Database - P43

CAP理论 (CAP theorem, Brewer's theorem), 用于设计和部署分布式应用

对于分布式数据存储系统(distributed data store)

同时最多只能满足三个核心需求中的两个

一致性(consistency), 可用性(availability), 分区容错性(partition tolerance)

一致性(consistency), 指所有结点, 访问同一份最新的数据副本

every read receive the most recent write or error

又或者说 all nodes see the same data at the same time

即更新操作成功并返回客户端完成后, 所有结点, 在同一时间的数据完全一致

从客户端看: 指多并发访问时更新过的数据如何获取的问题

从服务端看: 指更新如何复制分布到整个系统, 以保证数据最终一致

注意必须结合考虑并发读写的场景

强一致性(strong consistency): 即ACID中的Consistency

对于关系型数据库, 要求更新过的数据能被任何后续访问看到

同时也是CAP中要求满足的Consistency

弱一致性(weak consistency):

指可以容忍部分或全部的后续访问无法看到更新过的数据

最终一致性(eventual consistency):

指经过一段时间后, 所有后续访问可以看到更新过的数据

可用性(availability) 指每次请求均能获得非错误的响应, 但不保证获取的数据为最新版本

every request receive non-error response, without guarantee the most recent write

又或者说 read and write always succeed (服务一直可用)

对于满足可用性的分布式系统, 每个非故障的结点, 必须对每个请求作出响应

于是如果一个结点, 回报错误响应, 则认为结点, 是故障的

或者是超出响应时间, 结点, 没有做出回报

通常通过停机时间来衡量分布式系统的稳定性

分区容错性(partition tolerance). 指在遇到网络分区故障时, 仍能对外部保证满足一致性和可用性

system continue to operate despite arbitrary message loss or failure of part of system

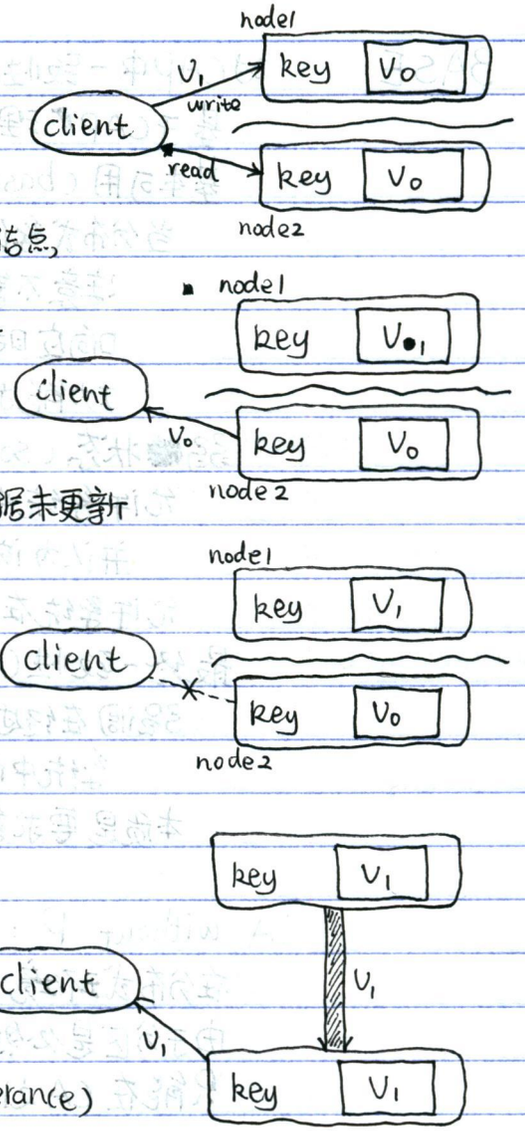
以实际效果而言, 相当于对通信时限的要求

如果无法在时限内达成数据一致性, 则认为发生了网络分区(partition)

