20 PM ENG 4K ATMOS enabl...
10:30 PM ENG 4K ATMOS enabl...

Tu Jhoothi Main Makkaar(UA)
Hindi,2D
01:15 PM ENG Dolby 7.1

Avatar: The Way of Water(UA)
English,3D
01:20 PM ENG Playhouse 4K

The API you wrote is very slow. The team has to now improve the performance of the API.

Airtribe

What all skills you need to learn to improve the performance ?

- Improving performance on the
    - Database level
    - ORM level
    - Application level

Airtribe

What all skills you need to learn to improve the performance ?

- Improving performance on the
  - Database level
  - ORM level
  - Application level

Airtribe

Agenda

- Query Lifecycle
- Slow Queries
- Optimising Performance at Database level using Indexing
- Optimising Performance at ORM level
- Optimising Performance at Application level using Caching
- Redis

Challenges with databases

- Consistency
- Availability
- Scaling
- Security and Privacy
- Performance - Database, ORM & Application Layer

Airtribe

Why focus on database performance?

- Database as a bottleneck - 50-60% of API time is spent as DB I/O
- Customer Experience Expectation
- Performance is precursor to scalability
- Saves Cost

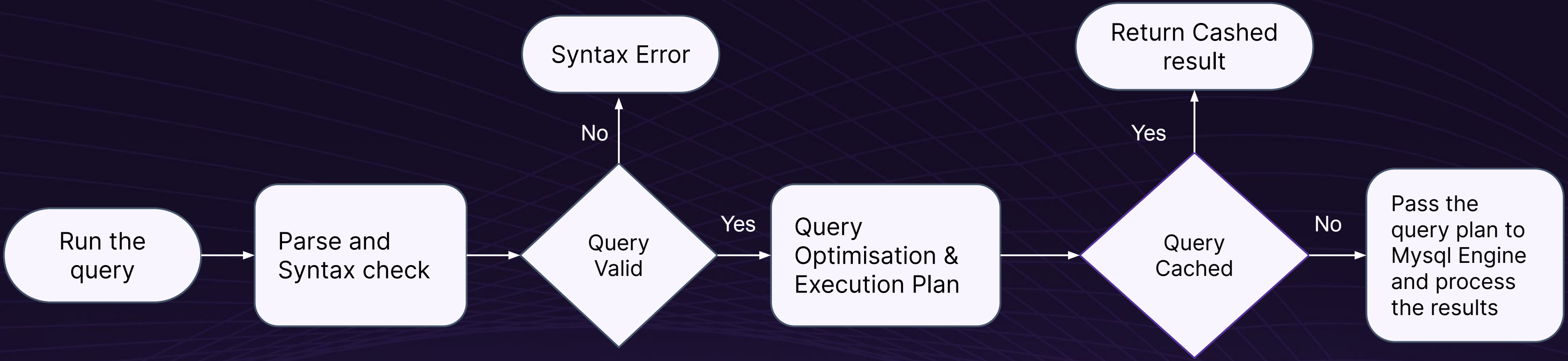
So what’s the culprit?

- Slow Queries
- A database query that takes longer to execute than what is considered optimal.
- What is optimal ?
- 100 ms - 500 ms API response time where 60% of the time spend in DB lookups.

Airtribe

Query Lifecycle

To understand why queries are slow, we must understand **Query lifecycle**



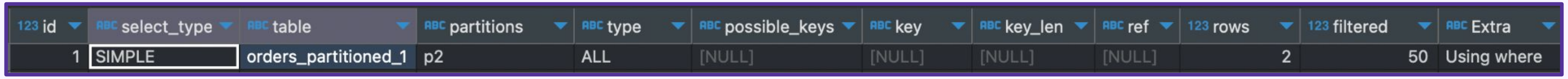
Airtribe

Analyse Queries using Explain

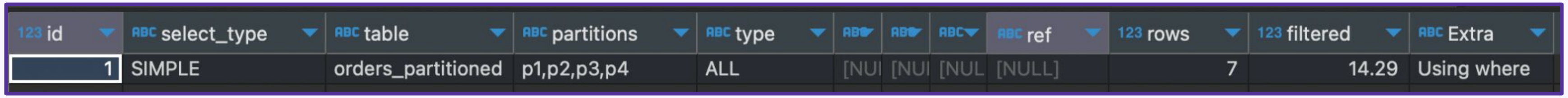
Explain shares the plan on how the MySQL optimizer will execute a SELECT statement

