Working with large databases (RDBMS)

Jay Kelkar

Jay Kelkar @jkelkar

Jay Kelkar

@jkelkar

Independent consultant

15 years as SAP Basis Consultant

25 years working with databases

Requirements

What we want from a database system:

1. Accuracy - consistent data
2. Safety - data stays available
3. Performance - for work + reporting

A modern RDBMS

Provides ACID compliance

In most cases, highly reliable

Works out of the box, for small instances

Larger systems need to be managed

Jay Kelkar @jkelkar

ACID

Atomicity

All changes are done or none!

No partial changes

Transactions - across multiple data

A C I D

Consistency

Satisfies all constraints

Triggers

Cascade - deletes/updates

Jay Kelkar @jkelkar

A C I D

Isolation

Parallel or serial operations has same outcome

Assuming all changes succeed

Jay Kelkar @jkelkar

A C I D

Durability

Data will continue to exist after a completed transaction

- even after a shutdown

Priority

The order of requirements has to be:

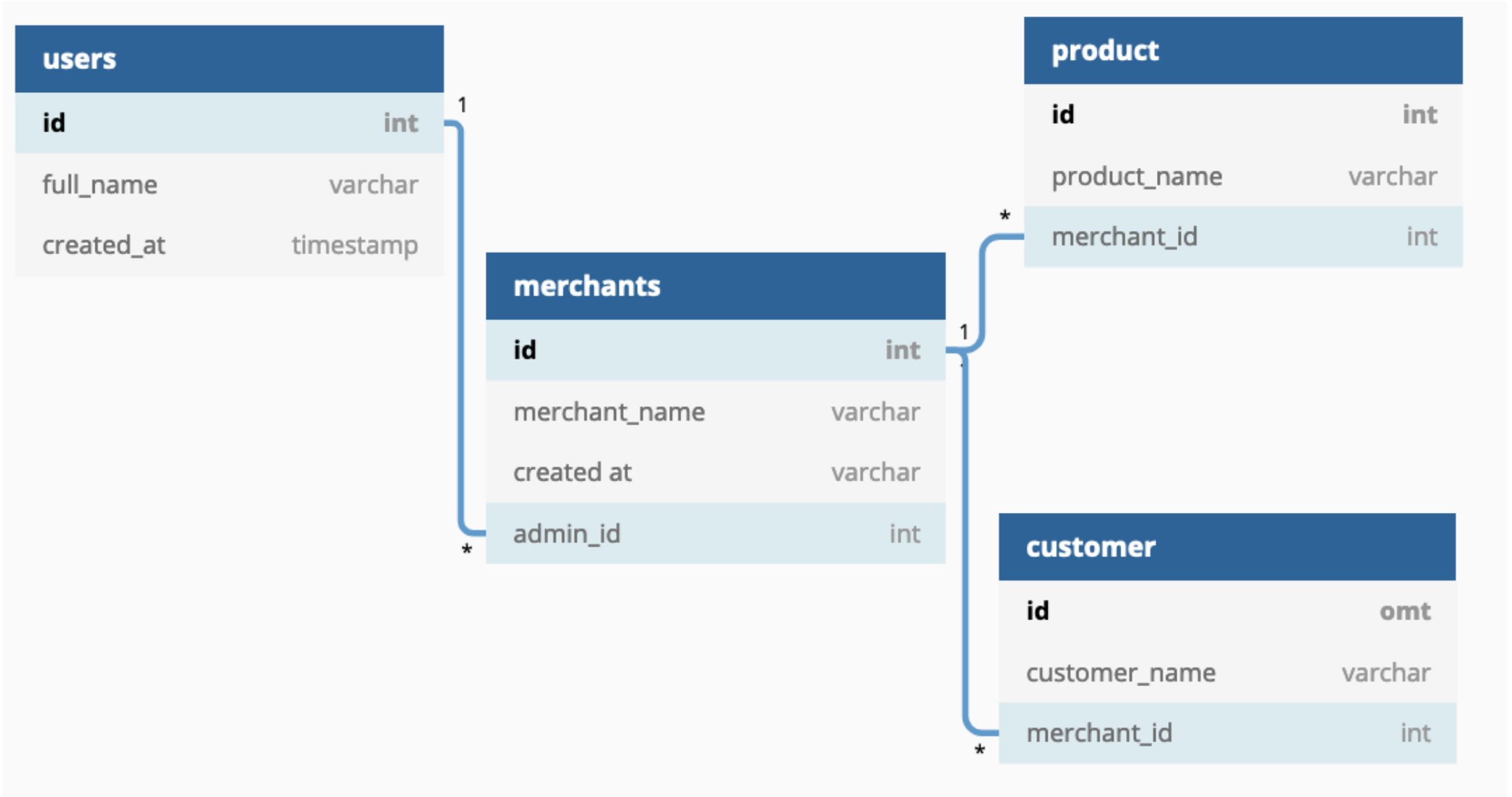
- Accuracy and safety - non negotiable

- Performance - important, not primary

Accuracy

Relationships

- Foreign key relationships
- Triggers
- Cascade deletes/updates



Relationships

Accuracy

Relationships

Foreign key relationships

Field data in foreign key columns exists only if it exists in the master table

Uses:

- master table access
- cascade deletes

Accuracy

Relationships

Triggers

Insert/update data into table following some action

- Relieves programmer of default tasks
- Audit tables
- Keeps data consistent

Accuracy

Relationships

Cascade deleted/updates

Automatically delete or update rows based on Foreign Key relationships

Uses:

- (delete) prevents orphan rows
- (update) set values in tables with FK on updates - in orphaned row

Safety

Backup and Restore

- Full backups
- Hot backups
- Redo log backups

Jay Kelkar @jkelkar

Safety

Full backups

- At reasonable intervals - weekly?
- Shut down database & copy all database files