分区容错性与系统的扩展性紧密相关

Database - P44

CAP理论证明，对于有两个结点的分布式系统，和一个客户端client
每个结点，都存储着键关键字key对应的值
初始状态时都存储着值V_0
假设此时发生了网络分区(partition)分隔了两个结点，
使client同时向node1发送write V_1的请求
并向node2发送read key的请求
则预期用node2响应回报数据V_1
考虑此时node2的可能行为
由于node1与node2被分隔，于是node2数据未更新
如果此时node2返回其存储的V_0
则不满足一致性(consistency)
如果此时node2由于已察觉网络分隔
从而回报错误信息(error)
或者长时间不进行回报
则不满足可用性(availability)
如果要求node2正常响应且回报数据V_1
则必定存在通路，保证node1的值V_1
经过通路更新至node2
于是此时满足一致性与可用性
但不再满足分区容忍性(partition tolerance)



于是可知对于分布式存储系统
一致性，可用性，分区容忍性之中最多满足两个

取舍

事务一致性：web实时系统通常不要求严格的数据库事务，允许实现最终一致性
读/写实时性：对于web应用，不高的读写实时性是可以接受的
多表关联查询：大数据量的web系统应避免多表关联查询和复杂的数据分析型的报表查询
而更多是单表主键查询和单表简单条件分页查询

CA　满足一致性，可用性，但缺乏扩展性：
tranditional RDBMS，如PostgreSQL，MySQL(relational)，Neo4j(Graph)

CP　满足一致性，分区容忍性，但是通常性能不高：
BigTable，HBase(column-oriented)，MongoDB(document-oriented)，Redis　(key-value)

AP　满足可用性，分区容忍性，但对一致性的要求降低：
Dynamo(key-value)，Cassandra(column-oriented)，CouchDB(document-oriented)

Database - P45

BASE 对CAP中一致性和可用性权衡的结果，来源于大规模互联网系统分布式实践的结论
基于CAP理论逐步演化，与实际结合的产物
基本可用（base availability）：相对于可用性（availability）
当分布式系统出现不可预知的故障，允许损失部分可用性
注意不等价于系统不可用
响应时间损失：出现异常时，结果返回用户出现一定的延迟
功能损失：为了保护系统的稳定性，引导部分用户至降级的服务
弱状态（soft state）：相对于ACID中的有效状态（valid state）
允许系统中的数据存在中间状态（intermediate state）
并认为该中间状态的存在不影响系统的整体可用性
允许系统在不同节点的数据副本之间进行数据同步的过程存在延时
最终一致性（eventually consistent）：相对于难以实现的强一致性（strong consistency）
强调在经过一段时间的同步过程后
系统中的所有数据副本最终能够达到一致的状态
本质是要求系统保证最终数据能够达到一致，而不需要实时保证 强一致性

CA without P：在分布式系统中几乎不存在
在分布式环境下，网络分区（partition）是一个自然事实
由于分区是必然的，则如果舍弃P即意味着舍弃分布式系统
只能在CA之间权衡，并且尽量提升P

CP without A：
分布式系统不要求强可用性，即允许系统停机或长时间无响应
一旦发生网络故障或消息丢失，则需要牺牲用户体验
等待所有数据全部一致，再开放用户访问系统
优先保证数据的强一致性，代价是舍弃系统的可用性

AP without C：
放弃一致性以获得高可用性并容忍分区
在节点之间失去联系时，需要在用户访问时立刻返回，以保证高可用性
此时每个节点仅使用本地数据提供服务，从而导致全局数据的不一致性
仅舍弃强一致性，退而求其次保证最终一致性
会影响用户体验，但不至于造成用户流程的严重阻塞

Database - P46

RDBMS (relational database management system, 关系数据库管理系统)
set-theory based system where data stored in two-dimensional table
table consist of data row and strictly enforced column type

SQL (structured query language, 结构化查询语言)
lingua franca (通用语) of relational database

PostgreSQL, one of finest open source implementations available
stored procedure can be written in different languages
and run on various platforms
built-in Unicode support
support sequence / table inheritance / sub select / ANSI SQL-compliant
fast / reliable / can handle TBs of data

psql command-line utility
prompt : name of database currently used
either dollar sign ($) as regular user
or hash mark (#) for administrator
\h : information about SQL command
\? : help with psql-specific command

table (relation) : set of tuples with same attributes
row (tuple) : object comprised of set of attribute values
column (attribute) : domain of values of certain type
CRUD : Create / Read / Update / Delete

index : special data structure built to avoid full table scan when query
automatically create index on primary key, specifically B-tree index
\di : list all indexes in the schema
not automatically create index on FOREIGN KEY targeted column
need to create index explicitly
B-tree : good standard index, flexible, default in PostgreSQL
Hash : index value unique, better for comparison operation
less flexible, not allow range query

