CH2: A Hybrid Operational/Analytical Processing Benchmark for NoSQL

M. Carey, D. Lychagin, M. Muralikrishna, V. Sarathy, and T. Westmann

Couchbase, Inc., Santa Clara, CA, USA
mike.carey@couchbase.com

Abstract. Database systems with hybrid data management support, referred to as HTAP or HOAP architectures, are gaining popularity. These first appeared in the relational world, and the CH-benCHmark (CH) was proposed in 2011 to evaluate such relational systems. Today, one finds NoSQL database systems gaining adoption for new applications. In this paper we present CH2, a new benchmark – created with CH as its starting point – aimed at evaluating hybrid data platforms in the document data management world. Like CH, CH2 borrows from and extends both TPC-C and TPC-H. Differences from CH include a document-oriented schema, a data generation scheme that creates a TPC-H-like history, and a "do over" of the CH queries that is more in line with TPC-H. This paper details shortcomings that we uncovered in CH, the design of CH2, and preliminary results from running CH2 against Couchbase Server 7.0 (whose Query and Analytics services provide HOAP support for NoSQL data). The results provide insight into the performance isolation and horizontal scalability properties of Couchbase Server 7.0 as well as demonstrating the efficacy of CH2 for evaluating such platforms.

Keywords: Benchmarks · NoSQL · HTAP · HOAP.

1 Introduction

In today's online world, organizations are becoming ever more reliant on real-time information and its analysis to steer and optimize their operations. Historically, operational (OLTP) and analytical (OLAP) processing were separate activities, with each running on their own separate infrastructures; periodic ETL processes served to bridge these worlds in the overall architecture of a typical enterprise [13]. Today, database system architectures with hybrid data management support – referred to as HTAP (Hybrid Transactional/Analytical Processing [24]) or HOAP (Hybrid Operational/Analytical Processing [1]) support – are appearing on the scene and gaining traction in both industry and research in order to address the pressing need for timely analytics. Originating in the relational world, hybrid platforms are commonly linked to other concurrent high-end server technology trends; columnar storage and main-memory data management are two of the technologies that are often assumed to be part of that picture.

While relational databases still dominate the enterprise IT landscape, today's applications demand support for millions of interactions with end-users

via the Web and mobile devices. Traditional relational database systems were built to target thousands of users. Designed for strict consistency and data control, relational database systems tend to fall short of the agility, flexibility, and scalability demands of today's new applications. This has led to the emergence of the new generation of data management systems known as NoSQL systems [21]; our focus here will be on its sub-category of document databases. Examples of such systems include Couchbase Server [2] and MongoDB [8]. NoSQL systems aim to scale incrementally and horizontally on clusters of computers as well as to reduce the mismatch between the applications' view of data and its persisted view, thus enabling the players – ranging from application developers to DBAs and data analysts as well – to work with their data in its natural form.

Given this state of affairs, a natural question arises: Is the NoSQL world HOAPless[1], particularly document databases? The answer is no, as the need to combine operational and analytical capabilities for timely analytics is very much required in the NoSQL world as well. Thus, NoSQL vendors are beginning to focus on providing their enterprise customers with HOAP and are including HOAP-ful messages in their marketing materials. One such vendor, one whose document data management technology we will benchmark here, is Couchbase; the Couchbase Server platform introduced HOAP with its addition of Couchbase Analytics [7]. Our benchmark is not specific to Couchbase, however. It can be implemented and executed on HOAP-ful configurations of other NoSQL databases as well – and that, in fact, is the point of this paper: We propose a new benchmark for evaluating HOAPfulness in the world of document data management.

In the relational world, the problem of evaluating platform performance under hybrid OLTP/OLAP workloads has attracted attention in recent years. One example is the mixed workload CH-benCHmark [4] (CH) proposed by a stellar collection of database query processing and performance experts and now used by others to assess the performance of new HTAP systems [20]. The same is not yet true in the NoSQL world; there has yet to be a benchmark proposed to assess HOAP for scalable NoSQL systems. This paper proposes such a benchmark based on extending and improving CH in several important ways.

The remainder of this paper is organized as follows: Section 2 briefly surveys related work on HTAP systems as well as past SQL and NoSQL benchmarks. Section 3, the main event, describes our CH2 proposal for a HOAP-for-NoSQL benchmark. As an example of a HOAPful document store, Section 4 provides an overview of Couchbase Server and its approach to supporting HOAP. As a demonstration of CH2's potential to serve as an effective hybrid workload benchmark for NoSQL systems, Section 5 presents a first collection of results from running CH2 on a Couchbase server cluster in AWS under different service configurations. Section 6 summarizes the CH2 proposal and initial results.

[1] We prefer the term HOAP over HTAP in the context of NoSQL, as it seems less tied to strict ACID transactions and columnar, main-memory technology presumptions.

2 Related Work

We briefly review related work on HTAP/HOAP and database benchmarks.

2.1 HTAP (HOAP)

As mentioned in the Introduction, the relational database world has witnessed an emergence of HTAP capabilities in a number of vendors' systems in recent years as well as growing research interest related to HTAP. Notable HTAP offerings today include such systems as HyPer [9] (born in research, but now owned by and used in Tableau) and SAP-HANA [12]. Other significant commercial relational HTAP offerings include DB2 BLU from IBM [19], Oracle's dual-engine main-memory database solution [10], and the real-time analytical processing capabilities now found in Microsoft's SQL Server [11]. As an example on the research side, a recent paper introduced and explored the concept of adaptive HTAP and how to manage the core and memory resources of a powerful (scale-up) many-core NUMA server running a mixed main-memory workload [20].

Stepping back, one sees that R&D in the relational HTAP world has focused heavily on in-memory scenarios for relatively "small" operational databases. Now that multi-core servers with very large main memories are available, and given the degree of compression enabled by columnar storage, it is possible for main memory to hold much or even all of an enterprises' operational business data. As a result, most current HTAP database offerings rely on main-memory database technology. And, as would be expected, the focus of these offerings is on single-server architectures – i.e., on scaling up rather than scaling out.

In contrast, providing HOAP for scalable NoSQL document databases brings different problems that require different solutions. To scale document databases while providing HOAP, the focus needs to be on Big Data – and flexible, schema-less data. In addition, NoSQL systems and applications tend to have different transactional consistency needs [21]. Data timeliness is equally important in the NoSQL world, but there is less of a need to focus on the reduction or elimination of ACID transaction interference and more of a need to focus on the successful provision of performance isolation at the level of a cluster's physical resources.

2.2 Benchmarks

Many benchmarks have been developed to evaluate the performance of relational database systems under various application scenarios [6]. The most notable are the TPC-x benchmarks developed by the Transaction Processing Council (TPC). These include TPC-C [18] for a typical transaction processing workload as well as TPC-H [16] and TPC-DS [17] for decision support and analytics. There has also been a variety of benchmarks proposed and employed in the NoSQL world, including YCSB [5] for key-value store workloads, BigFUN [14] for Big Data management platform operations' performance, MongoDB's recent adaptation

of TPC-C to evaluate NoSQL transactional performance [8], and a philosophically similar NoSQL adaptation [15] of TPC-H to evaluate Big Data analytics performance, to name a handful of the NoSQL and Big Data benchmarks.

To evaluate HTAP systems, a particularly noteworthy effort was the proposal of the mixed workload CH-benCHmark [4]. This benchmark resulted from a Dagstuhl workshop attended by a group of database query processing and performance experts drawn from a variety of companies and universities. The CH-benCHmark combines ideas and operations from the TPC-C and TPC-H benchmarks in order to bridge the gap between the established single-workload benchmark suites of TPC-C, for OLTP, and TPC-H, for OLAP, thereby providing a foundation for mixed workload performance evaluation. The original paper included first results from applying the benchmark to PostgreSQL with all data being in memory and a read-committed isolation level. The CH-benCHmark appears to have gained some traction for HTAP use, having recently been used to assess the performance of a new HTAP system and its scheduling ideas [20].

To the best of our knowledge, our paper represents the first mixed workload benchmark proposed to assess HOAP for scalable NoSQL systems. A first exploratory step was reported in [23], where performance isolation in Couchbase Server (6.6) was investigated by mixing concurrent TPC-C NewOrder transactions with a stream of join/group-by/top-K queries. The effort reported here was suggested as future work at the end of that paper.

3 CH2 Benchmark Design

When we undertook the effort reported here, our goal was to explore several key aspects of NoSQL platforms' support for HOAP, including (1) their effectiveness at providing performance isolation between the OLTP and OLAP components of a mixed workload, and (2) the effectiveness of their query engines for handling OLAP-style queries. These were of particular interest because most NoSQL systems were designed to scale out horizontally on shared-nothing clusters and their initial design points for query processing have been OLTP-oriented, i.e., they were generally built to support high-concurrency/low-latency operational workloads as opposed to more complex data analytics.

Our first instinct was to design a new benchmark involving a mix of operational and analytical operations. Soon, with a healthy appreciation of the difficulty of coming up with a new schema, data, and workload, we found ourselves attracted to what MongoDB did in extending TPC-C to evaluate their new NoSQL transactional support [8]; in a project of our own we had followed a similar path by extending TPC-H for a comparative study of Big Data platform performance [15]. We decided to follow MongoDB's path for the operational side of the workload but to design our own analytical queries over the TPC-C schema for the analytical side of the mix in [23]. We then came across the mixed workload CH-benCHmark [4] for relational systems and were pleased to find that it took a similar approach. We then decided that the next step should be to "lightly" adapt the original CH-benCHmark to the document database world, but we

quickly encountered a number of issues that required more extensive changes. The rest of this section details CH2, the benchmark proposal that we landed on by attempting to adapt the original CH-benCHmark (henceforth referred to simply as CH) to the NoSQL document world.

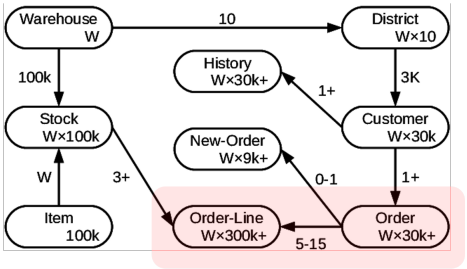


Fig. 1. TPC-C Schema (NoSQL Modification Highlighted)

3.1 Benchmark Schema

The bulk of the schema for CH2 is MongoDB's adaptation [8] of the TPC-C schema. Figure 1 summarizes the 9 tables and relationships of the standard relational TPC-C schema. This schema models businesses which "must manage, sell, or distribute products or services" [18] and it follows a continuous scaling model. The benchmark database size is scalable based on the number of warehouses (W), and the figure includes the scaling factors for each of the tables in the TPC-C schema. MongoDB's NoSQL adaptation of this schema involves 8 collections instead of 9, because in a non-1NF-limited NoSQL world, an order would naturally embed its line items as nested data. Figure 1 highlights the affected region of the TPC-C schema. No other nesting changes were made to the TPC-C schema, as in our view doing so would involve over-nesting and would be a poor database design for such use cases [8, 15].