- Lock database and dump structure & data as SQL (only for smaller databases)

Safety

Full backups

Pros:

- File backups are easiest to restore
- SQL backups essentials create all DB objects and load data (insert/copy data)

Cons:

- File backups require system to be quiescent
- SQL backups require database to be locked

Safety

Hot backups

Pros:

- Can be run when DB is productive

Cons:

- Can take longer
- Must put database in backup mode
- By itself cannot restore a database

Safety

Redo log backups

Pros:

- Only way to do a point in time recovery
- Fully supported by DB software

Cons:

- Requires careful sizing of redo log files
- Need to backup redo log files often

Safety

Saving backups

- * Must keep remote copies of backup data
- * Ideally, have backup data pulled off the system. Adds a level of safety
- Ideally keep 2 copies of redo logs on remote

Safety

Durability

- Redo log file sized on risk level (mins.)
- File size is ~1.5 - 2X for size of redo entries for a given time interval for busy system

Copy off a redo log file to backup ASAP

- smaller file size means more files

* Force log switch before hot backup

Safety

*Verify Backup integrity

- Must verify integrity of backup
- Only way to know if a backup is useable!
- Check regularly - Monthly?

Safety

*Limit Access

- Limit # users with direct access to DB
- ALL other access through known/ tested code - no exceptions

Safety

Changing table structure

Suggestion:

- Prefer not to “ALTER TABLE” directly

Instead:

- Dump, drop, create and restore

A lot higher safety + a copy of data

Safety is highest priority

"Happiness is hard won, but unhappiness comes easily"

–Anonymous

Performance

Query execution

- A query is analyzed
- Execution cost is estimated - the plan
- The plan with the lowest cost is chosen

* Explain queries

Run “explain” on queries

Pros:

- Shows expected I/O from DB
- Shows if/which indexes are being used

Cons:

- Have to understand output
- Results can change over time

Performance

* Create needed indexes

Why do we need indexes?