SQL Query Syntax

```
EXPLAIN SELECT * from orders_partitioned_1 WHERE id = 2 ;
```



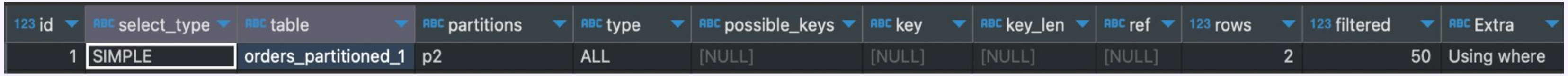
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	orders_partitioned_1	p2	ALL	[NULL]	[NULL]	[NULL]	[NULL]	2	50	Using where



id	select_type	table	partitions	type				ref	rows	filtered	Extra
1	SIMPLE	orders_partitioned	p1,p2,p3,p4	ALL	[NUI	[NUI	[NUL	[NULL]	7	14.29	Using where

Airtribe

Analyse Queries using Explain



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	orders_partitioned_1	p2	ALL	[NULL]	[NULL]	[NULL]	[NULL]	2	50	Using where

- id: A unique identifier for each SELECT statement within the query.
- table: The name of the table being accessed.
- type: The type of join or access method used, such as ALL, index, eq_ref etc.
- rows: The estimated number of rows examined or returned by the query.
- filtered: The percentage of rows filtered by table conditions.
- Extra: Additional information about the query execution plan, such as using where, temporary tables, sorting, etc.

Explain an old query

What is the average total amount spent by customers from each city, for orders placed after 3rd May 2022, ordered in descending order of the average amount?

SQL Query Syntax

```
EXPLAIN SELECT c.city, AVG(o.total_amount) AS average_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date >= '2022-05-03'
GROUP BY c.city
ORDER BY average_amount DESC;
```

Now that we know query execution, let's think about what can we done to speed up the query performance ?

- Select only the columns that are required
- Use appropriate joins based on relationships in real world
- Remove the data that is no longer required
- Indexing

Indexes

- Data structure that improves the speed and efficiency of data retrieval operations in a database.
- Very similar to the index section in the books
- Provides a quick lookup mechanism to locate specific rows based on the indexed column values
- Created on one or more columns of a table using B-tree data structure
- Column to index is decided based on most frequent queries. Check your WHERE Clause or JOINS conditions.

Airtribe

Syntax – Index Creation and Deletion

Create Index Syntax

```
CREATE INDEX index_name ON table_name(column_name);
Or
ALTER TABLE table_name ADD INDEX index_name (column_name);
```

Drop Index Syntax

```
ALTER TABLE table_name DROP INDEX index_name (column_name);
```

Show Index Syntax

```
SHOW INDEX from table_name;
```

```
// Define the User model
const User = sequelize.define('User', {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
  },
  // Other columns...
}, {
  indexes: [
    {
      unique: true,
      fields: ['email'],
      name: 'email_index',
    },
  ],
});
```

Airtribe

Explain an old query

What will happen if we add an index ? What column should we add index to ? Will it change the number of rows that need to be fetched ?

SQL Query Syntax

```
EXPLAIN SELECT c.city, AVG(o.total_amount) AS average_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date >= '2022-05-03'
GROUP BY c.city
ORDER BY average_amount DESC;
```

Things to keep in mind – Indexing

- Additional storage space required
- Write operations can potentially become slow
- Complex Query plan with multiple indexes
- Cautious with new queries on same table
- Maintenance overhead as can’t be managed via ORM’s