In addition to adopting the nested order modification and TPC-C's scaling rules, the CH2 benchmark adopts CH's approach of borrowing 3 TPC-H tables as additional CH2 collections to support the adaptation of TPC-H's queries for the analytical side of a mixed workload. Following CH, CH2 borrows Supplier and Region, both unchanged, from the TPC- H schema, along with a slightly modified version of Nation. Supplier has a fixed number of entries (10,000), and an entry in Stock is associated with its Supplier through the CH relationship Stock.s_i_id × Stock.s_w_id *mod* 10,000 = Supplier.su_suppkey. A Customer's Nation is identified by the first character of the field Customer.c_state. In TPC-C this character can have 62 different values (upper-case letters, lower-case letters and numbers), so CH chose 62 nations to populate Nation (vs. 25 nations in TPC-H) and CH2 follows suit. The Nation.n_nationkey values are chosen so that their associated ASCII values are letters or numbers. Region then contains the five regions of these nations. Linkages between the new relations are modeled via the foreign key fields Nation.n_regionkey and Supplier.su_nationkey.

Table 1 lists the CH2 collections and gives an example of their scaling by listing their 1,000-warehouse cardinalities. Orders are nested with an average of

Table 1. CH2 Collections and Example Sizes with 1,000 Warehouses.

Collection	Collection Size (W=1000)
Warehouse	1,000
District	10,000
History	30,000,000
NewOrder	9,000,000
Stock	100,000,000
Customer	30,000,000
Orders (Orderline)	30,000,000 (300,000,000)
Item	100,000
Supplier	10,000
Nation	62
Region	5

10 Orderline items in each. The line separates the modified TPC-C collections (top) from the three CH (and CH2) additions (bottom).

3.2 Benchmark Data

To populate the CH2 database, we first tried using the generator from the original CH effort, but found that it had significant problems – being literally based on TPC-C's data generation rules – in terms of generating data for the analytical queries. Its main problem can be described as "TPC-C's big bang" – all of the date fields in the initial CH database had the current date (i.e., the date when the benchmark is run) as their value, so there were no date ranges to be found in the data. (TPC-H queries are designed to operate on a 7-year history.) Thus, while CH's queries were purportedly based on TPC-H, many of the CH query predicates returned either nothing, everything, or something run-date-dependent as a result. In a nutshell, the CH queries in their original form were not meaningful when combined with the CH data generator's data; a "do-over" was necessary.

To address the aforementioned problems, we created a new database generator by modifying the TPC-C data generator from the publicly available py-tpcc benchmarking package from CMU to (i) generate the orders with nested items (the MongoDB NoSQL change) and (ii) carefully control the values generated for the fields used in query predicates on the analytical side of the benchmark (the CH2 data/query change). Regarding (ii), we introduced a RUN_DATE parameter to seed the data generation; RUN_DATE then leads to derivative parameters START_DATE and END_DATE that are similar to the date range in TPC-H. RUN_DATE controls the historical characteristics of the data by determining when the benchmark's past should end and when the operational workload's life should begin, i.e., START_DATE = RUN_DATE - 7 years and END_DATE = RUN_DATE - 1 day. In the process, we also fixed the dates in the data generator based on TPC-H, added the Supplier, Nation, and Region collections (as a change relative to py-tpcc, ported from the CH generator), and then (as will discuss shortly) fixed the date ranges and predicate ranges in the analytical workload's queries to conform to the TPC-H predicate selectivities and the TPC-H query business semantics. In all, the fields involved in these changes were: Cus-

tomer.c_since, Orders.o_entry_d, Orders.ol.ol_delivery_d, History.h_date, and also Supplier.su_nationkey (value generation changed to avoid skew).

3.3 Benchmark Operations

The operational workload of TPC-C models the transactions of a typical production order processing system that works against this schema. Its transactions are a specified mixture of five read-only and update-intensive business transactions, namely NewOrder, Payment, OrderStatus, Delivery, and StockLevel. CH2, like CH, uses TPC-C's transaction mix as its operational workload. Also, like TPC-C and CH, CH2's performance reporting focuses on just one of the five transactions from TPC-C, namely NewOrder. NewOrder represents a business transaction that enters a new order with multiple nested orderlines into the database. A NewOrder transaction touches most of the TPC-C schema's tables and consists of both read-only queries and updates. Details of all five TPC-C transactions' logic as well as the workload's prescribed mix percentages can be found in the official TPC-C specification [18]. One CH2 change that we made to py-tpcc's out-of-the-box TPC-C operation implementation was that we chopped the Delivery operation, which does delivery processing for 10 orders, into 10 per-order transactions rather than grouping them as a single transaction. This is closer to the implementation prescribed in the specification (see Section 2.7 in [18]) and follows best practices for picking transaction boundaries [22]. Interestingly, py-tpcc also deviates from the TPC-C specification in that the specification states that Delivery should be handled as an asynchronous request. We did not alter that aspect of the py-tpcc implementation, so CH2 is non-compliant (by design) in that respect as it processes the Delivery operations synchronously.

CH2's operational performance is reported, as for CH (and TPC-C), in terms of the throughput and response times for the NewOrder operations.

3.4 Benchmark Queries

For the analytical side of CH2's workload, we started with the 22 queries from the original CH effort, which in turn were inspired by the 22 queries of TPC-H. We then modified the CH queries one-by-one to operate meaningfully against the historical CH2 data with TPC-H-like predicate selectivity characteristics. This included replacing baked-in constants in predicates with controllable randomly generated values and carefully inspecting and modifying query predicates to yield a set of 22 queries that are much more aligned with TPC-H's query characteristics. (The full CH2 query set is at `https://github.com/couchbaselabs/ch2` both in the code and in the Appendix of an extended version of this paper.) The queries are expressed in N1QL (a.k.a. SQL++ [3]), a SQL-like query language that handles nested data. Note that other potential implementations of CH2 will necessarily use different languages for their versions of the benchmark due to a lack of NoSQL query standards.

Figures 2 through 4 show the first three of the 22 CH2 queries (on the right in each figure) along with the corresponding original relational CH queries (on

```
select ol_number,
sum(ol_quantity) as sum_qty,
sum(ol_amount) as sum_amount,
avg(ol_quantity) as avg_qty,
avg(ol_amount) as avg_amount,
count(*) as count_order
from orderline
where ol_delivery_d >
  '2007-01-02 00:00:00.000000'

group by ol_number
order by ol_number
```

```
SELECT ol.ol_number,
SUM(ol.ol_quantity) as sum_qty,
SUM(ol.ol_amount) as sum_amount,
AVG(ol.ol_quantity) as avg_qty,
AVG(ol.ol_amount) as avg_amount,
COUNT(*) as count_order
FROM orders o, o.o_orderline ol
WHERE ol.ol_delivery_d >
  DATE_ADD_STR('[START_DATE]',
    [DAYS],'day')
GROUP BY ol.ol_number
ORDER BY ol.ol_number;
```

Fig. 2. Query 1 – CH (left) vs. CH2 (right)

```
select su_suppkey, su_name,
  n_name, i_id, i_name,
  su_address, su_phone,
  su_comment
from item, supplier, stock, nation,
  region,
  (select s_i_id as m_i_id,
    min(s_quantity) as m_s_quantity
  from stock, supplier, nation,
    region
  where mod((s_w_id * s_i_id),10000)
      = su_suppkey
    and su_nationkey = n_nationkey

    and n_regionkey = r_regionkey

    and r_name like 'Europ%'
  group by s_i_id) m
where i_id = s_i_id
  and mod((s_w_id * s_i_id), 10000)
      = su_suppkey
  and su_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and i_data like '%b'
  and r_name like 'Europ%'
  and i_id=m_i_id
  and s_quantity = m_s_quantity
order by n_name, su_name, i_id
```

```
SELECT su.su_suppkey, su.su_name,
  n.n_name, i.i_id, i.i_name,
  su.su_address, su.su_phone,
  su.su_comment
FROM item i, supplier su, stock s,
  nation n, region r,
  (SELECT s1.s_i_id AS m_i_id,
    MIN(s1.s_quantity) AS m_s_quantity
  FROM stock s1, supplier su1,
    nation n1, region r1
  WHERE s1.s_w_id*s1.s_i_id MOD 10000
        = su1.su_suppkey
    AND su1.su_nationkey
        = n1.n_nationkey
    AND n1.n_regionkey
        = r1.r_regionkey
    AND r1.r_name LIKE '[RNAME]%'
 GROUP BY s1.s_i_id) m
WHERE i.i_id = s.s_i_id
  AND s.s_w_id * s.s_i_id MOD 10000
      = su.su_suppkey
  AND su.su_nationkey = n.n_nationkey
  AND n.n_regionkey = r.r_regionkey
  AND i.i_data LIKE '%[IDATA]'
  AND r.r_name LIKE '[RNAME]%'
  AND i.i_id=m.m_i_id
  AND s.s_quantity = m.m_s_quantity
ORDER BY n.n_name, su.su_name, i.i_id
LIMIT 100;
```

Fig. 3. Query 2 – CH (left) vs. CH2 (right)

the left). The key differences are highlighted. In Query 1 one can see changes due to the nesting of orderlines within orders as well as the replacement of a (meaningless) constant date with a TPC-H-inspired parameterized date range. In Query 2 one can see the replacement of several constants with randomly generated parameters, again akin to those of TPC-H, as well as the addition of a LIMIT clause, also akin to that of the corresponding TPC-H query. Query 3 has similar changes, plus it shows how the order/orderline join from the relational CH query becomes a simple unnesting (of o.o_orderline, where o ranges over orders) in the FROM clause. In a few queries we also added an additional predicate, inspired by their TPC-H cousins, to better align the two queries' business semantics. Similar changes were needed and made for all of the CH SQL queries in order to arrive at CH2's 22-query collection.