Database - P47

PostgreSQL aggregate function : group result from rows together by common criterica

GROUP BY : place rows into group and perform aggregate function on each group

HAVING : like WHERE clause , except can filter by aggregate function
注意，WHERE 不能使用 aggregate function

DISTINCT : shortcut for only unique values

window function : similar GROUP BY , allow use built-in aggregate function
without requiring every single field grouped to single row
not collapse result , can return separate record for each row
如 SELECT type, count(*)
OVER (PARTITION BY type)] return result of aggregate function
FROM items OVER a PARTITION
ORDER BY type of result set

transaction : key concept for relational database consistency (all or nothing)
ensure every command of set executed
when anything fail , all command roll back as never happened
follow ACID compliance
atomic (原子性), consistent (一致性), isolated (隔离性), durable (持久性)
all commands executed implicitly wrapped in transaction
explicitly wrap : BEGIN TRANSACTION ;
SELECT * FROM items ;
ROLLBACK ;]显式指明结束，或者使用 COMMIT
import when modify different tables / rows , can't get out of sync

Stored procedure : accomplish run the code within PostgreSQL
incredibly powful , can perform enormous range of tasks
complex mathematical operation not supported in SQL
trigger cascading series of event to validate data before written
performance benefit / architectural cost may be too high

Database - P48

PostgreSQL stored procedure : 定义函数的基本语法结构为

```
CREATE [OR REPLACE] FUNCTION <name> (
    [<argname> <argtype>]*                          ]声明传入参数
) [ RETURNS <rettype>                               声明返回值 ]
  | RETURNS TABLE ( [column_name<column_type>]+ ) ]
pgSQL
    AS $$
    DECLARE                                         ]声明和定义局部变量
      [<argname> <argtype> [:= <value>]; ]*         ]
    BEGIN
      <statements>                                  ]定义函数体
    END;
    $$ LANGUAGE plpgsql;                            ]定义过程的编程语言，由$$包括
```

PostgreSQL support core language for writing procedure

plpgsql : PL/pgSQL , pltcl : PL/Tcl

plperl : PL/Perl , plpythonu : PL/Python

other language supported via community extension

Java / PHP / Scheme / ...

trigger : automatically execute stored procedure when event happen

can be used to enforce required behavior

in response to changing data , 如 INSERT / UPDATE / DELETE

定义 TRIGGER 的基本语法结构为

```
CREATE TRIGGER <name>
  {BEFORE | AFTER | INSTEAD OF}                     ]声明TRIGGER的时机
  {[<event>]* } ON <table_name>                     ]声明捕获事件和TABLE
  [FOR [EACH] {ROW | STATEMENT}]                    ]声明处理对象
  [WHEN (<condition>)]                              ]声明判断执行条件
  EXECUTE {FUNCTION | PROCEDURE}<function_name>(<arguments>)
    <event> ::= {INSERT | UPDATE [OF [<column_name>]*]
                | DELETE | TRUNCATE}
```

