北京工业大学

硕士学位论文

基于TPC-C的数据库系统性能测试集的设计与实现

姓名：文栋

申请学位级别：硕士

专业：软件工程

指导教师：沈琦;陈晟

20071001

摘 要

关系型数据库系统已经逐渐成为公司或大型企业的 IT 系统中最核心的部分，而最能体现数据库能力的便是数据库的性能指标。鉴于数据库系统性能的重要性，其相应的测试方法就成为数据库领域的一个热门话题。许多数据库系统厂商都想通过数据库测试找到提高其性能的方法。作为一个权威的数据库性能评测方法，TPC-C 已经得到了数据库厂商的广泛认同。因此研究 TPC-C 规范并设计出一套有效的测试集便是一件很有意义的事情，它有助于数据库性能的提高以及进一步开发，对数据库的完善有着指导作用。

作为 TPC 组织发布的数据库联机事务处理性能测试标准，TPC-C 只给出了基准规范而不提供代码，所有的公司或个人都可以根据具体的规范设计出符合自己需要的测试集。

本文的主要内容就是深入研究 TPC-C 基准规范，在此基础上设计一套严格符合 TPC-C 标准的数据库性能测试集，然后将此测试集应用到实际数据库的测试中，得出测试结果，并给出恰当的分析。本测试集是在红旗 Linux DC Server 5.0 操作系统和 MySQL 5.0 数据库基础上设计的，由数据装载、事务处理、后处理(统计信息)三个模块组成。事务处理模块为程序的主模块，它负责处理 TPC-C 中设计的 5 种事务，包括下订单、付款、订单状态查询、发货和库存状态查询。测试集会对系统总吞吐量、响应时间等指标进行度量。在测试完成之后，测试系统会将事务的各响应时间、事务混合比以及最终测试结果 tpmC 值等信息记录在结果文件中。通过对结果信息的分析，我们可以判断此次测试是否满足 TPC-C 基准的要求，并评估测试系统的性能水平。

关键词：TPC-C；基准；数据库性能；测试集

Abstract

Relational Database Management Systems (RDBMS) have become more and more the core IT systems of companies and large enterprises. Due to the importance of database performance, the corresponding method of evaluation becomes a hot topic in the database testing field. Many DBMS companies always try to test their products and find method to improve the performance. As an authoritative evaluation method for testing database performance, TPC-C is accepted by almost all DBMS company. So it is valuable to investigate the TPC-C benchmark and design a database test-suite based on it. This will benefit the further development of databases system.

As an OLTP benchmark for database testing designed by Transaction Processing Performance Council, TPC-C only gives the benchmark specification instead of provide the code for testing. All the company or individuals can design their own test-suite according to the benchmark specification.

The main content of this thesis is study the specification of TPC-C benchmark, and design a set of database performance test-suite strictly fit for it. Then apply the test-suite to the database performance testing and analyze the test result. The test-suite this thesis designed is based on the RedFlag DC Server 5.0 linux operation system and MySQL 5.0 database system. It comprises three modules: loading data, transaction processing and post-processing. Transaction processing module is the primary module of the program. It deals with the five main transactions include ordering, paying, querying order, delivery item and querying stock. The test-suite measure the maximum qualified throughout (MQTh), response times, stable test interval, etc. The statistic information includes MQTh, response times of transactions, mix of the five transactions and the final test result – tpmC, etc. They all are stored in the test result file. According to the result information, we can judge if this testing keeps to the requirement of TPC-C benchmark t, and evaluate the performance of the database system.

Key Words: TPC-C; Benchmark; Database performance; test-suite

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京工业大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签名：_____ 日期：2008.1.2

关于论文使用授权的说明

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：_____ 导师签名：_____ 日期：2008.1.2

第 1 章 绪论

1.1 课题背景

数据库系统作为一种重要的计算机应用系统，由一组相互关联的数据集合和一组可以访问这些数据的程序组成，它经历了层次、网状、关系、对象等发展过程，已经成为一种相对成熟的研究科学。由于各种数据库产品提供的功能已经很相似，和外部的应用接口也已经变得越来越标准化，所以产品性能而不是功能开始成为不同数据库系统的主要差异因素。糟糕的性能在很大的程度上制约着数据库处理并发事务的能力，同时还影响着数据库的一致性。因此，保证数据库的良好性能对于核心数据库来说是相当重要的。

鉴于数据库系统的性能变得越来越重要，关于数据库性能测试也一直备受关注，由此人们研究了很多优化数据库性能的方法，也提出了很多评测和分析数据库性能的准则。

人们总希望有一种简单、高效的度量标准，来量化评价数据库服务器系统，以便作为选型的依据。但实际上，数据库服务器的系统性能很难用一两种指标来衡量。包括 TPC、SPEC、SAP SD、Linpack 和 HPCC 在内的众多服务器评测体系，从处理器性能、数据库系统性能、商业应用性能直到高性能计算机的性能，都给出了一个量化的评价指标。在如此多的标准中，用户该如何选择最适合自身应用环境的评价体系呢？归纳为以下三点：