- Access a data row - unique key index
- Get to part of data - sorted index
- Complex query support - multi column
- Avoid full table scan

Performance

Index

- By default PRIMARY INDEX exists - on the unique KEY column
- An index can be non unique
- Only create an index for a use case
- Adds cost to inserts, updates & deletes
- Multiple columns can be in one index

Jay Kelkar @jkelkar

Performance

More about indexes

- Column order matters
- Query and Index columns must match

Option:

- Adjust Index to suit query
- Create a new index

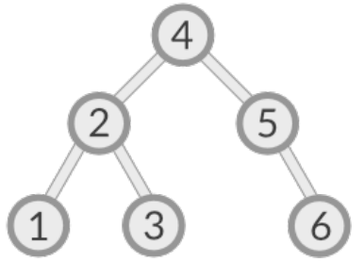
Performance

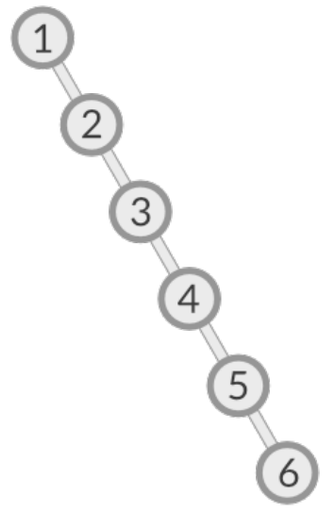
Maintain indexes

Indexes often are kept as b-trees

Over time they can get unbalanced

Balanced





Unbalanced

Performance

* Maintain indexes

Maintenance:

- Keep indexes up to date - drop/create or reindex [database | table | index]
- Check if index is relevant - cost Vs. benefit

Performance

Index planning

For high performance - adjust fill factor

- space used in each leaf page

lower - 60+

most space used

Dynamic Tables

higher ~ 100

least space used

Static Tables

Affects indexing or reindexing

Empty space used for insert/updates

Jay Kelkar @jkelkar

Performance

Statistics

Metadata about table data

- # unique entries per table column

Gotcha:

- Does not update automatically
- Out of date statistics result in bad query performance

* Update Statistics

Command varies by database:

- Typically you ANALYZE a table
- May use “ESTIMATE” or “COMPUTE”
- Analyzing 10-20% of data in a huge table is sufficient

* Log slow queries

A must with customer devs writing queries

- Queries can get slow over time
- Due to out of date statistics
- Due to query/index changes

Performance

Use parametric queries

If possible:

- “prepare” statements
- Use “stored procedures” & functions for Higher performance
- Can use existing execution plan

Performance

Good design

- * Balance between “normalization” and “joining” tables
- Plan “how much” data has to be read for queries
- * Use “views” when possible

Performance

DB Tuning

DB Buffers - various names

- about 25% of memory if available
- Larger amount if dedicated db server

Using a tuning specialist can give high returns in improved performance

kelkar

Performance

Data access speed

Keep data on the fastest device

- * Have separate caches on App Server
- * Secondary caches restricted to “most used data”, etc.
  e.g. Redis for certain tables

