Monitoring and optimization

> Looking after the state od db and making it better

Monitoring

- Scrutinization (examination) of day-to-day operational database status
- Crucial to maintain RDBMS health and performance
- Reasons:
    - identify issues in timely manner
    - If you don't monitor, problems might go undetected
    - forecasting the future requirements
    - analyzing performance of application or queries
    - Tracking usage of tables and index
    - determining root cause of system degradation
    - assessing impact of optimization
    - optimizing the db for performance
- most RDBMS have tools to monitor

Reactive monitoring Vs Proactive monitoring

- Reactive => after issue occurs, react to it, reactive panic occurs
- Proactive => Identify issues before they grow larger
    - Observe metrics and send alerts
    - use automation
    - better strategy
- Steps for proactive:
    - determine baseline
    - Record key performance metrics at regular intervals over a given time period
    - Compare baseline with performance
    - If significantly below or lower, might need optimization
    - Some other info: (that will come from monitoring)
        - peak time
        - good time for backup and maintenance
        - time taken
- Ways:
    - Point in time(manual)
        - monitoring table functions
        - examine monitor elements and metrics
        - lightweight, high speed monitoring infrastructure
    - Historical (Automated)
        - Event monitors

    - Capture info on database operation (For eg: deadlock, usage percentage...)

Monitoring usage and performance

> You need key performance indicators(KPI), also known as metrics

- metrics allow DBAs to optimize organizations' databases for best performance
- Regular monitoring also helps in operations, availability, and security
- Issues might be caused by:
    - Hardware
    - Software
    - Network
    - Queries
    - something else

Levels of Monitoring

- Infrastructure level:
    - All underlying components: OS, Servers, Storage hardware, network components working properly
- Instance or Db platform level:
    - Platform like mysql, db2...
- Query level:
    - Bad query might cause bottlenecks
- User level:
    - just because there is not a issue right now, doesn't mean there wont be.
- Monitoring at all level is crucial to maintaining SLA(Service level Agreements):
    - High availability
    - High uptime
    - Low latency

Key databse metrics:

- Throughput: how much total work is being taken (queries per second)
- Database resource usage: how much of cpu, memory , log and space usage. Avg, max, latest and time series number
- Data availability
- Database responsiveness: How fast
- Database contention: Simultaneous access from processes
- Units of work: what transactions are consuming most resource
- Connections: no of connections
- most frequent queries

- locked objects
- Stored procedures: metric for SP
- Buffer pools: for cached data
- Top consumers

> There are more

Monitoring tools:

- Admin dashbord or performance dashboard
- query profiler

Third party tools

- pganalyze for postgresql
- PRTG Network Monitor
- Solarwinds
- Quest foglight
- Datadog

> There are more

Optimizing database

Causes:

- Identify bottlenecks
- Fine-tune queries
- Reduce response times

> Each RDBMS have their own optimization commands

- In MySQL: OPTIMIZE TABLE command
- In PostgreSQL: VACUUM and REINDEX commands
- In Db2: RUNSTATS and REORG commands
- After significant CRUD, database get fragmented
    - Optimize table command will defragment
- VACUUM reclaims lost storage consumed by dead tuples
    - AUTOVACUUM works
    - VACUUM with or without parameter
    - FULL parameter can clain full space but slower
- REINDEX
    - rebuilds an index

    - if corrupt or mostly unused index

Using indexes

> Helps you find information faster

- Can improve performance
- Ordered copy of selected columns of data
    - Enables efficient searches without searching every row
- Columns defined based on frequently searched terms
- Lookup table points to original rows in table
    - Can include one or more columns

> Tradeoff between performance and maintenance(and storage). Need to find balance

- Narrow Index: few columns, less space but performance lower
- Wide index: more columns...
- Types:
    - Primary key: non-nullable, always unique, only one per table, data is stored in same order as PK (clustured)
    - Indexes: Non-clustered, unique or not, single or multiple columns

> Custom ordering is important

- You can create PK when creating or add later(useful for composite)
    - Auto incrementing is good for most cases
    - `AUTO_INCREMENT` in mysql
- Creating index:

```
CREATE UNIQUE INDEX unique_name
    ON project(projname);

CREATE INDEX job_by_dept
    ON employee (workdept, job);

DROP INDEX job_by_dept;
```

> Poorly designed or insufficient indexes => bottleneck

> Keep in mind db use, frequent queries, columnn characteristics(unique?)

- Index options(online), where to store

Slow queries:

- Causes:

- Size of db
- Unoptimized queries

```
EXPLAIN SELECT * FROM sales; -- this gives more information to show how the query performed
```

Improving queries:

- Be SELECTIVE with columns:
  - Only select columns you need not *
  - Avoid leading wildcards like `"%abc"` this finds value ending with specific characters, resulting in full table scan
    - Use full-text-index for improving this
  - Use the UNION ALL clause: when using OR operator with LIKE statement