1、在真实环境中运行实际应用

最理想的方式是准备一个试点，要求制造商或系统集成商配合将系统(含平台、软件和操作流程)在一个实际用户点真正试运行一段时间。这样，用户不仅能看到实际性能，也能观察到系统是否稳定可靠、使用是否方便、配置是否足够。这不仅是最精确、稳妥、最有效率的方法，用这种方式得到的度量值常常具有很明确和实际的含义。

2、使用用户定义的基准程序

用户还可以定义一组含有自己实际应用环境特征的应用基准程序。如近年来，由于 R/3 软件是应用层软件，SAP 公司的基准程序获得了越来越多国外企业的认可；中国税务总局也开发了自己的基准程序，以帮助税务系统进行计算机选

型。这种方式在中国尤其重要，因为中国的信息系统有其特殊性。

3、使用通用基准程序

如果前2种方式都做不到，则需使用通用基准程序。虽然从20世纪80年代开始人们曾提出许多不同的性能评价标准，但是很多评测标准都不十分严密，数据库企业公布的评测结果也都不一致，所以一直没有一个比较权威的数据库性能评测标准。直到由事务处理性能委员会(Transaction Process Performance Council, 简称TPC)提出的TPC-C性能测试，才完全的规范了数据库在事务处理性能方面的评测标准和测评结果，从而成为目前评价数据库性能的主流国际标准之一。[5]

TPC-C作为一种国际上权威的数据库性能测试为大多数数据库厂商所接受，并且对于TPC的测试结果所显示的数据库性能也已得到了数据库业界的普遍认可。同时对于TPC-C这类国际标准的研究和使用将促进数据库的性能的提高以及数据库的进一步开发，对于数据库的完善有着很大的指导意义。

1.2 数据库性能测试国内外研究现状

现今在数据库系统性能评价方面，存在着许多基准程序系列，其中主流的基准有TPC和SPEC系列等。SPEC(the Standard Performance Evaluation Corporation，标准性能评估机构)是一个全球性的、权威的第三方应用性能测试组织，它旨在确立、修改以及认定一系列服务器应用性能评估的标准。

TPC系列基准是现在流行的商业基准组，它由事务处理性能委员会(Transaction Processing Performance Council)开发，这是一个非盈利组织，主要服务器和数据库企业都派代表加入了这一组织，他们主要从事事务处理基准程序的开发。至今，TPC已经发布了若干基准程序，其中TPC-A, TPC-B, TPC-D, TPC-E, TPC-S等已经废弃，现在流行的有TPC-C，TPC-R，TPC-H和TPC-W。

- TPC-C 联机事务处理的基准测试
- TPC-H Ad-hoc 查询以及决策支持的基准测试
- TPC-R 商业报告以及决策支持的基准测试
- TPC-W 基于事务处理的互联网电子商务的基准测试

在TPC发布的所有基准测试当中，TPC-C是数据库OLTP性能测试标准。此外，它还适用于与数据库相关的整个服务器系统(包括硬件平台、操作系统和

数据库系统)性能的测试。作为一家非盈利性机构，事务处理性能委员会（TPC）依据这些基准测试项目发布客观性能数据。TPC 基准测试采用极为严格的运行环境，并且必须在独立审计机构监督下进行。委员会成员包括大多数主要数据库产品厂商以及服务器硬件系统供应商。相关企业参与 TPC 基准测试以期在规定运行环境中获得客观性能验证，并通过应用测试过程中所使用的技术开发出更加强健且更具伸缩性的软件产品及硬件设备。

TPC-C 是一种旨在让用户能够评估不同的联机事务处理（OLTP）系统性能与可伸缩性的行业标准基准测试项目，对包括查询、更新及队列式小批量事务在内的广泛数据库功能进行测试。它针对一种模拟订单录入与销售环境测量每分钟商业事务(tpmC)的吞吐量。

世界上很多知名的数据库厂商如 Oracle, SQL Server 等都已经通过了国际 TPC 组织的严格测试和权威认证，测试结果也都作为了数据库业界评价性能的事实标准。通常，执行一次 TPC-C 通常要花掉四百万美元，大部份是硬件花费。HP 机器于 2006 年创下至今为止排名第一的 TPC-C 成绩—4.09 百万 TPM，花了 1200 万美元。IBM 机器也花了 1200 万美元取得排名第二的 TPC-C 成绩－4.03 百万 TPM。因而以 TPC-C 作为基准进行的数据库系统测试，经常被称为“只有有钱公司才能支付得起的测试”。