CH2's analytical performance is reported, as for CH and TPC-H, in terms of the geometric mean ("power") of the response times of the 22 CH2 queries.

```
select ol_o_id, ol_w_id, ol_d_id,
  sum(ol_amount) as revenue,
  o_entry_d
from customer, neworder,
  orders, orderline
where c_state like 'A%'
  and c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and no_w_id = o_w_id
  and no_d_id = o_d_id
  and no_o_id = o_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and o_entry_d >
    '2007-01-02 00:00:00.000000'
group by ol_o_id, ol_w_id, ol_d_id,
  o_entry_d
order by revenue desc, o_entry_d
```

```
SELECT o.o_id, o.o_w_id, o.o_d_id,
  SUM(ol.ol_amount) AS revenue,
    o.o_entry_d
FROM customer c, neworder no,
  orders o, o.o_orderline ol
WHERE c.c_state LIKE '[CSTATE]%'
  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND no.no_w_id = o.o_w_id
  AND no.no_d_id = o.o_d_id
  AND no.no_o_id = o.o_id
  -- o and ol are implicitly joined
  -- as ol is nested within o

  AND o.o_entry_d < '[O_YEAR]-[O_MONTH]'
    || '-[O_DAY] 00:00:00.000000'
GROUP BY o.o_id, o.o_w_id, o.o_d_id,
  o.o_entry_d
ORDER BY revenue DESC, o.o_entry_d
LIMIT 10;
```

Fig. 4. Query 3 – CH (left) vs. CH2 (right)

4 A First Target: Couchbase Server

To illustrate the utility of CH2, we have tested the HOAP capabilities of Couchbase Server 7.0, a scalable document database system [2]. With a shared-nothing architecture, it exposes a fast key-value store with a managed cache for sub-millisecond data operations, secondary indexing for fast querying, and (as we will see) two complementary query engines [7] for executing declarative SQL-like N1QL[2] queries.

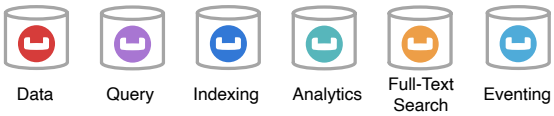


Fig. 5. Major Couchbase Server Components

Figure 5 lists Couchbase Server's major components. Architecturally, the system is organized as a set of services that are deployed and managed as a whole on a Couchbase Server cluster. Nodes can be added or removed through a rebalance process that redistributes the data across all nodes. This can increase or decrease the CPU, memory, disk, or network capacity of a cluster. The ability to dynamically scale the cluster and map services to sets of nodes is referred to as Multi-Dimensional Scaling (MDS). Figure 6 show how MDS might enable a cluster to have 3 nodes for its Data Service, 2 shared by its Index and Full-Text Search Services, 1 for the Query Service, and 2 for the Analytics Service.

A key aspect of Couchbase Server's architecture is how data changes are communicated across services. An internal Database Change Protocol (DCP) notifies all services of changes to documents managed by the Data Service.

The Data Service lays the foundation for document management. It provides caching, persistence, and inter-node replication. The document data model is

[2] N1QL is short for Non-1NF Query Language.

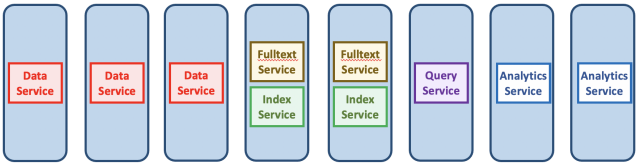


Fig. 6. Multi-Dimensional Scaling (MDS)

JSON, and documents live in containers called buckets. A bucket contains related documents, akin to a database in a relational DBMS. There is no explicitly defined schema, so the "schema" for documents is based on the application code and captured in the structure of each stored document. Developers can add new objects and properties at any time by deploying new application code that stores new JSON data without having to also make and deploy corresponding changes to a static schema. As of Couchbase Server 7.0, documents within a bucket reside in collections (similar to RDBMS tables) that can be grouped together logically using scopes (similar to RDBMS schemas).

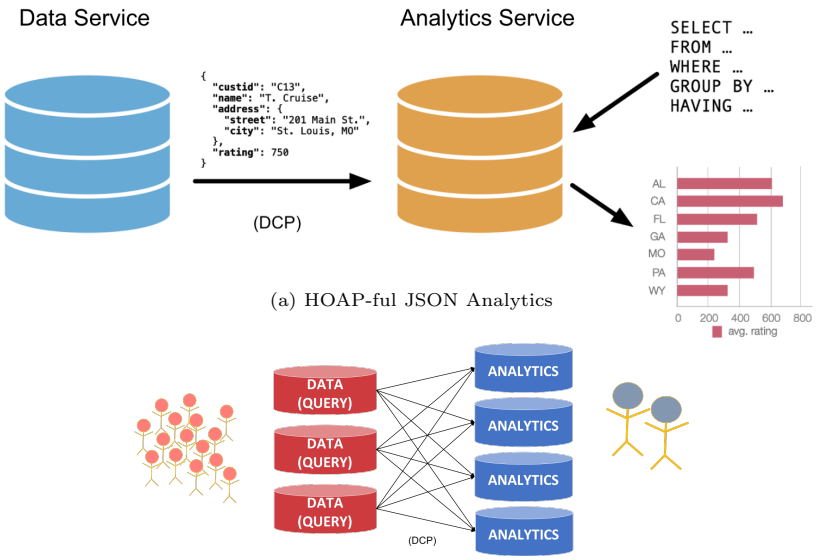
The Indexing, Full-Text Search, and Query Services coordinate via DCP to provide document database management functionality that supports low-latency queries and updates for JSON documents. The Indexing Service provides global secondary indexing for the data managed by the Data Service, and the Full-Text Search service adds richer text indexing and search. The Query Service ties this all together by exposing Couchbase Server's database functionality through N1QL, a declarative, SQL-based query language that relaxes the rigid 1NF and strongly-typed schema demands of the relational SQL standard. As of Couchbase Server 7.0, N1QL supports SQL-style, multi-document, multi-statement transactions using a combination of optimistic and pessimistic concurrency control. A series of N1QL DML statements can be grouped into an atomic transaction whose effects span the Query, Indexing, and Data Services.

The Analytics Service complements the Query Service by supporting more expensive ad-hoc analytical queries (e.g., large joins and aggregations) over JSON document collections. Figures 7(a) and 7(b) show its role in Couchbase Server. The Data and Query Services provide low-latency key-value-based and query-based access to their data. Their design point is operational; they support many users making well-defined, programmatic requests that tend to be small and inexpensive. In contrast, the Analytics Service focuses on ad hoc and analytical requests; it has fewer users posing larger, more expensive N1QL queries against a real-time shadow copy of the same JSON data. The Query service has a largely point-to-point/RPC-based query execution model; the Analytics Service employs partitioned parallelism under the hood, using parallel query processing to bring all of the resources of the Analytics nodes to bear on each query [7].

The Eventing Service provides an Event-Condition-Action based framework that application developers can use to respond to data changes in real time.

So what about HOAP? As Figures 7(a) and 7(b) try to indicate, operational data in Couchbase Server is available for analysis as soon as it is created; analysts always see fresh application data thanks to DCP. They can immediately pose questions about operational data, in its natural data model, reducing the time

to insight from days or hours to seconds. There are several differences between this approach and HTAP in the relational world. One is scale: The Analytics Service can be scaled out horizontally on a shared-nothing cluster [7], and it can be scaled independently (Figure 6). It maintains a real-time shadow copy of operational data that an enterprise wants to analyze; the copy is because Analytics is deployed on its own nodes with their own storage to provide performance isolation for the operational and analytical workloads. Another difference relates to technology: Couchbase Analytics is not an in-memory solution. It is designed to handle a large volume of NoSQL documents – documents whose individual value and access frequency would not warrant the cost of a memory-resident solution, but whose aggregated content can still be invaluable for decision-making.



(a) HOAP-ful JSON Analytics

(b) Scalable Hybrid Architecture

Fig. 7. Couchbase Analytics Service in Couchbase Server

5 Benchmark Results

In this section we present preliminary results from implementing and running CH2 on an AWS cluster running Couchbase Server 7.0.

5.1 Benchmark Implementation

The CH2 data was stored in a scope called *ch2* in a bucket called *bench* in the Data Service. Data Definition 1.1 shows the N1QL statements for creating the CH2 benchmark's collections. The *bench* bucket was the target for the operational queries and updates and was indexed to support them. Data Definition 1.2 shows the N1QL statements used to create these indexes. For the analytical

workload, shadow collections were created in the Analytics Service for each of the aforementioned Data Service collections. Data Definition 1.3 shows the N1QL statements to create these shadow collections in Analytics. Hybrid performance trends were the main focus of this exercise, not absolute performance, so little effort was made to tune the indexing choices or queries. In fact, the Analytics Service queries were run without any indexing for these initial experiments.

```
CREATE SCOPE bench.ch2;
CREATE COLLECTION bench.ch2.customer;
CREATE COLLECTION bench.ch2.district;
... (etc.)
```

Data Definition 1.1. Query Service Collection DDL

```
CREATE INDEX cu_w_id_d_id_last
          ON bench.ch2.customer(c_w_id, c_d_id, c_last) USING GSI;
CREATE INDEX di_id_w_id
          ON bench.ch2.district(d_id, d_w_id) USING GSI;
CREATE INDEX no_o_id_d_id_w_id
          ON bench.ch2.neworder(no_o_id, no_d_id, no_w_id) USING GSI;
CREATE INDEX or_id_d_id_w_id_c_id
          ON bench.ch2.orders(o_id, o_d_id, o_w_id, o_c_id) USING GSI;
CREATE INDEX or_w_id_d_id_c_id
          ON bench.ch2.orders(o_w_id, o_d_id, o_c_id) USING GSI;
CREATE INDEX wh_id
          ON bench.ch2.warehouse(w_id) USING GSI;
```

Data Definition 1.2. Query Service Index DDL

```
ALTER COLLECTION bench.ch2.customer ENABLE ANALYTICS;
ALTER COLLECTION bench.ch2.district ENABLE ANALYTICS;
... (etc.)
```

Data Definition 1.3. Analytics Service Collection DDL

As mentioned earlier, to drive the benchmark's mixed workload we started with the py-tpcc benchmarking package from CMU, the same package recently used by MongoDB [8], modified its data generator following the earlier description, and added a CH2 driver for Couchbase Server to meet our modified schema and mixed workload requirements.[3] Each operational or analytical user is simulated by a thread running on a client node of the cluster under test. Each thread consistently sends query requests to the system. Up to 128 threads send TPC-C operations to the Query Service, with 0 or 1 threads sending analytical queries to the Analytics Service. These thread counts simulate a typical business model with many front-end users but just a few data analysts (in this case one).

[3] The software artifacts associated with this paper's benchmark can be found at `https://github.com/couchbaselabs/ch2`

5.2 Benchmark Configuration(s)

Our goal here is to use CH2 to explore several key characteristics of HOAP-ful platforms for NoSQL, including (1) their effectiveness at providing performance isolation between the OLTP and OLAP components of a mixed workload, and (2) the scalability of their architectures when faced with either a need to support more operational users or a need to perform faster analytics.