Airtribe

N+1 Query problem

Eager/Lazy Loading

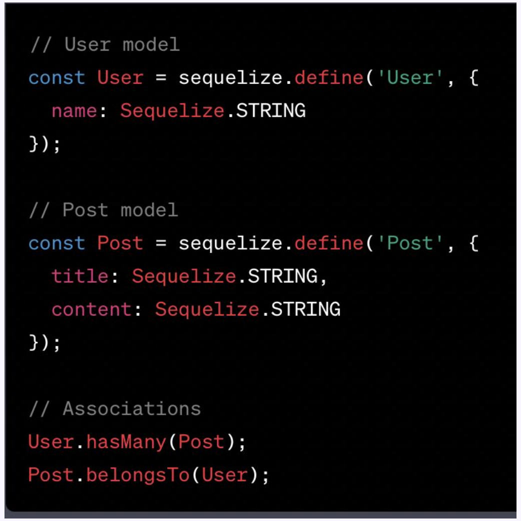
ORM Optimisations

Connection Pooling

Batching & Limiting

Airtribe

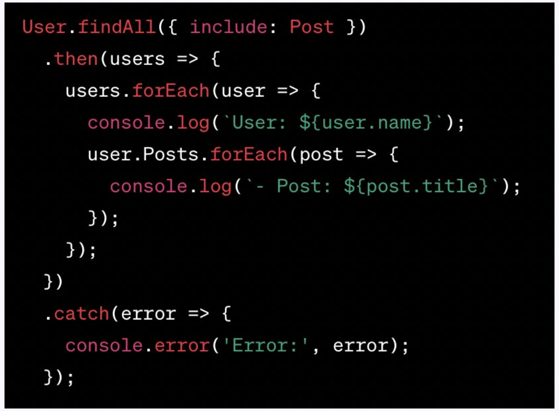
N+1 Problem & Eager Loading



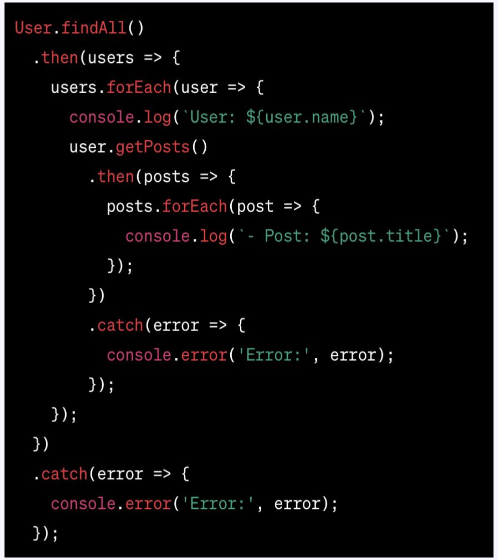
```
// User model
const User = sequelize.define('User', {
  name: Sequelize.STRING
});

// Post model
const Post = sequelize.define('Post', {
  title: Sequelize.STRING,
  content: Sequelize.STRING
});

// Associations
User.hasMany(Post);
Post.belongsTo(User);
```

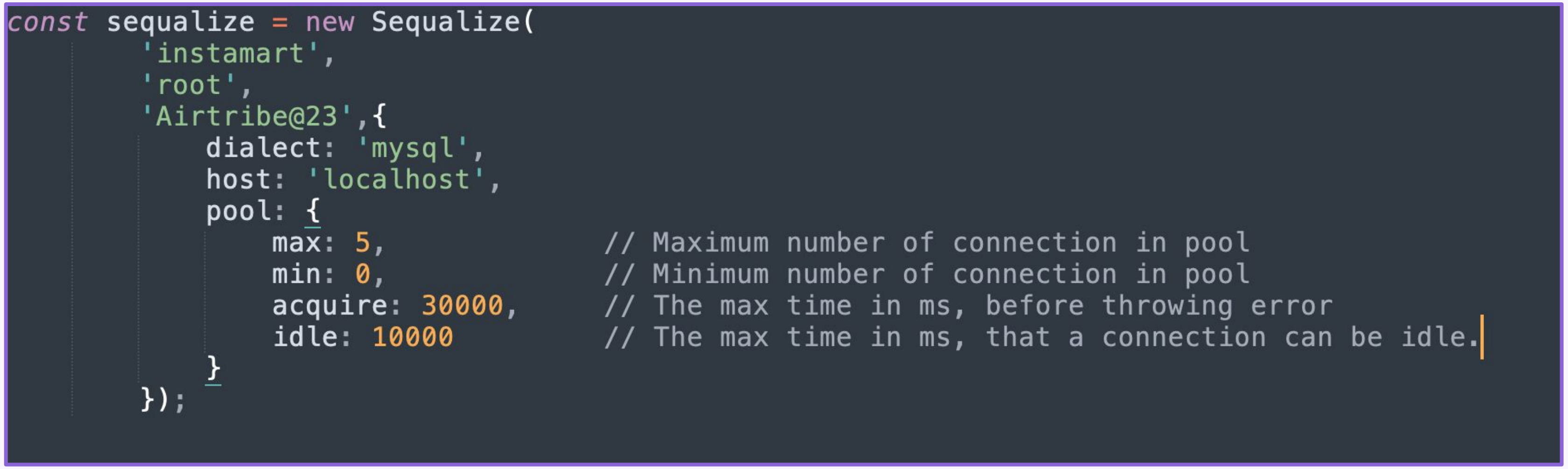


```
User.findAll({ include: Post })
  .then(users => {
    users.forEach(user => {
      console.log(`User: ${user.name}`);
      user.Posts.forEach(post => {
        console.log(`- Post: ${post.title}`);
      });
    });
  })
  .catch(error => {
    console.error('Error:', error);
  });
```



```
User.findAll()
  .then(users => {
    users.forEach(user => {
      console.log(`User: ${user.name}`);
      user.getPosts()
        .then(posts => {
          posts.forEach(post => {
            console.log(`- Post: ${post.title}`);
          });
        })
        .catch(error => {
          console.error('Error:', error);
        });
    });
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Connection Pooling