相比之下，国产数据库的 TPC-C 测试就落后了很多。到目前为止，很少有国产数据库通过了 TPC-C 的测试，当然这一方面是由于数据库本身性能的问题，另一方面，国产数据库厂家也难以或不愿支付如此昂贵的费用来做一次 TPC-C 基准测试，因为要想得到好的性能指标和高的 tpmC 值，往往要花费巨额资金来购买高性能的硬件，以体现数据库系统的最大能力，这对于国产数据库厂家来说是难以承受的，也是没有必要的。国产数据库目前仍处在不断进步和完善的阶段，在成熟度和稳定性方面还有待进一步提高，在支持海量数据管理、安全性、系统可扩展性方面仍显薄弱，无需花费巨资与发展数十年的国外大牌数据库厂商比拼性能指标。其需要的是一个标准的 TPC-C 测试集在适合数据库本身的国产应用系统环境和相应的数据量下对系统进行规范化的测试，找出影响系统性能的因素。一方面对性能水平做出评价，以作为改进数据库产品性能的有效手段之一；另一方面能够与数据库基准性能测试的国际水平接轨。

值得欣喜的是国产数据库已经意识到基准性能测试的重要性，并且已经采取了相应的行动，设计出了一些 TPC-C 基准性能测试集，但大多数都不完善，运用起来与 TPC-C 基准也存在差异，甚至大部分情况下将其用作一种压力测试工具，即与 TPC-C 测试模拟环境描述不符，又使其测试结果值失去了本来的意义。因此设计一套完全符合 TPC-C 基准的测试集，规范其应用环境和测试流程就显得尤为重要，也更加有助于数据库管理系统本身的不断改进和提高。

1.3 本文主要研究内容及组织结构

本课题研究的主要内容是研究 TPC-C 当前版本的标准，分析基于 TPC-C 的数据库基准测试的商业模拟环境、系统组成、数据库结构、事务处理过程和方法，以及其执行规则和约束条件，在此基础上，设计完全符合 TPC-C 基准的数据库系统性能测试集，并运用此测试集对 Linux 操作系统上运行的典型关系数据库进行多次不同数据量的性能测试，得出性能测试结果，即 tpmC 值，并对比分析每次测试的系统联机事务处理(OLTP)性能，为数据库性能评估和分析提供量化的数值结果参考。

在论文的组织结构方面，本文在第二章对 TPC-C 基准背景知识做了介绍，包括 TPC 组织的简介，TPC-C 基准测试介绍，以及 TPC-C 模拟了怎样的商业环境；第三章给出了测试集的详细设计方案，深一层的讲解 TPC-C 测试集的系统组成、数据库的表结构以及如何初始化数据，并详细分析了测试集如何用虚拟用户模拟用户事务请求的操作过程，以及五种事务处理过程和数据库中各表中数据如何变化，第三章结尾还对 TPC-C 基准测试的执行规则、约束条件以及测试方法的局限性作了说明。第四章进入测试的实施阶段，描述了测试环境，介绍了测试集各模块的功能和作用，然后以 10、20、30 个仓库数的数据量分别进行了测试，并对系统的 CPU 占用率、内存占用情况等性能参数进行了分析，对系统的性能状态作出评价，最后根据测试结果 tpmC 值及数据统计信息分析测试结果的基准符合性，得出测试结论。

第 2 章 TPC-C 基准测试介绍

2.1 TPC 组织简介

TPC 组织(Transaction Processing Performance Council，事务处理性能委员会)是由数 10 家会员公司创建的非盈利组织，总部设在美国，该组织对全世界开放，但迄今为止，绝大多数会员都是美、日、西欧的大公司。TPC 的成员主要是计算机软硬件厂家，而非计算机用户，会员从成立之初的 8 家公司发展到目前的 50 余家，IBM、NCR、HP、Oracle、Microsoft 等国际著名公司均是其会员。它的功能是制定商务应用基准程序(Benchmark)的标准规范、性能和价格度量，并管理测试结果的发布。

TPC 的出版物是开放的，可以通过网络获取(http://www.tpc.org)。TPC 不给出基准程序的代码，而只给出基准程序的标准规范(Standard Specification)。任何厂家或其它测试者都可以根据规范，最优地构造出自己的系统(测试平台和测试程序)。TPC 基准主要是定义一套功能需求，而这些需求可以运行在任意一个事务处理系统上(硬件或者操作系统上)，为保证测试结果的客观性，被测试者(通常是厂家)必须提交给 TPC 一套完整的报告(Full Disclosure Report)，包括被测系统的详细配置、分类价格和包含五年维护费用在内的总价格。该报告必须由 TPC 授权的审核员核实(TPC 本身并不做审计)。现在全球只有几个审核员，全部在美国。这种方法允许任何一个测试者用专有的或者开放的系统来实现 TPC 基准，并保证测试者能看到一个有可比性的比较，这一点和其它限制在一个系统上运行的标准是有很大的不同的。[2]

相比来说，其它很多基准是一种单独的系统测试，这种基准不能评估诸如用户接口，通信情况，数据存储等多方面的性能。与之不同的是 TPC 是以实际的商业应用环境为基础建模的，因此可以很好的衡量系统各方面的性能。但由于系统从操作系统、数据库、客户端各个层面的变动都有可能影响其测试结果，因此 TPC-C 基准最大的困难就是在商业应用模型中减少系统的多样性，保留系统的性能特征(系统使用水平和操作的复杂性)。