To this end, we ran our CH2 benchmark implementation on a cluster consisting of 5-17 nodes that we configured in the AWS cloud. Hardware-wise, the cluster was comprised of 4-16 m5d.4xlarge instances, each with 16 vCPUs, 64GB of memory, 2 300GB NVMe SSDs, and up to 10 Gbps of network bandwidth, forming the Couchbase Server cluster, plus one m5d.24xlarge instance with 96 vCPUs, 384GB of memory, 2 900GB NVMe SSDs, and up to 25 Gbps of network bandwidth that was used to run the client workload driver. The AWS nodes running a Data, Index, and Query Service combination utilized one of the SSD drives for data and the other for indexes, while the nodes running the Analytics Service utilized both drives uniformly for enhanced query parallelism.

We configured Couchbase Server clusters in five different ways, as shown in Figure 8. In the first configuration, the 4 Query + 4 Analytics case (4Q+4A), 4 nodes are configured to have the Data Service, Index Service, and Query Service, forming the operational subcluster. The other 4 nodes have the Analytics Service, forming the analytical subcluster. In each subcluster the CH2 data is hash-partitioned across the subcluster's nodes. Operational N1QL requests from the CH2 driver are directed to the Query Service's API endpoints, while Analytical N1QL requests are directed to the Analytics Service's endpoint. In the second configuration, the 8 Query + 4 Analytics case (8Q+4A), the operational subcluster is doubled. Symmetrically, in the third cluster configuration, the 4 Query + 8 Analytics case (4Q+8A), the analytical subcluster is instead doubled relative to its initial size. In the fourth configuration, the 8 Query + 8 Analytics case (8Q+8A), both subclusters are twice their initial size. Finally, for "extra credit", we also included a fifth configuration, the 8 Query + 2 Analytics case (8Q+2A), to test a configuration that has a scaled-up operational subcluster paired with a scaled-down analytical subcluster. Note that the Analytics Service is always given a set of nodes to itself, in all configurations, in order to provide performance isolation for the operational workload.

Data-wise, the operational data resides in the *ch2* scope of the *bench* bucket in the Data Service (in JSON document form). For these experiments we generated a 1,000 warehouse instance of CH2. The cardinalities of the CH2 collections are thus consistent with the example numbers shown earlier in Table 1.

5.3 Initial Benchmark Results

Our initial goal is to explore Couchbase Server's behavior regarding operational performance, analytical performance, and performance isolation for CH2's mixed workload. We first ran one full loop of the 22 CH2 analytical queries in isolation (i.e., with no operational clients) and measured the time required for each

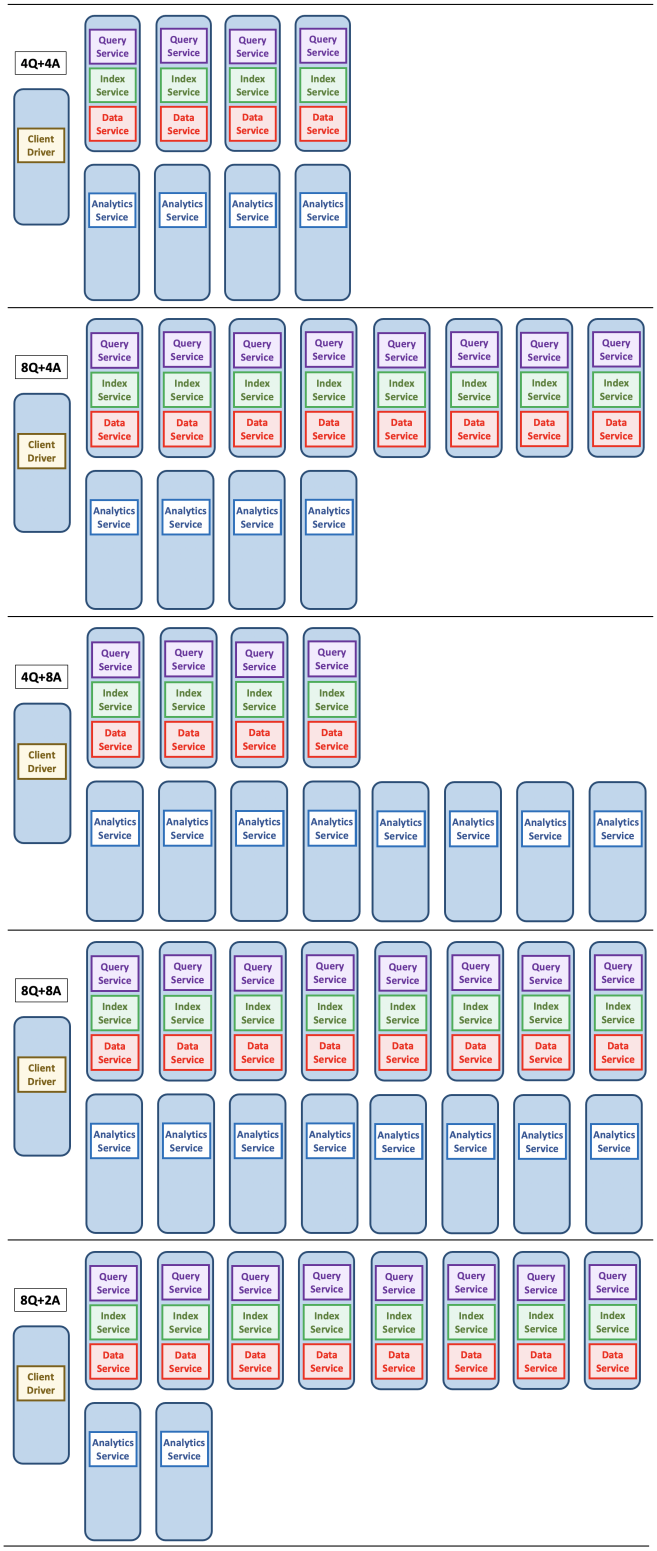
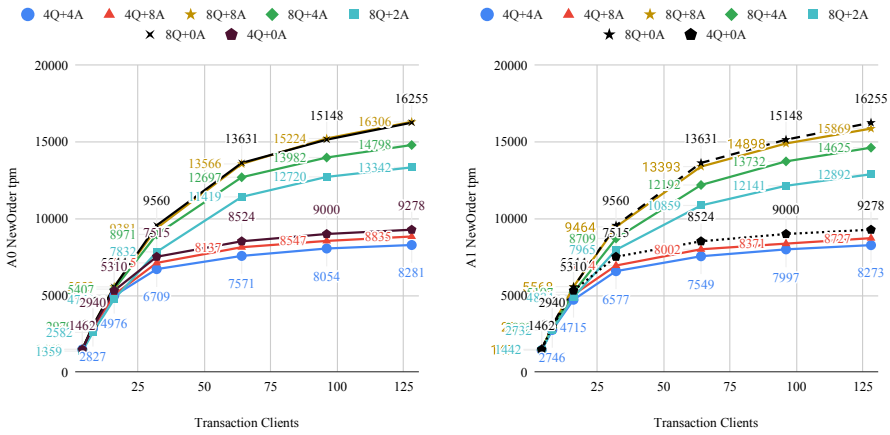


Fig. 8. Five Couchbase Cluster Configurations

configuration. Then, for the purely operational CH2 runs (without analytical clients), we ran the operational workload for this measured duration (rounded up to the nearest minute). For the mixed workload CH2 runs, the operational and analytical clients were run concurrently until the client running the 22 CH2 analytical queries completed one full loop.



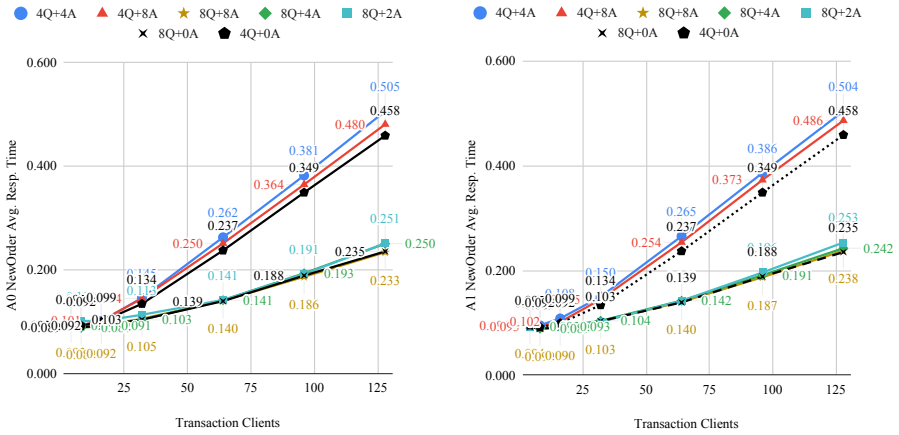
(a) Tx Clients vs NewOrder tpM (A0)

(b) Tx Clients vs NewOrder tpM (A1)

Fig. 9. Tx Clients vs. NewOrder tpM, without (A0) and with (A1) analytical queries, for different cluster configurations

Figure 9 shows the operational CH2 performance, in NewOrder transactions per minute, as the number of operational client threads is varied from 4 to 128 for the five different cluster configurations. Figure 9(a) shows the performance results without a concurrent analytical workload – i.e., when zero analytical query threads are running (A0). Figure 9(b) shows the performance results in the presence of a concurrent analytical workload – i.e., when there is one analytical query thread running (A1). In addition, Figure 9(a) shows results for one extra cluster configuration, labeled 8Q+0A – a configuration with an 8-node operational subcluster but *no* analytical subcluster. Recall that the Analytics service maintains its own real-time shadow copy of operational data for analysis purposes. In the 8Q+0A configuration, with no Analytics nodes, *no* shadow collections are being maintained, while in the five other 8Q (A0) configurations, with Analytics nodes, shadow collections are being kept up-to-date (via DCP) in real-time even though there are no concurrent analytical queries making use of them. Figures 10(a) and (b) show the response time results for NewOrder transactions corresponding to Figure 9's throughput results.

