Database Operation

http://goo.gl/QvSEz0

CS5200 DBMS
Bruce Chhay

Database Operation

L1. Transactions

Integrity of a Database

- Physical factors: power failures, errors, crashes, concurrency, etc.
- Transaction: unit of work that may consist of multiple read/write operations.
- Syntax:

  ```
  START TRANSACTION
  … statements ...
  COMMIT | ROLLBACK
  ```
- Alternatively without START TRANSACTION, each statement that modifies a table is automatically committed.
- Transactions have ACID properties.

Atomicity

- Atomicity - all operations in a transaction must occur, or nothing occurs.
- Journaling through a chronological record of operations.
- Temporary copies of data being modified by active transactions.
- Commit: transaction is complete, write temporary data to disk (usually), so another transaction can be started.
- Rollback: undo operations in the transaction, using the journal and discarding temporary copies (undo log).

Consistency

- Consistency - a transaction brings the database from one valid state to another.
- Future transactions can view the effects of past transactions, but earlier transactions cannot view the effects of future transactions.
- Integrity constraints are not violated (pk, fk, cascade, etc.).

Isolation

- Isolation - ability to reproduce the same results when multiple transactions are occurring at the same time.

Isolation

Different levels of isolation trade off concurrency performance:

- **Serializable**: use read/write locks on entire table(s) so that concurrent execution of transactions yield results as if they were executed one after another (standard SQL default).

Isolation

Different levels of isolation trade off concurrency performance:

- **Serializable**
- **Repeatable read**: lock on rows being read.
  Phantom read can occur: running the same query twice in the same transaction results in more records because a new record was inserted that matches the WHERE clause of the query.
  Consistent read using multiversion concurrency control (MVCC): instead of using locks, use snapshots based on when the first read operation is performed.

Transaction 1	**Transaction 2**
Phantom Read	
SELECT * FROM BlogPosts WHERE Title LIKE ‘%cat%’;	
	INSERT INTO BlogPosts(Title) VALUES (‘Moar cats’);
SELECT * FROM BlogPosts WHERE Title LIKE ‘%cat%’;	

Isolation

Different levels of isolation trade off concurrency performance:

- Serializable
- Repeatable read
- **Read committed**: can read any committed data (no read lock), so the read in a transaction depends on the timing of concurrent transactions.
  Non-repeatable read can occur: the same record is retrieved twice but the second time returns different values. Technically a violation of transactional isolation.

Transaction 1	**Transaction 2**
Non-repeatable Read	
SELECT * FROM BlogPosts WHERE PostId=1;	
	UPDATE BlogPosts SET Title='Cat nap' WHERE PostId=1;
SELECT * FROM BlogPosts WHERE PostId=1;	

Isolation

Different levels of isolation trade off concurrency performance:

- Serializable
- Repeatable read
- Read committed
- **Read uncommitted**: no waiting for other transactions.
  Dirty read can occur: retrieval of data that is not yet committed.
  No isolation guarantees.

Transaction 1	**Transaction 2**
Dirty Read	
SELECT * FROM BlogPosts WHERE PostId=1;	
	UPDATE BlogPosts SET Title='Cat nap' WHERE PostId=1;
SELECT * FROM BlogPosts WHERE PostId=1;	
	ROLLBACK

Durability

- Durability - once a transaction is committed, then it will remain permanently even after crashes.
- Write the transaction journal entry to persistent disk before commit is signaled.
- To recover from a crash, the committed transactions that were written to the journal but the results not yet saved to disk are re-applied (redo log).

Database Operation

L2. Storage

Storage

- MySQL default storage engine: InnoDB (Previously: MyISAM).
- Data in a table is divided into pages.
  - Each page is fixed at 16KB (most disks have 4KB sectors), and can contain one or more rows. Balance of size: large enough to hold the data for most rows, and small enough for transferring between memory-disk (especially when individual row is updated).
  - Compact format for nulls and variable-length fields, compressed saves more disk (at the expense of CPU).
  - Overflow pages: varchar, blob columns that are too long to fit on a page are stored in singly linked lists.

Storage

- Tablespace: data file (.ibd file) containing one or more tables and its indexes.
  - (Table metadata, such as table definition, is kept in .frm file.)
  - The system tablespace (ibdata1, ibdata2 files) contain metadata about the tables (data dictionary) and store journals (such as undo log; redo logs are stored in the log group: ib_logfile0, ib_logfile1 files).
  - Default is one file per table for easier space management and backups.
  - A table can be partitioned into multiple files to handle large tables.

Indexes

- Index: tree data structure for fast row lookup.
- Clustered index: storage organization of the entire table (all rows and all columns) based on the primary key columns to speed up queries involving the primary keys. A table only has one clustered index.
- Secondary index: index for a subset of table columns. A table can have any number of secondary indexes. Useful for queries that can be computed entirely from indexes without need to read actual table data (covering index). Otherwise, can be used to identify the relevant rows in a query, which are then retrieved using the clustered index. Creation involves overhead for copying data from the table. FK constraints create an index so parent-child checks can be done quickly (including FK constraint cascading).