2.2 TPC-C 基准测试

TPC 基准™C（TPC-C）是一个反映系统在线事务处理（OLTP）能力的标准规范。它是读操作和更新事务操作频繁交互的混合处理过程，模拟复杂 OLTP 应用环境中的事务活动，通过把各个的系统组成部件,包括处理器、内存、网络、存储、操作系统和数据库软件,与特定环境相关联来实现的。这些特定环境包括：

- 不同复杂程度的多种事务类型的同时执行
- 具有在线和延迟的事务执行模式
- 支持多个在线终端会话
- 协调系统运行和应用程序的执行时间
- 支持大容量磁盘输入/输出
- 事务完整性（ACID 属性）
- 通过主关键字和次关键字来访问非均匀分布的数据
- 数据库是由具有多种大小、属性和关联的多个表组成的
- 数据访问和更新冲突

这是 TPC-C 基准测试的独特之处，很多其它基准测试都无法通过这样一种方式来证明系统的总体系统性能表现。

TPC-C 报告的性能度量是一个“商业吞吐量”，它测量每分钟处理的订单数。多种事务被用于模拟处理订单的商业活动，且每个事务受反应时间约束的支配。基准的性能度量用每分钟新增订单事务数（tpmC）表示。为了与 TPC-C 标准一致，TPC-C 结果的所有参考值必须包括 tpmC 值、与之相关的每个 tpmC 的价格和给定系统价格的可用日期。[7]

基准规定了相关数据的模型和执行的规则，数据库可以用任何可用的商业数据库管理系统（DBMS）、数据库服务器、文件系统或其它提供功能等价的数据存储方式来实现。TPC-C 使用的术语和度量与其它 TPC 或者另外一些机构组织的基准相似。这样的术语相似性并不表示 TPC-C 结果与其它基准具有可比性。唯一与 TPC-C 结果具有可比性的是相同版本的其它 TPC-C 结果。

尽管这种基准提供了很好的环境来模拟许多联机事务处理设备，但是这种基准不能反映全部的联机事务处理所涉及的必要条件。顾客能达到厂商报告的结果的很大程度依赖于 TPC-C 能多大程度模拟顾客应用程序。测试系统的性能不是可

以被其他的工作量和外界因素所控制的，不推荐使用对其他环境的外推法。

基准结果很大程度上依赖系统负载、特定的应用需求和系统设计与实现。相关的系统性能将随着这些或其它一些因素变化。因此，当预测关键的临界值或产品评估决策时，TPC-C 不能被用作特定的顾客应用基准的代替品。

TPC-C 基准是由一套设计用来在典型的复杂联机事务处理运行环境下执行系统功能的基础操作组成。这些基础操作可以提供上下文关系、描述批发商的行为、帮助用户直观的同基准组成相联系。系统负载以处理订单行为为中心，提供逻辑的数据库设计。

TPC-C 虽然不代表特定的交易片断的活动，但是更适合那些必需有管理、出售/发布产品或提供服务信息的行业(例如汽车租用、食品销售、部件供应商等)。TPC-C 并不关注怎样构造真实应用程序的模型。

建立基准的目的是在保持应用程序执行的基本特点的同时简化由生产应用程序所产生的运转多样性，应用程序执行的基本特点也就是系统利用水平和运转的复杂性。我们需要大量的功能处理来管理产品订单登录系统，这里面的许多功能对于性能分析来说不是最重要的，因为它们耗费很小的系统资源，并且并不频繁的执行，尽管这些功能对于系统产品是至关重要的，但是它们对于基准来说创造了额外的多样性，因此 TPC-C 将它们省略掉。[7]

2.3 TPC-C 基准商业模型

TPC-C 是专门针对联机交易处理系统（OLTP 系统）的，一般情况下我们也把这类系统称为业务处理系统。TPC-C 模拟了一个复杂的环境，大量的终端操作者可以在同一个数据库上频繁的执行各种事务。这个环境以新增订单事务为核心活动，其它事务包括在一个仓库范围内区域中进行的支付操作、订单状态查询、发货、库存水平查询。需要指出的是 TPC-C 的内容并不是在于怎样实现一个新订单事务，而是描述了一个货物批发商的所有的活动，它并不仅限于某个特定的商业运行环境，相反，它体现了任何一个可以管理，销售，分配产品及服务的通用的商业环境。

TPC-C 测试规范中模拟了一个比较复杂并具有代表意义的 OLTP 应用环境：假设有一个大型商品批发商，在地理分布的多个区域有业务，并且使用仓库管理。

每个仓库负责为 10 个区域的供货；每个区域为 3000 个客户提供服务；每个仓库维护公司销售的 100,000 条商品的库存记录。每个客户平均一个订单有 10 项产品；所有订单中约 1%的产品在其直接所属的仓库中没有存货，需要由其他区域的仓库来供货。图 2-1 描述了 TPC-C 商业模型的逻辑结构。