There is a great deal of information packed into Figure 9, so let us proceed to unpack it and see what we can glean from the results shown there. First of all, for all configurations, with or without concurrent analytical queries, we can see that the system delivers textbook performance – i.e., the curve shapes are as expected for throughput under a closed workload, first increasing linearly and then reaching a plateau when the system's resources are saturated – so the



(a) Tx Clients vs NewOrder RT (A0)

(b) Tx Clients vs NewOrder RT (A1)

Fig. 10. Tx Clients vs. NewOrder Response Time, without (A0) and with (A1) analytical queries, for different cluster configurations

system is *well-behaved* and exhibits no thrashing. A second observation is that if we compare the system's A0 vs. A1 throughputs in Figures 9(a) and (b), we see *effective performance isolation*. The throughput trends and levels achieved for a given configuration in Figure 9(b), where the CH2 analytical query workload is constantly running, are only slightly less those in Figure 9(a) where there are no concurrent queries. These observations are borne out by the response time results in Figure 10 as well. The response times are initially flat and then slowly become linearly proportional to the client thread count once the system becomes saturated – textbook behavior – and the corresponding response times in Figures 9(a) and (b) are essentially pairwise identical.

Continuing with our analysis, let us now examine the throughput results (Figures 9(a) and (b)) *within* a given operational subcluster size (Q) but with different analytical subcluster sizes (A). Let us first look at the 4Q subcluster configurations, 4Q+4A and 4Q+8A. What we see is that their operational performance is essentially the same, i.e., the system's 4Q operational performance is unaffected by the analytical subcluster size, so feeding the real-time operational updates to the analytical subcluster is not a problem here. Next let us look at the 8Q subcluster configurations. There we see that the performance for 8A+0A and 8Q+8A are also identical to one another, so having an 8-node analytical subcluster to feed does not place a burden on the 8Q operational performance. In addition, we see that the throughput delivered by these two 8Q configurations is twice the throughput of the 4Q configurations – i.e., the system appears capable of delivering *linear transactional scaleup* – so if you require twice the operational throughput, and you double your investment in hardware, your requirement can be met. However, if we turn now to the 8Q+4A and 8Q+2A results, we do observe some throughput degradation on the 8Q operational side when the analytical subcluster is "undersized" in comparison. Mysteriously, however, if we

examine the response time results in Figure 10, we do not observe this same effect – i.e., we do not see a noteworthy degradation in the NewOrder response times due to smaller analytical subclusters. Thus, if you need your operational response times to be twice as fast, you can double your investment in operational subcluster hardware to meet your performance requirement.[4]

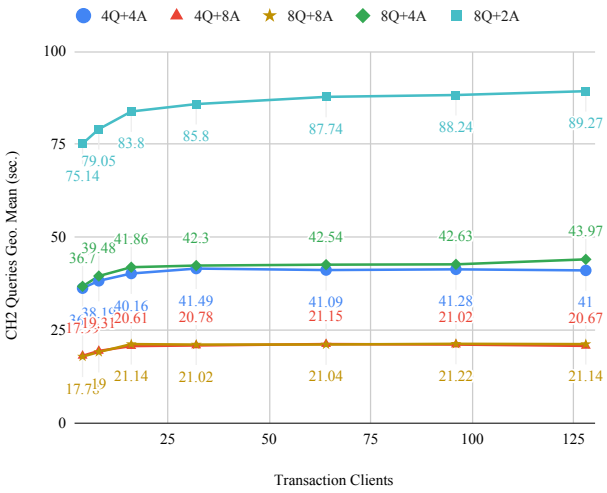


Fig. 11. Tx Clients vs. Power (Geometric Mean of Analytical Query Times)

Let us now turn our attention to what the CH2 benchmark can reveal about Couchbase Server's analytical query performance. Figure 11 shows the geometric mean of the 22 queries' average response times (query power) for the five different cluster configurations versus the number of operational client threads. There are two key take-aways that are evident in this figure. First, we see the flip side of Couchbase Server's largely successful delivery of operational performance isolation (HOAP) – the analytical workload's performance is not overly affected as the number of concurrent operational client threads is increased on the x-axis. Queries initially slow down somewhat as the operational subcluster begins generating more updates to be ingested, i.e., as the number of clients grows, because ingestion "steals" some of the analytical subcluster's query processing capacity, but query performance then levels out and is unaffected once the operational subcluster is saturated. Second, we see the very successful delivery of *linear query speedup* – i.e., when the number of nodes given to the Analytics service is doubled, the CH2 query execution times are essentially cut in half. Thus, to have your analytical queries run twice as fast, you can simply double your investment in the analytical subcluster's hardware.

[4] The reason why we see some NewOrder throughput impact for the smaller 8Q configurations, but apparently without a corresponding NewOrder response time impact, is currently a bit of a performance mystery that running CH2 has revealed. We have several theories and we are currently investigating this behavior in order to further enhance Couchbase Server's performance isolation and scaling characteristics.

6 Conclusion

Systems that provide hybrid workload support (i.e., HTAP or HOAP) first arose in the relational world, where they are often linked to server technology trends such as columnar storage and memory-rich, many-core, scale-up servers. Our focus here is on hybrid NoSQL platforms. We introduced CH2, a benchmark for evaluating hybrid platforms in the document database world. Like CH, its inspiration, the CH2 benchmark borrows from and extends both TPC-C and TPC-H. Differences from CH include a document-oriented schema, a data generation scheme that provides a TPC-H-like history for meaningful analytics, and a "do over" of the CH queries that is more aligned with TPC-H. We detailed the shortcomings that we found in CH, described the design of CH2, and shared preliminary results from running CH2 against Couchbase Server 7.0. These initial results provide insight into the performance isolation and horizontal scalability of Couchbase Server 7.0 as well as showing the value of CH2 for evaluating HOAP-ful NoSQL platforms.

Acknowledgments

The authors wish to thank the Couchbase Query Service team, especially Sitaram Vemulapalli and Kamini Jagtiani, for assisting us with the new 7.0 N1QL transaction support, and Michael Blow and Ian Maxon from the Couchbase Analytics Service team, for invaluable assistance in setting up the AWS clusters used for the experiments.

References

1. 451 Research: Hybrid processing enables new use cases (business impact brief) (2018), `https://www.intersystems.com/isc-resources/wp-content/uploads/sites/24/Hybrid_Processing_Enables_New_Use_Cases-451Research.pdf` [Online; accessed 19-October-2020]
2. Borkar, D., et al.: Have your data and query it too: From key-value caching to big data management. In: Proc. ACM SIGMOD Conf. pp. 239–251. ACM (2016)
3. Chamberlin, D.: SQL++ for SQL Users: A Tutorial. Couchbase, Inc. (Available via Amazon.com.) (2018)
4. Cole, R.L., et al.: The mixed workload CH-benCHmark. In: Proc. Fourth Int'l. Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011. p. 8. ACM (2011)
5. Cooper, B.F., et al.: Benchmarking cloud serving systems with YCSB. In: Proc. 1st ACM Symp. on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010. pp. 143–154. ACM (2010)
6. Gray, J. (ed.): The Benchmark Handbook for Database and Transaction Systems (1st Edition). Morgan Kaufmann (1991)
7. Hubail, M.A., et al.: Couchbase Analytics: NoETL for scalable NoSQL data analysis. PVLDB **12**(12), 2275–2286 (2019)
8. Kamsky, A.: Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB. PVLDB **12**(12), 2254–2262 (2019)

9. Kemper, A., Neumann, T.: Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: 2011 IEEE 27th Int'l. Conf. on Data Engineering. pp. 195–206 (2011)

10. Lahiri, T., et al.: Oracle database in-memory: A dual format in-memory database. In: 2015 IEEE 31st Int'l. Conf. on Data Engineering. pp. 1253–1258 (2015)

11. Larson, P., et al.: Real-time analytical processing with SQL server. PVLDB **8**(12), 1740–1751 (2015)

12. May, N., Böhm, A., Lehner, W.: SAP HANA - the evolution of an in-memory DBMS from pure OLAP processing towards mixed workloads. In: Proc. BTW 2017, 17. Fachtagung des GI-Fachber. DBIS, März 2017, Stuttgart, Germany (2017)

13. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 4th Edition. Springer (2020)

14. Pirzadeh, P., Carey, M., Westmann, T.: BigFUN: A performance study of big data management system functionality. In: 2015 IEEE Int'l. Conf. on Big Data. pp. 507–514 (2015)

15. Pirzadeh, P., Carey, M., Westmann, T.: A performance study of big data analytics platforms. In: 2017 IEEE Int'l. Conf. on Big Data. pp. 2911–2920 (2017)

16. Pöss, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. SIGMOD Record **29**(4), 64–71 (2000)

17. Pöss, M., et al.: TPC-DS, taking decision support benchmarking to the next level. In: Proc. ACM SIGMOD Conf. pp. 582–587. ACM (2002)

18. Raab, F.: TPC-C - The standard benchmark for online transaction processing (OLTP). In: Gray, J. (ed.) The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann (1993)

19. Raman, V., et al.: DB2 with BLU acceleration: So much more than just a column store. PVLDB **6**(11), 1080–1091 (2013)

20. Raza, A., et al.: Adaptive HTAP through elastic resource scheduling. In: Proc. ACM SIGMOD Conf. pp. 2043–2054. ACM (2020)

21. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, Upper Saddle River, NJ (2013)

22. Shasha, D.E.: Database Tuning – A Principled Approach. Prentice-Hall (1992)

23. Tian, Y., Carey, M., Maxon, I.: Benchmarking HOAP for scalable document data management: A first step. In: 2020 IEEE Int'l. Conf. on Big Data. pp. 2833–2842 (2020)

24. Wikipedia contributors: Hybrid transactional/analytical processing — Wikipedia, the free encyclopedia (2020), `https://en.wikipedia.org/w/index.php?title=Hybrid_transactional/analytical_processing&oldid=981969658`, [Online; accessed 19-October-2020]

A Queries

In this appendix we provide all 22 pairs of queries — highlighting the salient differences — with the original CH query on the left and the modified CH2 query on the right.