Indexes

- Indexes are essential to performance and database administration. Choose pks wisely, based on expected queries. Expensive to modify which columns belong to the primary key due to the need to rebuild the clustered index.
- If only PK-FK joins are performed, then default clustered and secondary indexes usually suffice. Add more secondary indexes if:
  - Frequent JOINs do not use PK-FK relationships.
  - Frequent lookups for specific columns or column lists.

Index Syntax

```
CREATE INDEX index_name
    ON tbl_name (index_col_name,...)
```

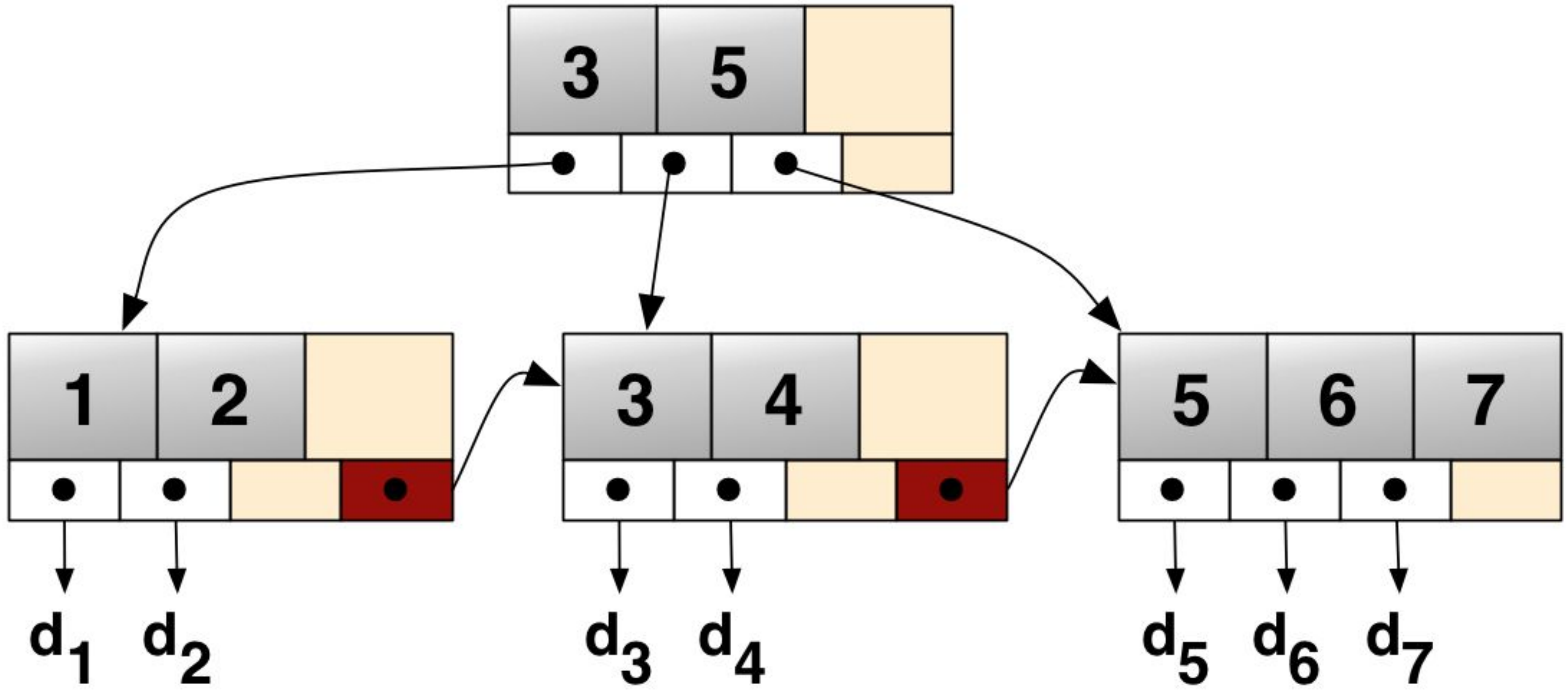
B-tree

- Search, insert, delete operations are O(log n) average and worst case.
- Storage is O(n).
- Sorted at all times to enable fast lookups for exact matches (equality operator) and ranges (greater than, less than operators).

B+ tree

- Structure (see next slide):
  - n-ary, variable number of children instead of two in a binary tree.
  - Keys are separators to divide subtrees. Keys chosen by index definition.
  - Values of internal nodes point to subtrees. Balance achieved by requiring all leaf nodes to have same depth.
- B+ tree: similar as B-tree
  - Leaves are linked (via linked-list) to allow fast range queries.
  - Leaves contains a key and value. Value contains row data (pages).
- (Alternative is a hash index, which only supports exact matches.)

B+ tree



http://upload.wikimedia.org/wikipedia/commons/3/37/Bplustree.png

Caching

- Buffer pool: in-memory, linked list of pages using least-recently-used caching for table (including results) and index data.
    - Can have multiple pools for better concurrency.
    - Pages (16KB) are transferred between disk and memory.
      (Disk sectors usually 4KB.)
    - Disk is 80x slower than memory (SSD is 4x slower than memory).