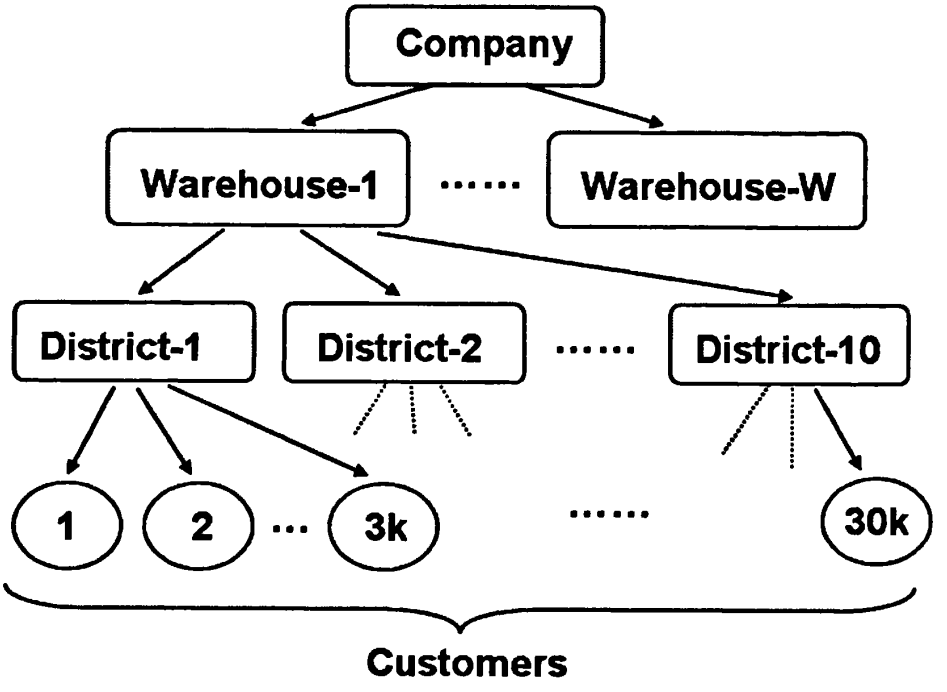

图 2-1 TPC-C 商业模型逻辑结构图

Figure 2-1 The logical structure legend of TPC-C commercial model

该系统需要处理的交易为以下五种：

- New-Order: 客户输入一笔新的订货交易
- Payment: 更新客户账户余额以反映其支付状况
- Delivery: 发货(模拟批处理交易)
- Order-Status: 查询客户最近交易的状态
- Stock-Level: 查询仓库库存状况，以便能够及时补货

对于前四种类型的交易，要求响应时间在 5 秒以内；对于库存状况查询事务，要求响应时间在 20 秒以内。[7]

TPC-C 的事务处理在一个以 9 张表为基础的数据库上实现处理过程，其数据量及其对应关系如图 2-2 所示。

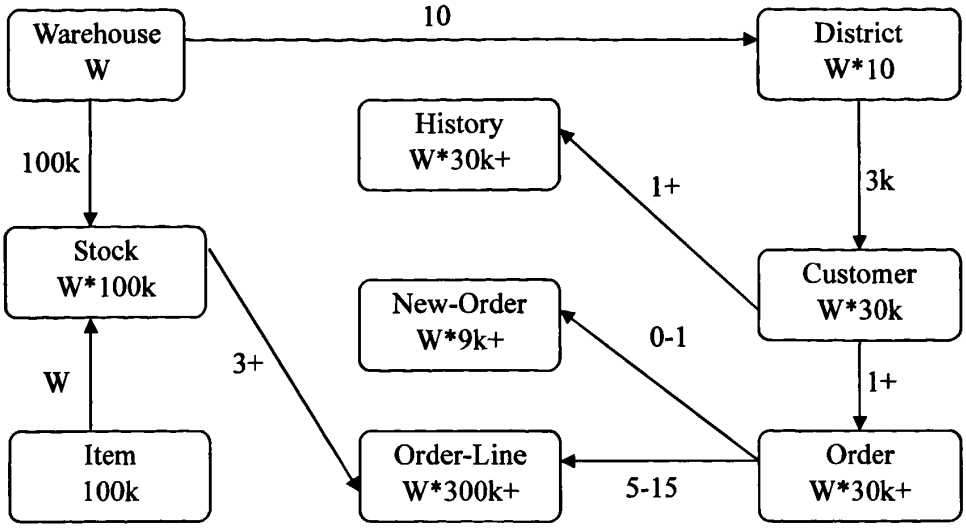

图 2-2 表的数据量及其对应关系

Figure 2-2 The data volume and relationship of tables

图 2-2 中，方框内为各个表的名称及记录条数，W 表示仓库的数量，仓库数 W 的调整在测试中能够体现数据库所能够支持的数据规模的能力。

箭头上的数字表示表数据的父子关系之间儿子的倍数，如 Warehouse→District 之间箭头上的 10 表示 1 个 Warehouse 对应 10 个 District。

+：表示个数可能会更多；K：表示 1000。

TPC-C 测试中所用到的五种事务也都是对以上 9 张表的读、写、更新、查询等操作。客户可以向此批发商请求一张新订单或者查询一张已存在订单的状态。每张订单要平均包括 10 条订单项。这些订单项中有大概 1%不能从本地仓库中得到，而必须从别的仓库中获得。除此以外，此模型还支持用户付款，公司发货以及检查商品存储情况三项活动。上面提到的 5 种操作就是 TPC-C 测试中频繁被用到的 5 种事务。[7]