Query 1

```
select ol_number,
sum(ol_quantity) as sum_qty,
sum(ol_amount) as sum_amount,
avg(ol_quantity) as avg_qty,
avg(ol_amount) as avg_amount,
count(*) as count_order
from orderline
where ol_delivery_d >
  '2007-01-02 00:00:00.000000'

group by ol_number
order by ol_number
```

```
SELECT ol.ol_number,
SUM(ol.ol_quantity) as sum_qty,
SUM(ol.ol_amount) as sum_amount,
AVG(ol.ol_quantity) as avg_qty,
AVG(ol.ol_amount) as avg_amount,
COUNT(*) as count_order
FROM orders o, o.o_orderline ol
WHERE ol.ol_delivery_d >
  DATE_ADD_STR('[START_DATE]',
    [DAYS],'day')
GROUP BY ol.ol_number
ORDER BY ol.ol_number;
```

Query 2

```
select su_suppkey, su_name,
  n_name, i_id, i_name,
  su_address, su_phone,
  su_comment
from item, supplier, stock, nation,
  region,
  (select s_i_id as m_i_id,
    min(s_quantity) as m_s_quantity
  from stock, supplier, nation,
    region
  where mod((s_w_id * s_i_id),10000)
      = su_suppkey
    and su_nationkey = n_nationkey

    and n_regionkey = r_regionkey

    and r_name like 'Europ%'
  group by s_i_id) m
where i_id = s_i_id
  and mod((s_w_id * s_i_id), 10000)
      = su_suppkey
  and su_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and i_data like '%b'
  and r_name like 'Europ%'
  and i_id=m_i_id
  and s_quantity = m_s_quantity
order by n_name, su_name, i_id
```

```
SELECT su.su_suppkey, su.su_name,
  n.n_name, i.i_id, i.i_name,
  su.su_address, su.su_phone,
  su.su_comment
FROM item i, supplier su, stock s,
  nation n, region r,
  (SELECT s1.s_i_id AS m_i_id,
    min(s1.s_quantity) AS m_s_quantity
  FROM stock s1, supplier su1,
    nation n1, region r1
  WHERE s1.s_w_id*s1.s_i_id MOD 10000
      = su1.su_suppkey
    AND su1.su_nationkey
      = n1.n_nationkey
    AND n1.n_regionkey
      = r1.r_regionkey
    AND r1.r_name LIKE '[RNAME]%'
 GROUP BY s1.s_i_id) m
WHERE i.i_id = s.s_i_id
  AND s.s_w_id * s.s_i_id MOD 10000
     = su.su_suppkey
  AND su.su_nationkey = n.n_nationkey
  AND n.n_regionkey = r.r_regionkey
  AND i.i_data LIKE '%[IDATA]'
  AND r.r_name LIKE '[RNAME]%'
  AND i.i_id=m.m_i_id
  AND s.s_quantity = m.m_s_quantity
ORDER BY n.n_name, su.su_name, i.i_id
LIMIT 100;
```

Query 3

```
select ol_o_id, ol_w_id, ol_d_id,
  sum(ol_amount) as revenue,
  o_entry_d
from customer, neworder,
  orders, orderline
where c_state like 'A%'
  and c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and no_w_id = o_w_id
  and no_d_id = o_d_id
  and no_o_id = o_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and o_entry_d >
    '2007-01-02 00:00:00.000000'
group by ol_o_id, ol_w_id, ol_d_id,
  o_entry_d
order by revenue desc, o_entry_d
```

```
SELECT o.o_id, o.o_w_id, o.o_d_id,
  SUM(ol.ol_amount) AS revenue,
    o.o_entry_d
FROM customer c, neworder no,
  orders o, o.o_orderline ol
WHERE c.c_state LIKE '[CSTATE]%'
  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND no.no_w_id = o.o_w_id
  AND no.no_d_id = o.o_d_id
  AND no.no_o_id = o.o_id
  -- o and ol are implicitly joined
  -- as ol is nested within o

  AND o.o_entry_d < '[O_YEAR]-[O_MONTH]'
    || '-[O_DAY] 00:00:00.000000'
GROUP BY o.o_id, o.o_w_id, o.o_d_id,
  o.o_entry_d
ORDER BY revenue DESC, o.o_entry_d
LIMIT 10;
```

Query 4

```
select o_ol_cnt,
  count(*) as order_count
from orders
where o_entry_d >=
  '2007-01-02 00:00:00.000000'

and o_entry_d <
  '2012-01-0200:00:00.000000'

and exists (select *
 from orderline
 where o_id = ol_o_id
 and o_w_id = ol_w_id
 and o_d_id = ol_d_id
 and ol_delivery_d >=
   o_entry_d)
group by o_ol_cnt
order by o_ol_cnt
```

```
SELECT o.o_ol_cnt,
  COUNT(*) AS order_count
FROM orders o
WHERE o.o_entry_d >=
  '[O_YEAR]-[O_MONTH]-[O_DAY]'
  || ' 00:00:00.000000'
AND o.o_entry_d <
  DATE_ADD_STR('[O_YEAR]-[O_MONTH]-'
  || '[O_DAY] 00:00:00.000000', 3, 'month')
AND EXISTS (SELECT VALUE 1
  FROM o.o_orderline ol



 WHERE ol.ol_delivery_d >=
   DATE_ADD_STR(o.o_entry_d, 1, 'week'))
GROUP BY o.o_ol_cnt
ORDER BY o.o_ol_cnt;
```

Query 5

```
select n_name,
  sum(ol_amount) as revenue
from customer, orders,
  orderline, stock,
  supplier, nation, region
where c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and ol_o_id = o_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_w_id = s_w_id
  and ol_i_id = s_i_id
  and mod((s_w_id * s_i_id),10000)
      = su_suppkey

  and ascii(substr(c_state,1,1))
      = su_nationkey
  and su_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'Europe'
  and o_entry_d >=
    '2007-01-02 00:00:00.000000'




group by n_name
order by revenue desc
```

```
SELECT n.n_name,
  SUM(ol.ol_amount) AS revenue
FROM customer c, orders o,
  o.o_orderline ol, stock s,
  supplier su, nation n, region r
WHERE c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id




  AND o.o_w_id = s.s_w_id
  AND ol.ol_i_id = s.s_i_id
  AND s.s_w_id * s.s_i_id MOD 10000 =
    su.su_suppkey
  AND
    string_to_codepoint(c.c_state)[0]
    = su.su_nationkey
  AND su.su_nationkey = n.n_nationkey
  AND n.n_regionkey = r.r_regionkey
  AND r.r_name = '[RNAME]'
  AND o.o_entry_d >=
    '[O_YEAR]-[O_MONTH]-[O_DAY]'
    || ' 00:00:00.000000'
  AND o.o_entry_d <
    DATE_ADD_STR('[O_YEAR]'
    || '-[O_MONTH]-[O_DAY]'
    || '00:00:00.000000', 1, 'year')
GROUP BY n.n_name
ORDER BY revenue DESC;
```

Query 6

```
select sum(ol_amount) as revenue
from orderline
where ol_delivery_d >=
    '1999-01-01 00:00:00.000000'

  and ol_delivery_d <
    '2020-01-01 00:00:00.000000'


  and ol_quantity between 1 and 100000
```

```
SELECT SUM(ol.ol_amount) AS revenue
FROM orders o, o.o_orderline ol
WHERE ol.ol_delivery_d >=
   '[OL_YEAR]-[OL_MONTH]-[OL_DAY]'
   || ' 00:00:00.000000'
 AND ol.ol_delivery_d <
   DATE_ADD_STR('[OL_YEAR]-[OL_MONTH]'
     ||'-[OL_DAY] 00:00:00.000000', 1,
     'year')
  AND ol.ol_amount > [OL_AMOUNT];
```

Query 7

```
select su_nationkey as supp_nation,
  substr(c_state,1,1)
    as cust_nation,
  extract(year from o_entry_d)
    as l_year,
  sum(ol_amount) as revenue
from supplier, stock,
  orderline, orders,
  customer, nation n1, nation n2
where ol_supply_w_id = s_w_id
  and ol_i_id = s_i_id
  and mod((s_w_id * s_i_id), 10000) =
    su_suppkey
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and su_nationkey = n1.n_nationkey

  and ascii(substr(c_state,1,1)) =
    n2.n_nationkey
  and ((n1.n_name = 'Germany' and
       n2.n_name = 'Cambodia') or
      (n1.n_name = 'Cambodia' and
       n2.n_name = 'Germany'))
  and ol_delivery_d between
    '2007-01-02 00:00:00.000000'
    and '2012-01-02 00:00:00.000000'



group by su_nationkey,
  substr(c_state,1,1),
  extract(year from o_entry_d)
order by su_nationkey,
  cust_nation, l_year
```

```
SELECT su.su_nationkey AS supp_nation,
  SUBSTR1(c.c_state,1,1)
    AS cust_nation,
  DATE_PART_STR(o.o_entry_d, 'year')
   AS l_year,
 SUM(ol.ol_amount) AS revenue
FROM supplier su, stock s,
  orders o, o.o_orderline ol,
  customer c, nation n1, nation n2
WHERE ol.ol_supply_w_id = s.s_w_id
  AND ol.ol_i_id = s.s_i_id
  AND s.s_w_id * s.s_i_id MOD 10000 =
    su.su_suppkey



  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND su.su_nationkey = n1.n_nationkey
  AND
    string_to_codepoint(c.c_state)[0]
    = n2.n_nationkey
  AND ((n1.n_name = '[NNAMEA]' AND
       n2.n_name = '[NNAMEB]' OR)
      (n1.n_name = '[NNAMEB]' AND
       n2.n_name = '[NNAMEA]'))
  AND ol.ol_delivery_d BETWEEN
    '[START_OL_YEAR]-[START_OL_MONTH]'
    || '-[START_OL_DAY] '
    || '00:00:00.000000' AND
    '[END_OL_YEAR]-[END_OL_MONTH]'
    || '-[END_OL_DAY] 00:00:00.000000'
GROUP BY su.su_nationkey,
  SUBSTR1(c.c_state,1,1),
  DATE_PART_STR(o.o_entry_d, 'year')
ORDER BY su.su_nationkey,
  cust_nation, l_year;
```

Query 8

```
select extract(year from o_entry_d)
    as l_year,
  sum(case
    when n2.n_name = 'Germany'
    then ol_amount
    else 0 end)/sum(ol_amount)
    as mkt_share
from item, supplier, stock,
  orderline, orders,
  customer, nation n1, nation n2,
  region
where i_id = s_i_id
  and ol_i_id = s_i_id
  and ol_supply_w_id = s_w_id
  and mod((s_w_id * s_i_id),10000) =
    su_suppkey
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and n1.n_nationkey =
    ascii(substr(c_state,1,1))
  and n1.n_regionkey = r_regionkey
  and ol_i_id < 1000
  and r_name = 'Europe'
  and su_nationkey = n2.n_nationkey
  and o_entry_d between
    '2007-01-02 00:00:00.000000'
    and '2012-01-02 00:00:00.000000'



  and i_data like '%b'
  and i_id = ol_i_id
group by extract(year from o_entry_d)

order by l_year
```

```
SELECT DATE_PART_STR(o.o_entry_d,
    'year') AS l_year,
  SUM(CASE
    WHEN n2.n_name = '[NNAME]'
    THEN ol.ol_amount
    ELSE 0 END) / SUM(ol.ol_amount)
    AS mkt_share
FROM item i, supplier su, stock s,
  orders o, o.o_orderline ol,
  customer c, nation n1, nation n2,
  region r
WHERE i.i_id = s.s_i_id
  AND ol.ol_i_id = s.s_i_id
  AND ol.ol_supply_w_id = s.s_w_id
  AND s.s_w_id * s.s_i_id MOD 10000 =
    su.su_suppkey



  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND n1.n_nationkey =
    string_to_codepoint(c.c_state)[0]
  AND n1.n_regionkey = r.r_regionkey
  AND ol.ol_i_id < [OLIID]
  AND r.r_name = '[RNAME]'
  AND su.su_nationkey = n2.n_nationkey
  AND o.o_entry_d BETWEEN
    '[START_O_YEAR]-[START_O_MONTH]'
    || '-[START_O_DAY]'
    || ' 00:00:00.000000' AND
    '[END_O_YEAR]-[END_O_MONTH]'
    ||'-[END_O_DAY] 00:00:00.000000'
  AND i.i_data LIKE '%[IDATA]'
  AND i.i_id = ol.ol_i_id
GROUP BY DATE_PART_STR(o.o_entry_d,
  'year')
ORDER BY l_year;
```