Caching

- Integration with memcached daemon (in-memory LRU caching layer) to store and retrieve directly from InnoDB.
    - (memcached API: get, set/add/replace, incr/decr.)
    - Clients can store/retrieve results from memcached directly and MySQL can store/retrieve from memcached.

Optimizations

- Normalization (and small lookup tables for frequently scanned values).
- Understanding logical query execution.
- Use EXPLAIN to view a query execution plan.

```
DESCRIBE BlogApplication.BlogPosts;
EXPLAIN SELECT * FROM BlogApplication.BlogPosts WHERE PostID = 1;
```

- DTrace for probing queries and resource utilization.
- Benchmarking.
- Storage, indexing, caching.
- Availability and scalability.
- And more...

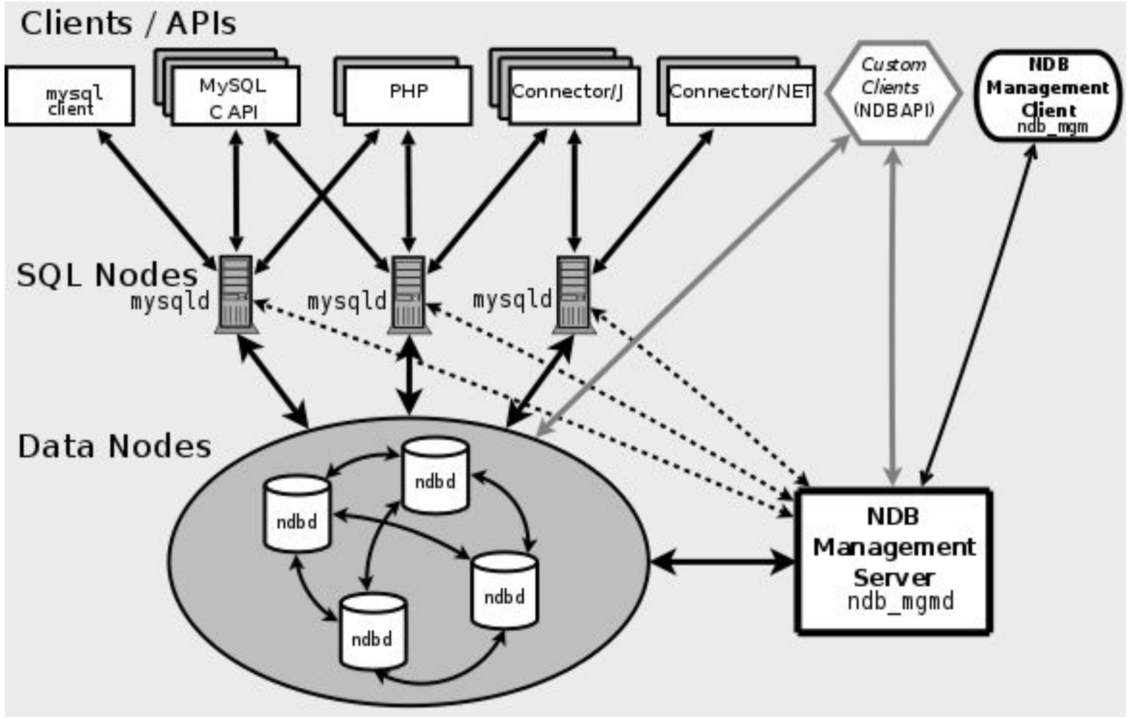
Database Operation

L3. Clustering & Replication

Clustering & Replication

- Availability and scalability through cluster nodes and data replication.
- Nodes: management nodes for coordination, Network Database (NDB in-memory storage engine) data nodes for storing tables (shards and replicas), and MySQL daemon (mysqld) for accessing data.

Clustering & Replication



http://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-overview.html

Clustering & Replication

- Consistency/Isolation can use two-phase locking (with a distributed lock manager):
  - Expanding phase: locks are acquired (none released).
  - Shrinking phase: locks are released (none acquired).
- Atomicity/Durability can use two-phase commit:
  - Commit-request: coordinator signals a vote from participants whether to commit a transaction or not.
  - Commit phase: coordinator decides to commit or abort based on the votes. Participants then commit or abort their local resources.

Storage Comparison

Feature	MyISAM	InnoDB	NDB
Transactions	No	Yes	Yes
Locking granularity	Table	Row	Row
MVCC	No	Yes	Yes
Storage limit	256TB	64TB	384EB
Compressed data	Yes	Yes	No
Data caches	No	Yes	Yes
Clustered indexes	No	Yes	No
B-tree indexes	Yes	Yes	Yes
Backup/recovery	Yes	Yes	Yes
Replication	Yes	Yes	Yes
Clustering	No	No	Yes

Projects

- Presentation signups.
- Milestone 5
  - CloverETL: what ETLs are teams using?
  - Examples of external data sources?
  - Interesting analysis and reporting?