根据图 2-2 中 9 张表中每张表的记录条数，我们可以计算出数据库大致的数据量，具体数值见表 2-1。

表 2-1 数据库数据量

Table 2-1 The database population

序号	表 名	记录条数(条)	典型字段长度 (bytes)	典型表大小 (1000 bytes)
1	Warehouse	1	89	0.089
2	District	10	95	0.950
3	Customer	30k	655	19650
4	History	30k	46	1380
5	Order	30k	24	720
6	New-Order	9k	8	72
7	Order-Line	300k	54	16200
8	Stock	100k	306	30600
9	Item	100k	82	8200
合计				76823.04

从表 2-1 中可以看出，由于 TPC-C 的各个表的数据量之间存在着特定的比例对应关系．因此 TPC-C 的数据量可以用 Warehouse 表中的数据量(行数)为基数进行估算。就是说，我们只要知道数据库中有多少个 Warehouse，即 Warehouse 表中有多少行数据，就可以大致确定其数据库中数据总量的大小。

数据量与 Warehouse 之间的关系为：每个 Warehouse 对应大约 80MB 数据。

计算公式为：数据总量（KB）≈Warehouse 个数*76823.04KB

以 10 个 Warehouse 的数据量为例计算其数据总量大小约为：10*76823.04=768230.4KB。通常 TPC-C 测试报告中说明的性能指标可以达到支持多少个 Warehouse 数目，这也就说明了其可以支持的数据量大小。

2.4 本章小结

本章首先对发布 TPC-C 基准程序的 TPC 组织的作了简要介绍，讲述 TPC 组织的主要成员和基本功能。然后对 TPC-C 基准测试规范作了简要介绍，主要从 TPC-C 基准测试的系统行为、区别于其他基准程序的特点、如何度量系统性能，并从相关执行约束以及对于客户的适用性等方面进行描述。最后讲解了 TPC-C 的商业应用模型，阐述其模拟的大型销售商的商业活动、涉及的五种事务操作、事务处理所需用到的 9 张表以及各表之间的对应关系，并且通过计算出 1 个仓库规模近似的数据库数据量，估算出测试过程中总的数据量大小。