Performance

Table fill factor

For high performance - adjust table fill factor

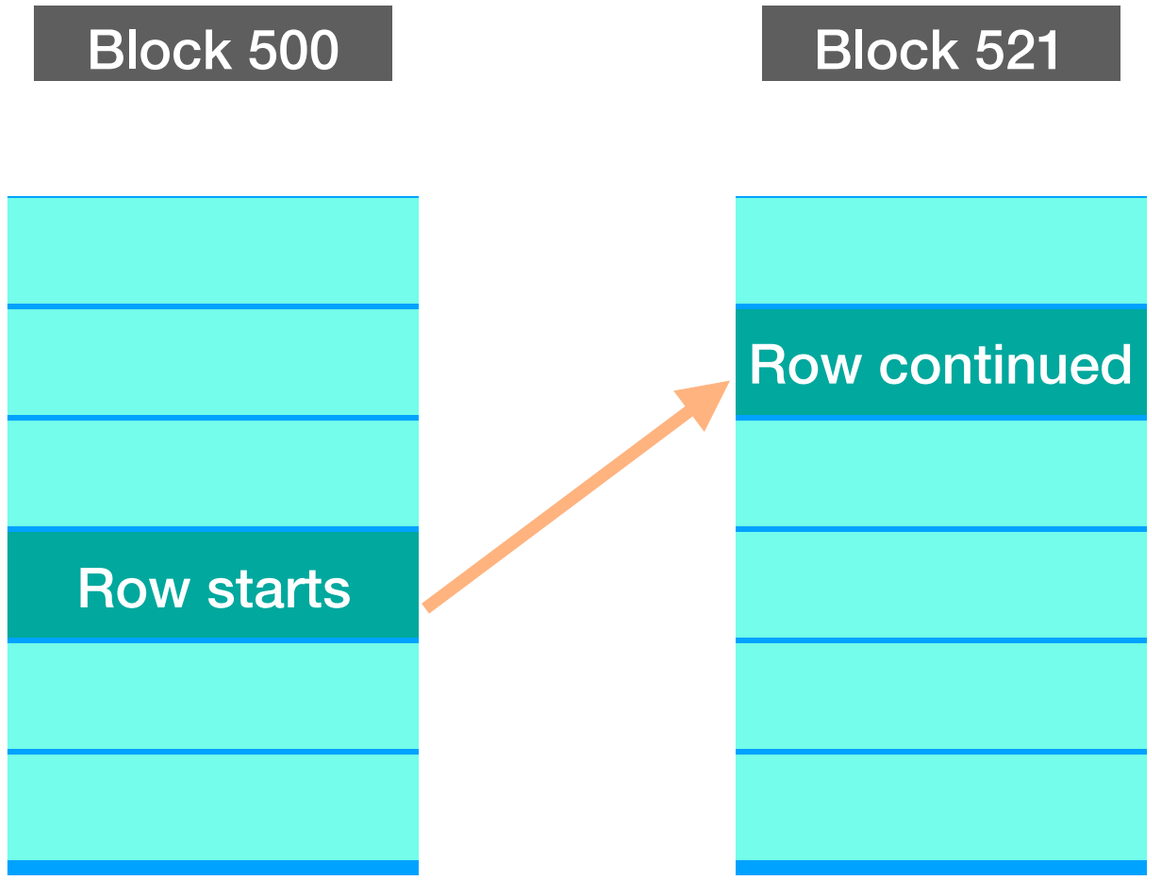
- only controls “insert” fill, rest for updates

lower - 60+ higher ~ 100

Dynamic Tables **Static Tables**

Performance

- prevents “row chaining”



Performance

Types of data stored

Transactional data:

- most data - OLTP

Non transactional data (need not log):

- derived and repeated data
- analytics, metrics, statistics, OLAP
- anything that can be recreated

Track Performance

- slow queries
- tracking backup: full, hot and redo
- Cache hits - should be high > 90-95%
  Can tell health of the system
- figure out the working set size for your application (and set buffer size)

Performance

* Offload DB disk I/O

- Offload DB disk access as possible
- Implement large caches (db buffers)
- Implement application caches

Have to deal with cache invalidations

Selected data

- * Select no more data than needed

Unnecessary data:

- Uses more DB cache, less efficient (LRU)
- Increases I/O and CPU cost of processing
- Increases data processing in application

Performance

Some suggestions

- * Use cryptographically secure passwords (size matters)
- Use secure connections between machines
- Limit select set size (possible?)
- Could prevent large amount of data loss during an attack

Performance

Reporting

- If possible report on aggregate data
- Disk space is cheap - time feels expensive
- Cache quality is not sacrificed
- Aggregate data during slow times

Performance

DB I/O Patterns

- Isolate Table data I/O from Index I/O
- Typically separate tablespaces (areas)

Keep separate network data flow
  for DB data and User data

Jay Kelkar @jkelkar

General

Version control

- Consider versioning your data objects, esp. Tables, Views and Indexes
- Makes it easy to keep track of changes
- Ideally set up a data dictionary to manage all db objects

Overall

Relative importance

- In everything we have seen, aim for high accuracy and high safety
- In the worst case, we can fudge on performance
- This gives us a reliable system, and
- Performance can always be improved

kelkar

The End

Jay Kelkar @jkelkar