function_name : user-supplied function executed when TRIGGER fire

returning type : trigger

Database - P49

PostgreSQL RAISE NOTICE : log message to the console
注意这个语句是输出显示，而非抛出异常

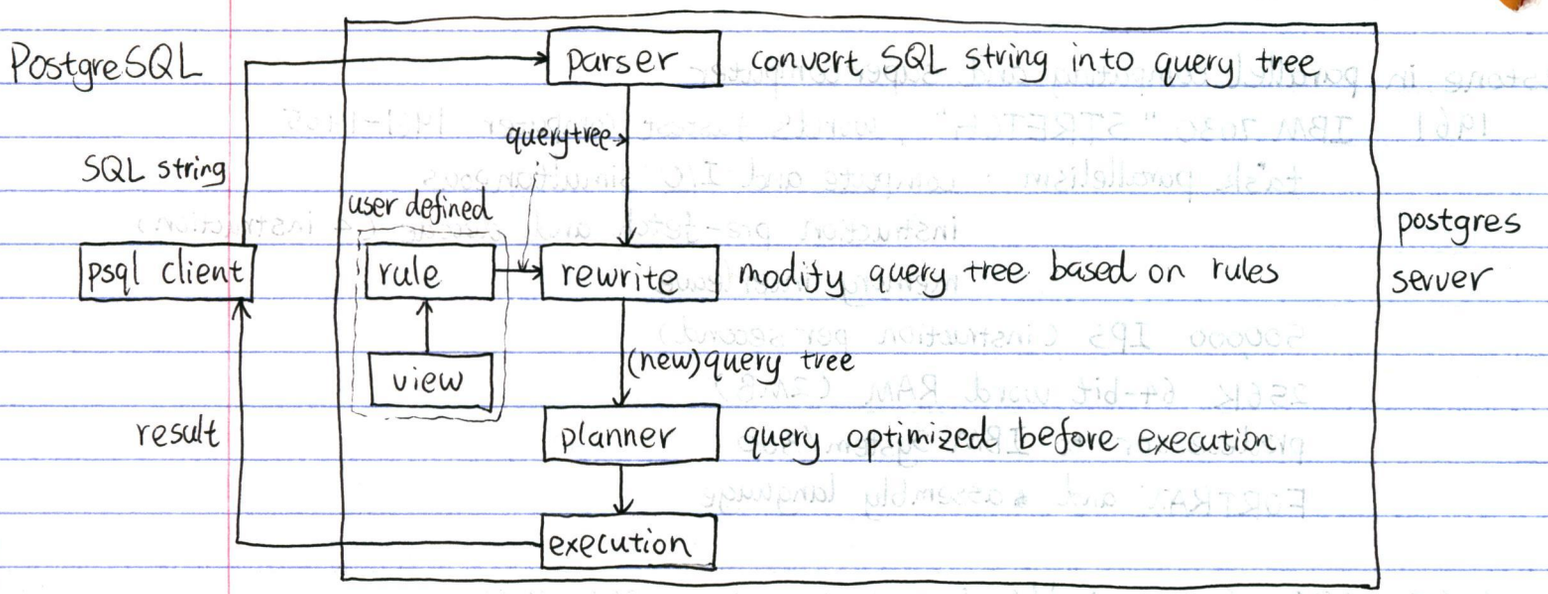
VIEW : view • result of complex query like table
基本语法结构类似于 SELECT 语句
CREATE VIEW <view_name> AS
{SELECT query}
view share data with the table it created from
performance of view not better than SELECT statement
unless create materialized view

MATERIALIZED VIEW : stored on disk like table
与 view 的创建命令一样：CREATE MATERIALIZED VIEW
materialized view can have index like table
performance worth the cost of extra disk space
especially view span multiple table / complex query

automatically updatable : simple views • are automatically updatable
system will allow INSERT/UPDATE/DELETE to be used
on view in the same way as on regular table
view satisfy conditions:
must have exactly one entry in FROM list
entry must be table or updatable view
must not contain at the • top level
WITH / DISTINCT / GROUP BY / HAVING / LIMIT / OFFSET
must not contain set operation at the top level
UNION / INTERSECT / EXCEPT
select list must not contain any
aggregate / window function / set-returning function

query tree : every time run SQL statement, parse the statement into query tree
generally referred to as abstract syntax tree
operator / value become branch / leaf , tree walked / pruned / modified before sent to query planner
query planner rewrite query so run optimally

Database - P50



EXPLAIN VERBOSE：描述给定query语句的query plan

RULE: allow alternative action to be performed on INSERT/UPDATE/DELETE
rule cause additional command to be executed when given command on given table
基本语法结构为：

```
CREATE RULE <name> AS ON <event>
    TO <table_name>
    [WHERE <condition>]
    DO [ALSO | INSTEAD]
    {NOTHING | command | (command ; command ...)}
<event> ::= SELECT | INSERT | UPDATE | DELETE
```

strength: years of research and production use across nearly every field of computing
flexible query (flexibility of join)
consistent and durable data
great programming language support with battle-tested drivers
programming modle (object-relational mapping, ORM) built on relational database
great at handling fairly homogeneous data
conforming well to structured schema