第 3 章 TPC-C 测试集设计

首先明确一下此次测试的目的：

1．得出性能测试的量化结果，即 tpmC 值。

2．对测试结果进行讨论，分析数据库性能的特点，以期找到优化方案。

3．弥补测试过程中发现的不足。

要达到以上目的，采用的测试方法：

1．严格遵循 TPC-C 基准规范，设计测试框架及模块间的通信方法。

2．根据数据库特点，配置合理系统参数、测试的 Ramp-up 时间、系统稳定运行的时间，得出准确的测试结果。

3．通过测试过程中各项指标和系统资源消耗情况的记录，分析测试的有效性，评价数据库性能。

3.1 测试系统组成

在对 MySQL 数据库的性能测试过程中采取了 C-S 的测试框架，测试系统的结构主要由三部份构成：

SUT：System Under Test，待测试的系统。

驱动系统：提供 RTE 功能，RTE 即 Remote Terminal Emulate，远程终端模拟器。

驱动系统和 SUT 之间的通信接口。

其中在具体的实现过程中多个终端是由多个进程来模拟的。

如图 3-1：

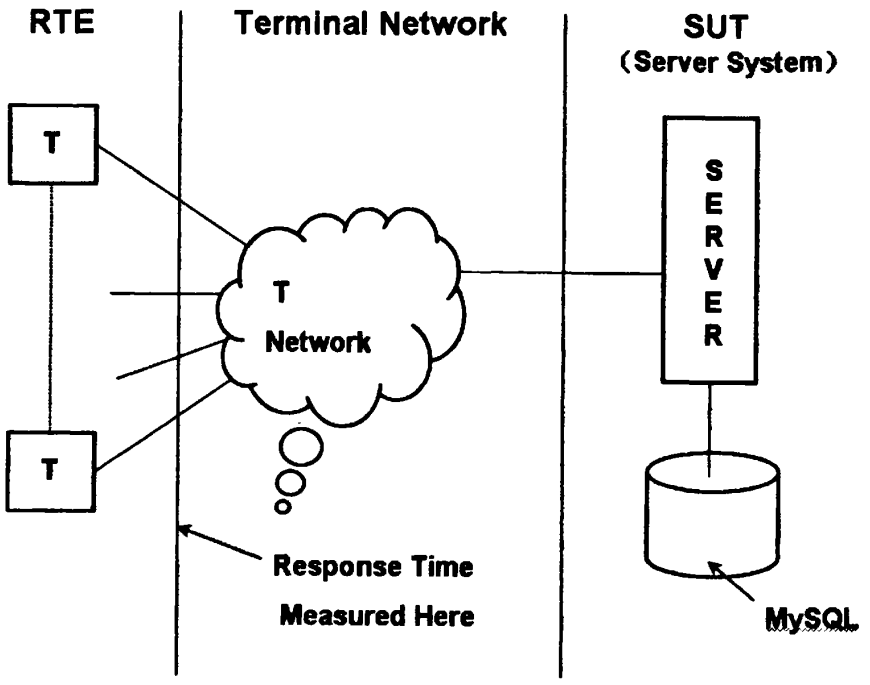

图 3-1 测试系统模型图

Figure 3-1 The model legend of test system

SUT 的组成如下：

1. 装载着 MySQL 数据库的主机系统，包括硬件和软件，它接收出客户端送入的数据并处理相应的请求，同时送回结果数据。

2. 用于连接和支持 SUT 的所有网络硬件和软件成分。

3. 足以满足可扩展需求和 ACID 特征的数据存储媒体。

RTE 的功能：

• 模拟一个用户在仿真终端的输入输出屏幕上录入输入数据，产生并发送事务消息给 SUT；

• 模拟一个终端在输入输出屏幕上显示输出消息，接受来自 SUT 的响应消息来；

• 记录事务执行的响应时间；

• 执行驱动系统与 SUT 之间通信接口所用的通信协议的转换，执行统计记账（如：计算测量 90%的响应时间、吞吐量、事务数的计算等）。

3.2 数据库结构设计

TPC-C 基准中数据库由第二章中 2.3 小节提到的九张表组成，在测试开始之前需要在数据库中初始化数据库和符合 TPC-C 基准规定的 9 张表，下面将逐一介绍每张表的结构和它们的数据组织，以及测试集设计中的实现方式。

3.2.1 表结构初始化

TPC-C 对被测数据库中 9 张表的字段属性及数据内容都有严格规定，对于每个表，在附录中的表结构中分别列出的各表属性列表可使用被测系统中可用的任何物理表示和顺序来实现，基本表和属性名可以用不同的名字。

测试集中表结构用函数 create_tables()生成，严格按照附录中的表结构属性描述设计。在 SHELL 脚本中对于每个基本表，定义一个与其表名称相同的变量，赋予初值为创建表的 SQL 语句字符串，下面为 warehouse 表对应的变量赋值代码：

```
WAREHOUSE="CREATE TABLE warehouse (
  w_id int(11) NOT NULL default '0',
  w_name varchar(10) default NULL,
  w_street_1 varchar(20) default NULL,
  w_street_2 varchar(20) default NULL,
  w_city varchar(20) default NULL,
  w_state char(2) default NULL,
  w_zip varchar(9) default NULL,
  w_tax double default NULL,
  w_ytd decimal(24,12) default NULL,
  PRIMARY KEY   (w_id)
)"
```

引号中的内容为 warehouse 表的建表 SQL 语句。其他 8 张表也同样根据上面 TPC-C 定义的基本表属性来创建变量，并赋予初始值为对应表的建表 SQL 语句字符串。

然后在 SHELL 脚本中调用 SQL 语句来创建数据库表结构：

```
TABLES="STOCK ITEM ORDER_LINE ORDERS NEW_ORDER HISTORY
CUSTOMER DISTRICT WAREHOUSE"
for TABLE in $TABLES ; do
command_exec "$MYSQL $DB_NAME -e \"\$$TABLE
ENGINE=$DB_ENGINE\""
```

其中$DB_NAME 为数据库名，$TABLE 变量依次取各表对应的变量名，从而完成

全部 9 张表在数据库中的创建。

3.2.2 表数据初始化

在将数据库中表结构定义好之后，进行测试前，数据库的每个表中都必须有一定的初始数据，TPC-C 规定了各表中初始数据的产生应遵循的规则，并且数据量以 1 个仓库为单位按比例扩展，其对应关系在 2.3 小节中已经介绍。此节详细介绍每张表在测试前产生初始化数据的方法及具体内容。

术语解释：

random：在指定的范围内任意选择一个值。

random a-string[x ..y]：随机产生一个字符串，其最小长度为 x，最大长度为 y,平均长度为(y+x)/2。可以是字母字符串也可以是数字字符串.采用的算法是通过产生随机数来得到字符串的长度 N ，然后再随机产生长度为 N 的字符串。

C_LAST：在 CUSTOMER 表中的 C_LAST 字段内容必须通过以下方法产生。在下表中每个数字会对应相应的字符串，在 0-999 中随机产生一个数，分别将产生的随机数的每一位换成对应的字符串即的到 C_LAST 的值。

0	1	2	3	4	5	6	7	8	9
BAR	OUGHT	ABLE	PRI	PRES	ESE	ANTI	CALLY	ATION	EING

例如：产生随机数为 234，那它对应的 C_LAST 的值为 ABLEPRIPRES。

unique within[x]：在小于 x 的范围内取值，且每个值必须唯一即某个值只能被唯一的一条记录所使用。

random within[x..y]：在 x、y 的范围内随机取值，某个值可以被重复使用。

random permutation of [x .. y]：表示从 x 到 y 随机顺序排列的一组数字。这一般被称为无重复排列(或选择)。

W_ZIP,D_ZIP,C_ZIP：这三个属性必须通过以下的方式实现：

1. 产生一个四位的数字随机字符串

2. 在这个字符串后面加上‘11111’