```
const sequalize = new Sequalize(
        'instamart',
        'root',
        'Airtribe@23',{
            dialect: 'mysql',
            host: 'localhost',
            pool: {
                max: 5,                     // Maximum number of connection in pool
                min: 0,                     // Minimum number of connection in pool
                acquire: 30000,             // The max time in ms, before throwing error
                idle: 10000                 // The max time in ms, that a connection can be idle.
            }
        });
```

Airtribe

Connection Pooling

- Max 10-20 in production.
- Max and min connection pool count depends on
  - Application Traffic
  - Database server capacity
  - Resource constraints (CPU, Memory)
  - Connection latency and usage patterns
- Blinkit story

Airtribe

Batching using bulkCreate

Or

Batching using Limit & Offset

Airtribe

Done everything, API is still slow

You have batched, indexed and eager loaded the data but still you feel that product read API should improved further for consumer experience.

The API became slow because we introduced 20-25 attributes per product like it’s color, dimensions, packaging, manufacturer etc. What is special about the product attribute data for a given product ?

What optimisation technique can be used when your data doesn’t change frequently but is accessed frequently?

- Caching
- Temporary storage location called a cache for frequent access
- Every resource which needs quick and frequent access to data can have a cache. eg. CPU, Memory, Disk, Web, Database, CDN etc.
- Server side vs Client side

Cache terminology

- Cache hit
  - Data found in cache and served directly.
- Cache miss
  - Data not found in cache, has to be fetched from main memory or data source. This reduces performance and creates issues if this happens frequently.
- Cache Invalidation
  - Process of removing cache entries when the corresponding data in the underlying data source becomes outdated. Eg. Manual, TTL and Event Based

Cache terminology

- Rate of Eviction
  - Frequency at which cache entries are removed or replaced from the cache to make room for new data.
- Eviction Strategies
  - LRU (Least Recently Used), FIFO (First In First Out), Random Replacement

Airtribe

Writing Data to Cache

- **Write Through**
  - Data written to cache and underlying storage at the same time ensuring data sync at all times.
  - Ensures consistency
  - Increases Write Latency

- **Write Back**
  - Data written only to cache and updates to underlying storage deferred. Modified data marked “dirty”. Written to storage at the time of eviction.
  - Lower write latency
  - Increased complexity and risk of inconsistency

Airtribe

Reading Data

- **Look Through**
  - Cache directly involved in data retrieval. If cache miss, data brought from source stored in cache and then returned.
  - Complex
  - Increases Latency
- **Look Aside**
  - If cache miss, data brought from data source returned to requestor and then stored in cache.
  - Simple
  - Potential stale data

Airtribe

Things to keep in mind

- Our goal is to maximise hit rate
- Amount of memory allocated basis the kind of data. Too small memory leads to high miss rate. Too large leads to unoptimised lookups.
- Balancing act of eviction, cache invalidation, reading and updation.
- Cache invalidation is the process of removing or marking cached data as invalid when the underlying data in the source or database changes.
- Cache eviction is the process of removing items from the cache to make space for new data. It is a mechanism to control the size of the cache and manage memory resources effectively.

Redis

- In-memory data structure store that can be used as a key value store, database, cache, and message broker
- In memory means data accessed from RAM
- Persists data to disk - dump.rdb
- Supports Data structures - Strings, lists, sets, sorted sets, hashes, bitmaps

Airtribe

Setting up Redis Cache in Codebase

- You have already installed the redis server and redis cli using brew install.
- Now you need to install Redis Library in package.json as dependencies.
- Next we create a config to write code to connect with Redis.
- We test the connection on the server launch.
- Notice the dump.rdb (Redis Backup File) getting created. This is an in-memory database leading to quick access of data.

Redis DataTypes and Operations

Basic String CRUD operations (GET, SET, DELETE)

Redis Query Syntax