partitioning is not great strength of relational database

Database - P51

PostgreSQL CREATE EXTENSION : install custom extension, contributed package
not part of SQL standard, specific to PostgreSQL

crosstab : generally accomplished via pivot table (数据透视表)
construct "pivot" grouped data around other output
函数 crosstab(text <source_sql>, text <category_sql>)
source_sql : 生成 source set of data
返回包含 <row_name>, <category>, <value>
可以包含其他列 "extra" column
注意 row_name 必须是返回的第一列
category, value 必须是返回的最后两列
"extra" column expected to be same for all rows
with the same row_name value
category_sql : 生成 set of category
返回必须包含且仅包含一列，且不包含重复值
且至少包含一行，否则会报出错误

如 SELECT * FROM crosstab(
'SELECT year, month, count(*)
FROM table',
'SELECT * FROM months')
AS (year, jan, ···, dec);

→

year	jan	···	dec
2019	3	···	
2018		···	
⋮	⋮		⋮

extract : 函数 extract() return subfield from date / timestamp
如 SELECT extract(year from time) FROM table;

LIKE / ILIKE : help with matching data against partial / misspelled search criteria
supported by Postgres by default,
command method of performing text search
ILIKE is case-insensitive version of LIKE
wildcard : '%' match any number of character
'_' match exactly one character
如 '%_text' 表示 text 不在开头
'text_%' 表示 text 不在结尾

Database - P52

PostgreSQL

regex : regular expression, more powerful string-matching syntax
Postgres support POSIX style regular expression
match operator : [!] ~ [*]
~ (tilde) : lead regular expression string match
! (exclamation) : not matching
* (casterisk) : case insensitive

defined as one or more branches separated by |, match one of branches
branch : zero or more quantified atoms / constraints concatenated
empty branch match empty string
quantified atom : atom possibly followed by single quantifier
match a match of atom without quantifier
constraint : match only when specific condition met or empty string

Atom	description
(re)	match a match of re (regular expression) with match noted for possible reporting
.	match any single character
[chars]	bracket expression, match any one of chars (characters)
\k	match non-alphanumeric character k as ordinary character
\c	escape, alphanumeric character c, ARE (advanced RE) only
x	match single character x with no other significance

Quantifier	match
*	sequence of 0 or more matches of atom
+	sequence of 1 or more matches of atom
?	sequence of 0 or 1 match of atom
{m}	sequence of exactly m matches of atom
{m,}	sequence of m or more matches of atom
{m,n}	sequence of m through n (inclusive) matches of atom, $m \leq n$

greedy version : prefer largest number of matches

Constraint	description
^	match at the beginning of the string
$	match at the end of the string

Database - P53

PostgreSQL Levenshtein distance: string comparison algorithm compare how similar strings are
by how many steps required to change one string into another
replaced/ missing/ added character
levenshtein() provided by fuzzystrmatch contrib package
eg SELECT levenshtein('bat', 'fads'); → 3
'b' => 'f', 't' => 'd', + 's'

lower(): 用于将字符串转换为小写形式，可以直接在参数中传入列名
eg lower('AbCdE') → 'abcde'
相对的 upper() 用于将字母串转换为大写形式

trigram: group of three consecutive characters taken from string
pg_trgm module provide functionality to break string into as many as can trigrams
eg SELECT show_trgm('Avatar');
→ {" a", " av", "ar ", ata, ava, tar, vat}
finding matching string as easy as counting number of matching trigram
useful when doing search for string
with either slight misspelling(拼写错误) or minor word missing
the longer the string, the more trigrams and more likely a match

GIST: generalized index search tree (通用索引搜索树)
generic index API made available by PostgreSQL
generalization of B+ tree (对 B+ tree 的推广)
providing concurrent and recoverable height-balanced search tree infrastructure
提供并发和可恢复的高度平衡搜索树基础结构
without assumption about type of data stored or query servieed
不对存储的数据类型或正在服务的查询做任何假设
easily implement range of well-known index (轻松实现常见的索引)
B+ tree / R-tree / hB-tree / RD-tree
allow for easy development of specialized index for new data type
cannot be used directly to implement non-height-balanced tree
used for data type naturally ordered into hierarchy of superset
allow extension writer to support any query predicate choosen