Query 9

```
select n_name,
  extract(year from o_entry_d)
    as l_year,
  sum(ol_amount) as sum_profit
from item, stock, supplier,
  orderline, orders,
  nation
where ol_i_id = s_i_id
  and ol_supply_w_id = s_w_id
  and mod((s_w_id * s_i_id), 10000) =
    su_suppkey
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and ol_i_id = i_id
  and su_nationkey = n_nationkey
  and i_data like '%BB'
group by n_name,
  extract(year from o_entry_d)
order by n_name, l_year desc
```

```
SELECT n.n_name,
  DATE_PART_STR(o.o_entry_d, 'year')
    AS l_year,
  SUM(ol.ol_amount) AS SUM_profit
FROM item i, stock s, supplier su,
  orders o, o.o_orderline ol,
  nation n
WHERE ol.ol_i_id = s.s_i_id
  AND ol.ol_supply_w_id = s.s_w_id
  AND s.s_w_id * s.s_i_id MOD 10000 =
    su.su_suppkey



  AND ol.ol_i_id = i.i_id
  AND su.su_nationkey = n.n_nationkey
  AND i.i_data LIKE '%[IDATAA][IDATAB]'
GROUP BY n.n_name,
  DATE_PART_STR(o.o_entry_d, 'year')
ORDER BY n.n_name, l_year DESC;
```

Query 10

```
select c_id, c_last,
  sum(ol_amount) as revenue,
  c_city, c_phone, n_name
from customer,
  orders, orderline,
  nation
where c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and o_entry_d >=
    '2007-01-02 00:00:00.000000'

  and o_entry_d <=
    ol_delivery_d


  and n_nationkey =
    ascii(substr(c_state,1,1))
group by c_id, c_last, c_city,
  c_phone, n_name
order by revenue desc
```

```
SELECT c.c_id, c.c_last,
  SUM(ol.ol_amount) AS revenue,
  c.c_city, c.c_phone, n.n_name
FROM customer c,
  orders o, o.o_orderline ol,
  nation n
WHERE c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id



  AND o.o_entry_d >=
    '[O_YEAR]-[O_MONTH]-[O_DAY]'
    || ' 00:00:00.000000'
  AND o.o_entry_d <
    DATE_ADD_STR('[O_YEAR]'
    || '-[O_MONTH]-[O_DAY]'
    || ' 00:00:00.000000', 3, 'month')
  AND n.n_nationkey =
    string_to_codepoint(c.c_state)[0]
GROUP BY c.c_id, c.c_last, c.c_city,
  c.c_phone, n.n_name
ORDER BY revenue DESC
LIMIT 20;
```

Query 11

```
select s_i_id,
  sum(s_order_cnt) as ordercount
from stock, supplier, nation
where mod((s_w_id * s_i_id),10000) =
    su_suppkey
  and su_nationkey = n_nationkey
  and n_name = 'Germany'
group by s_i_id
having sum(s_order_cnt) >
  (select sum(s_order_cnt) * 0.005

   from stock, supplier, nation

   where mod((s_w_id * s_i_id),10000)
     = su_suppkey

     and su_nationkey = n_nationkey

     and n_name = 'Germany')
order by ordercount desc
```

```
SELECT s.s_i_id,
  SUM(s.s_order_cnt) AS ordercount
FROM stock s, supplier su, nation n
WHERE s.s_w_id * s.s_i_id MOD 10000
    = su.su_suppkey
  AND su.su_nationkey = n.n_nationkey
  AND n.n_name = '[NNAME]'
GROUP BY s.s_i_id
HAVING SUM(s.s_order_cnt) >
  (SELECT VALUE SUM(s1.s_order_cnt)
     * 0.00[FRACTION]/[SCALE]
   FROM stock s1, supplier su1,
     nation n1
   WHERE
     s1.s_w_id * s1.s_i_id MOD 10000
     = su1.su_suppkey
   AND su1.su_nationkey
     = n1.n_nationkey
   AND n1.n_name = '[NNAME]')[0]
ORDER BY ordercount DESC;
```

Query 12

```
select o_ol_cnt,
  sum(case
    when o_carrier_id = 1
      or o_carrier_id = 2
    then 1
    else 0 end) as high_line_count,
  sum(case
    when o_carrier_id <> 1
      and o_carrier_id <> 2
    then 1
    else 0 end) as low_line_count
from orders, orderline
where o_entry_d <= ol_delivery_d
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
  and ol_delivery_d <
    '2020-01-01 00:00:00.000000'




group by o_ol_cnt
order by o_ol_cnt
```

```
SELECT o.o_ol_cnt,
  SUM (CASE
    WHEN o.o_carrier_id = 1
      OR o.o_carrier_id = 2
    THEN 1
    ELSE 0 END) AS high_line_count,
  SUM (CASE
    WHEN o.o_carrier_id <> 1
      AND o.o_carrier_id <> 2
    THEN 1
    ELSE 0 END) AS low_line_count
FROM orders o, o.o_orderline ol
WHERE o.o_entry_d <= ol.ol_delivery_d



  AND o.ol_delivery_d <
    DATE_ADD_STR('[OL_YEAR]-'
    || '[OL_MONTH]-[OL_DAY]'
    || ' 00:00:00.000000', 1, 'year')
  AND ol.ol_delivery_d >=
    '[OL_YEAR]-[OL_MONTH]-[OL_DAY]'
    || ' 00:00:00.000000'
GROUP BY o.o_ol_cnt
ORDER BY o.o_ol_cnt;
```

Query 13

```
select c_count,
  count(*) as custdist
from (select c_id, count(o_id)

  from customer
    left outer join orders on (
      c_w_id = o_w_id
      and c_d_id = o_d_id
      and c_id = o_c_id
      and o_carrier_id > 8)

  group by c_id)
    as c_orders (c_id, c_count)
group by c_count
order by custdist desc,
  c_count desc
```

```
SELECT c_orders.c_count,
  COUNT(*) AS custdist
FROM (SELECT c.c_id,
   COUNT(o.o_id) AS c_count
   FROM customer c
     LEFT OUTER JOIN orders o ON (
       c.c_w_id = o.o_w_id
       AND c.c_d_id = o.o_d_id
       AND c.c_id = o.o_c_id
       AND o.o_carrier_id >
         [OCARRIERID])
   GROUP BY c.c_id) AS c_orders

GROUP BY c_orders.c_count
ORDER BY custdist DESC,
  c_orders.c_count DESC;
```

Query 14

```
select 100.00 *
  sum(case
      when i_data like 'PR%'
      then ol_amount else 0 end) /
        (1+sum(ol_amount))
    as promo_revenue
from orderline, item

where ol_i_id = i_id
  and ol_delivery_d >=
    '2007-01-02 00:00:00.000000'

  and ol_delivery_d <
    '2020-01-02 00:00:00.000000'
```

```
SELECT 100.00 *
  SUM(CASE
      WHEN i.i_data LIKE 'PR%'
      THEN ol.ol_amount ELSE 0 END) /
        (1+SUM(ol.ol_amount))
    AS promo_revenue
FROM orders o, o.o_orderline ol,
  item i
WHERE ol.ol_i_id = i.i_id
  AND ol.ol_delivery_d >=
    '[OL_YEAR]-[OL_MONTH]-[OL_DAY]'
    || ' 00:00:00.000000'
  AND ol.ol_delivery_d <
    DATE_ADD_STR('[OL_YEAR]-'
    || '[OL_MONTH]-[OL_DAY]'
    || ' 00:00:00.000000', 1,
    'month');
```

Query 15

```
with revenue (supplier_no,
  total_revenue) as
  (select mod((s_w_id * s_i_id),10000)
     as supplier_no,
     sum(ol_amount) as total_revenue
   from orderline, stock

   where ol_i_id = s_i_id
     and ol_supply_w_id = s_w_id
     and ol_delivery_d >=
       '2007-01-02 00:00:00.000000'




   group by
     mod((s_w_id * s_i_id),10000))
select su_suppkey, su_name,
  su_address, su_phone,
  total_revenue
from supplier, revenue
where su_suppkey = supplier_no
  and total_revenue = (
    select max(total_revenue)
    from revenue)
order by su_suppkey
```

```
WITH revenue AS (

  SELECT s.s_w_id * s.s_i_id MOD 10000
    AS supplier_no,
    SUM(ol.ol_amount) AS total_revenue
  FROM orders o, o.o_orderline ol,
    stock s
  WHERE ol.ol_i_id = s.s_i_id
    AND ol.ol_supply_w_id = s.s_w_id
    AND ol.ol_delivery_d >=
      '[OL_YEAR]-[OL_MONTH]-[OL_DAY]'
      || ' 00:00:00.000000'
    AND ol.ol_delivery_d <
      DATE_ADD_STR('[OL_YEAR]-'
      || '[OL_MONTH]-[OL_DAY]'
      || ' 00:00:00.000000', 3,
      'month')
  GROUP BY
    s.s_w_id * s.s_i_id MOD 10000)
SELECT su.su_suppkey, su.su_name,
  su.su_address, su.su_phone,
  r.total_revenue
FROM supplier su, revenue r
WHERE su.su_suppkey = r.supplier_no
  AND r.total_revenue =(
    SELECT VALUE max(r1.total_revenue)
    FROM revenue r1)[0]
ORDER BY su.su_suppkey;
```