```
SET key val        //key and val both are strings
> SET product_grammage_1 1kg

GET key
> GET product_grammage_1

DEL key
> DEL product_grammage_1

keys * (Get all keys)
```

Airtribe

Redis DataTypes and Operations

- List operations (PUSH, POP, TRIM)
- Set operations (ADD, REMOVE, INTERSECT)
- Hash operations (HSET, HGET, HDEL)
- Sorted Set operations (ZADD, ZRANK, ZRANGE)

Key expiry and Eviction

EXPIRY and TTL

Redis Query Syntax

```
EXPIRE key ttl_in_seconds
> EXPIRE product_discount_1 10

TTL key
> TTL product_discount_1

SETEX key value ttl_in_seconds
> SETEX product_discount_1 100 3600
```

Airtribe

Redis get key

```javascript
// Route handler to get data from Redis
app.get('/data/:key', (req, res) => {
  const key = req.params.key;

  // Read data from Redis
  client.get(key, (error, result) => {
    if (error) {
      console.error(error);
      return res.status(500).json({ error: 'An error occurred' });
    }

    if (result) {
      // Data found in Redis
      const data = JSON.parse(result);
      return res.json({ data });
    } else {
      // Data not found in Redis
      return res.status(404).json({ error: 'Data not found' });
    }
```

Airtribe

Redis Middleware to cache & invalidate data

```javascript
// Middleware function to check if data exists in Redis cache
function checkCache(req, res, next) {
  const { productId } = req.params;

  // Check if data exists in cache for the given product ID
  redisClient.get(productId, (err, data) => {
    if (err) {
      console.error(err);
      res.status(500).send('Internal Server Error');
    }

    if (data !== null) {
      // Data found in cache, send the cached response
      res.send(JSON.parse(data));
    } else {
      // Data not found in cache, proceed to the next middleware or route
      next();
    }
  });
}
```

```javascript
// Middleware function to cache product attribute data
function cacheData(req, res, next) {
  const { productId } = req.params;
  const productData = {
    id: productId,
    name: 'Example Product',
    description: 'This is an example product.',
    price: 9.99,
  };

  // Store the product attribute data in Redis cache
  redisClient.setex(productId, 3600, JSON.stringify(productData));

  // Send the response with the product attribute data
  res.send(productData);
}

// Endpoint for retrieving product attribute data
app.get('/product/:productId', checkCache, cacheData);
```

Airtribe

Redis as a cache

- In-memory data store
  - Read from RAM vs Disk
- Efficient data structures
  - Help with quick retrieval → high performance.
- Data expiration and TTL
  - Easy invalidation
- Web page caching/Full page caching, API response caching, User session management
- Rate Limiting

Redis as a key value store

- Schema less data stores, so very fast reads and writes
- Options to persist data in both memory and disk (dump.rdb)
- Supports Data structures - Strings, lists, sets, sorted sets, hashes, bitmaps
- Real time analytics - Unique Visitors on a page, Event Logs, Game Leaderboards
- Configuration Storage for constants - Infra configs, Application configs, Business configs

Redis as a message broker

- Publisher/Subscriber model. Publishers can send messages (publish) to specific channels, and subscribers can receive messages (subscribe) from these channels.
- Topic-based message routing allows for message filtering and selective consumption of messages by subscribers based on their interests.
- Data persistence for consistency of delivery of message.
- Blocking queue features
- Real-Time Messaging and TTL Features
- Real time chat or instant messaging systems
- Task Queue for asynchronous systems like Celery

Redis as a database

- In-memory nature leads to extremely fast data access and real-time processing capabilities.
- Supports Atomic Operations and Transactions
    - Counter increment and decrement
    - Distributed locking
- Machine Learning Model Storage
- Social media votes and likes
- Shopping cart operations
- User authentication

Airtribe

Thank You!

Airtribe