Query 16

```
select i_name,
  substr(i_data, 1, 3) as brand,
  i_price,
  count(distinct
    (mod((s_w_id * s_i_id),10000)))
    as supplier_cnt
from stock, item
where i_id = s_i_id
  and i_data not like 'zz%'

  and (mod((s_w_id * s_i_id),10000)
    not in
    (select su_suppkey
     from supplier
     where su_comment like '%bad%'))


group by i_name,
  substr(i_data, 1, 3), i_price
order by supplier_cnt desc
```

```
SELECT i.i_name,
  SUBSTR1(i.i_data, 1, 3) AS brand,
  i.i_price,
  COUNT(DISTINCT(
    s.s_w_id * s.s_i_id MOD 10000))
    AS supplier_cnt
FROM stock s, item i
WHERE i.i_id = s.s_i_id
  AND i.i_data NOT LIKE
    '[IDATAA][IDATAB]%'
  AND s.s_w_id * s.s_i_id MOD 10000
    NOT IN
   (SELECT su.su_suppkey
    FROM supplier su
    WHERE su.su_comment LIKE
      '%[SU_COMMENT_PREFIX]%'
      || '[SU_COMMENT_SUFFIX]%')
GROUP BY i.i_name,
  SUBSTR1(i.i_data, 1, 3), i.i_price
ORDER BY supplier_cnt DESC;
```

Query 17

```
select sum(ol_amount) / 2.0
  as avg_yearly
from orderline,
  (select i_id,
     avg(ol_quantity) as a
   from item, orderline

   where i_data like '%b'
     and ol_i_id = i_id
   group by i_id) t
where ol_i_id = t.i_id
  and ol_quantity < t.a
```

```
SELECT SUM(ol.ol_amount) / 2.0
  AS avg_yearly
FROM orders o, o.o_orderline ol,
  (SELECT i.i_id,
     AVG(ol1.ol_quantity) AS a
   FROM item i, orders o1,
     o1.o_orderline ol1
   WHERE i.i_data LIKE '%[IDATA]'
     AND ol1.ol_i_id = i.i_id
   GROUP BY i.i_id) t
WHERE ol.ol_i_id = t.i_id
  AND ol.ol_quantity < t.a;
```

Query 18

```
select c_last, c_id, o_id,
  o_entry_d, o_ol_cnt,
  sum(ol_amount)
from customer, orders,
  orderline
where c_id = o_c_id
  and c_w_id = o_w_id
  and c_d_id = o_d_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_o_id = o_id
group by o_id, o_w_id,
  o_d_id, c_id, c_last,
  o_entry_d, o_ol_cnt
having sum(ol_amount) > 200
order by sum(ol_amount) desc,
  o_entry_d
```

```
SELECT c.c_last, c.c_id, o.o_id,
  o.o_entry_d, o.o_ol_cnt,
  SUM(ol.ol_amount)
FROM customer c, orders o,
  o.o_orderline ol
WHERE c.c_id = o.o_c_id
 AND c.c_w_id = o.o_w_id
 AND c.c_d_id = o.o_d_id



GROUP BY o.o_id, o.o_w_id,
  o.o_d_id, c.c_id, c.c_last,
  o.o_entry_d, o.o_ol_cnt
HAVING SUM(ol.ol_amount) > [OLQUANTITY]
ORDER BY SUM(ol.ol_amount) DESC,
  o.o_entry_d
LIMIT 100;
```

Query 19

```
select sum(ol_amount) as revenue
from orderline, item

where (
    ol_i_id = i_id
    and i_data like '%a'
    and ol_quantity >= 1

    and ol_quantity <= 10

    and i_price between 1 and 400000
    and ol_w_id in (1,2,3)

  ) or (
    ol_i_id = i_id
    and i_data like '%b'
    and ol_quantity >= 1

    and ol_quantity <= 10

    and i_price between 1 and 400000
    and ol_w_id in (1,2,4)

 ) or (
   ol_i_id = i_id
   and i_data like '%c'
   and ol_quantity >= 1

   and ol_quantity <= 10

   and i_price between 1 and 400000
   and ol_w_id in (1,5,3)

 )
```

```
SELECT SUM(ol.ol_amount) AS revenue
FROM orders o, o.o_orderline ol,
  item i
WHERE (
    ol.ol_i_id = i.i_id
    AND i.i_data LIKE '%[IDATAA]'
    AND ol.ol_quantity >=
      [OLQUANTITYASTART]
    AND ol.ol_quantity <=
      [OLQUANTITYAEND]
    AND i.i_price BETWEEN 1 AND 5
    AND o.o_w_id IN [[OLWID.1],
      [OLWID.2], [OLWID.3]]
  ) OR (
    ol.ol_i_id = i.i_id
    AND i.i_data LIKE '%[IDATAB]'
    AND ol.ol_quantity >=
      [OLQUANTITYBSTART]
    AND ol.ol_quantity <=
      [OLQUANTITYBEND]
    AND i.i_price BETWEEN 1 AND 10
    AND o.o_w_id IN [[OLWID.4],
      [OLWID.5], [OLWID.6]]
 ) OR (
    ol.ol_i_id = i.i_id
    AND i.i_data LIKE '%[IDATAC]'
    AND ol.ol_quantity >=
      [OLQUANTITYCSTART]
    AND ol.ol_quantity <=
      [OLQUANTITYCEND]
    AND i.i_price BETWEEN 1 AND 15
    AND o.o_w_id IN [[OLWID.7],
      [OLWID.8], [OLWID.9]]
 );
```

Query 20

```
select su_name, su_address
from supplier, nation
where su_suppkey in
 (select mod(s_i_id * s_w_id, 10000)

  from stock, orderline

  where s_i_id in
   (select i_id
    from item
    where i_data like 'co%')

    and ol_i_id=s_i_id
    and ol_delivery_d >
      '2010-05-23 12:00:00'




    group by s_i_id, s_w_id,
      s_quantity
    having 2 * s_quantity >
      sum(ol_quantity))
   and su_nationkey = n_nationkey
   and n_name = 'Germany'
order by su_name
```

```
SELECT su.su_name, su.su_address
FROM supplier su, nation n
WHERE su.su_suppkey in
 (SELECT VALUE
    s.s_i_id * s.s_w_id MOD 10000
  FROM stock s, orders o,
    o.o_orderline ol
  WHERE s.s_i_id in
   (SELECT VALUE i.i_id
    FROM item i
    WHERE i.i_data LIKE
      '[IDATAA][IDATAB]%')
    AND ol.ol_i_id=s.s_i_id
    AND ol.ol_delivery_d >=
      '[OL_YEAR]-[OL_MONTH]-[OL_DAY]'
      || ' 00:00:00.000000'
    AND o.ol_delivery_d <
      DATE_ADD_STR('[OL_YEAR]'
      || '-[OL_MONTH]-[OL_DAY]'
      || ' 00:00:00.000000', 1,
      'year')
  GROUP BY s.s_i_id, s.s_w_id,
    s.s_quantity
  HAVING 20 * s.s_quantity >
    SUM(ol.ol_quantity))
  AND su.su_nationkey = n.n_nationkey
  AND n.n_name = '[NNAME]'
ORDER BY su.su_name;
```

Query 21

```
select su_name, count(*) as numwait
from supplier, orderline l1,
  orders, stock, nation

where ol_o_id = o_id
  and ol_w_id = o_w_id
  and ol_d_id = o_d_id
  and ol_w_id = s_w_id
  and ol_i_id = s_i_id
  and mod((s_w_id * s_i_id),10000)
    = su_suppkey
  and l1.ol_delivery_d > o_entry_d









  and not exists (select *
    from orderline l2
    where l2.ol_o_id = l1.ol_o_id
      and l2.ol_w_id = l1.ol_w_id
      and l2.ol_d_id = l1.ol_d_id
      and l2.ol_delivery_d >
        l1.ol_delivery_d)
  and su_nationkey = n_nationkey
  and n_name = 'Germany'
group by su_name
order by numwait desc, su_name
```

```
SELECT su.su_name, COUNT(*) AS numwait
FROM supplier su, orders o1,
  o1.o_orderline ol1, stock s,
  nation n
WHERE o1.o_w_id = s.s_w_id



  AND ol1.ol_i_id = s.s_i_id
  AND s.s_w_id * s.s_i_id MOD 10000
    = su.su_suppkey
  AND o1.o_entry_d BETWEEN
    '[O_YEAR]-[O_MONTH]-[START_O_DAY]'
    || ' 00:00:00' AND '[O_YEAR]'
    || '-[O_MONTH]-[END_O_DAY]'
    || ' 00:00:00'
  AND ol1.ol_delivery_d >
    DATE_ADD_STR(o1.o_entry_d,
      [NUMDAYS], 'day')
  AND NOT EXISTS (SELECT VALUE 1
    FROM orders o2, o2.o_orderline ol2
    WHERE o2.o_id = o1.o_id
      AND o2.o_w_id = o1.o_w_id
      AND o2.o_d_id = o1.o_d_id
      AND ol2.ol_delivery_d >
        ol1.ol_delivery_d)
  AND su.su_nationkey = n.n_nationkey
  AND n.n_name = '[NNAME]'
GROUP BY su.su_name
ORDER BY numwait DESC, su.su_name
LIMIT 100;
```

Query 22

```
select substr(c_state,1,1)
    as country,
  count(*) as numcust,
  sum(c_balance) as totacctbal
from customer
where substr(c_phone,1,1) in
    ('1','2','3','4','5','6','7')


  and c_balance >
   (select avg(c_balance)
    from customer
    where c_balance > 0.00
      and substr(c_phone,1,1) in
        ('1','2','3','4','5','6','7'))



  and not exists (
    select *
    from orders
    where o_c_id = c_id
      and o_w_id = c_w_id
      and o_d_id = c_d_id)
group by substr(c_state,1,1)
order by substr(c_state,1,1)
```

```
SELECT SUBSTR1(c.c_state,1,1)
    AS country,
  COUNT(*) AS numcust,
  SUM(c.c_balance) AS totacctbal
FROM customer c
WHERE SUBSTR1(c.c_phone,1,1) in
    ['[CCA.1]','[CCA.2]','[CCA.3]',
     '[CCA.4]','[CCA.5]','[CCA.6]',
     '[CCA.7]']
 AND c.c_balance >
   (SELECT VALUE AVG(c1.c_balance)
    FROM customer c1
    WHERE c1.c_balance > 0.00
      AND SUBSTR1(c1.c_phone,1,1) in
        ['[CCB.1]','[CCB.2]',
         '[CCB.3]','[CCB.4]',
         '[CCB.5]','[CCB.6]',
         '[CCB.7]'])[0]
      AND NOT EXISTS (
        SELECT VALUE 1
        FROM orders o
        WHERE o.o_c_id = c.c_id
        AND o.o_w_id = c.c_w_id
        AND o.o_d_id = c.c_d_id)
GROUP BY SUBSTR1(c.c_state,1,1)
ORDER BY SUBSTR1(c.c_state,1,1);
```