例如:如果产生的字符串是 0825,则最后产生的对应的 ZIP 值就是 082511111。在每个 WAREHOUSE 中有 30,000 个客户但是却只有 10,000 个 ZIP 值可用，因此在每个仓库中平均每三个客户会使用相同的 ZIP 值。

Non_uniform random：只用来产生 Customer numbers,customer last names 和 items numbers，它通过在一个范围[x. . y]内产生的一个随机数来产生。这个数由一个函数 NURand 来产生，它的结果必须被转换成一个合法的数字或者名字。

NURand 函数的实现如下：

NURand(A,x,y) = (((random(O,A)| random(x,y))+C)%(y-x+1))+x

其中：

exp-1| exp-2 代表了在 exp-1 和 exp-2 之间的逻辑或操作

exp-1 % exp-2 表示 exp-1 余 exp-2 的值

random(x,y)代表在[x..y]中随机选取一个值

对于上面提到的三个属性：C_LAST,C_ID,OL_I_ID，在指定的[x..y]范围内 A 的值是固定的：

C_LAST： 范围[0..999] A=255

C_ID： 范围[1. . 3000] A=1023

OL_I_ID： 范围[1..100000] A=8191

C 是在[O..A]中的一个随机选择的值。

下面分别描述数据库中每个表的初始化数据：[7]

- 在 ITEM 表中，记录数为经销商的总商品数，恒定为 100,000 行，其中：

  I_ID unique within [100,000]

  I_IM_ID random within [1 .. 10,000]

  I_NAME random a-string [14 .. 24]

  I_PRICE random within [1.00 .. 100.00]

  I_DATA random a-string [26 .. 50]. 在随机选择的 10%的行中，字符串"ORIGINAL"必须由 I_DATA 内随机位置开始的 8 个连续的字符组成。

- 在 WAREHOUSE 表中每个指定的仓库要配置 1 行数据，其总行数为指定仓库的数量，其中：

  W_ID unique within [指定的仓库数]

  W_NAME random a-string [6 .. 10]

  W_STREET_1 random a-string [10 .. 20]

W_STREET_2　　random a-string [10 .. 20]

W_CITY　　random a-string [10 .. 20]

W_STATE　　random a-string of 2 letters

W_ZIP　　由上面术语解释中介绍的方法生成

W_TAX　　random within [0.0000 .. 0.2000]

W_YTD = 300,000.00

对于 WAREHOUSE 表中的每一行：

- 在 STOCK 表中有 100,000 行，其中：

S_I_ID　　unique within [100,000]

S_W_ID = W_ID

S_QUANTITY　　random within [10..100]

S_DIST_01　　random a-string of 24 letters

S_DIST_02　　random a-string of 24 letters

S_DIST_03　　random a-string of 24 letters

S_DIST_04　　random a-string of 24 letters

S_DIST_05　　random a-string of 24 letters

S_DIST_06　　random a-string of 24 letters

S_DIST_07　　random a-string of 24 letters

S_DIST_08　　random a-string of 24 letters

S_DIST_09　　random a-string of 24 letters

S_DIST_10　　random a-string of 24 letters

S_YTD = 0

S_ORDER_CNT = 0

S_REMOTE_CNT = 0

S_DATA　　random a-string [26 .. 50]. 在随机选择的 10%的行中，字符串"ORIGINAL"必须由 S_DATA 内随机位置开始的 8 个连续的字符组成。

- 在 DISTRICT 表中有 10 行，其中：

D_ID　　unique within [10]

D_W_ID = W_ID

D_NAME　　random a-string [6 .. 10]

D_STREET_1　　random a-string [10 .. 20]

D_STREET_2　　random a-string [10 .. 20]

D_CITY　　random a-string [10 .. 20]

D_STATE　　random a-string of 2 letters

D_ZIP　　由上面术语解释中介绍的方法生成

D_TAX　　random within [0.0000 .. 0.2000]

D_YTD = 30,000.00

D_NEXT_O_ID = 3,001

对 DISTRICT 表中的每一行：

✧ 在 CUSTOMER 表中有 3,000 行，其中：

C_ID　　unique within　[3,000]

C_D_ID = D_ID

C_W_ID = D_W_ID

C_LAST　　根据上面介绍的方法生成，对于前 1,000 个顾客，在范围[0 .. 999]内重复，对剩下的 2,000 个顾客，用函数 NURand (255,0,999)来生成一个非一致随机的数字。运行期常数 C 必须独立于测试的运行随机选择。

C_MIDDLE = "OE"

C_FIRST　　random a-string [8 .. 16]

C_STREET_1　　random a-string [10 .. 20]

C_STREET_2　　random a-string [10 .. 20]

C_CITY　　random a-string [10 .. 20]

C_STATE　　random a-string of 2 letters

C_ZIP　　由上面术语解释中介绍的方法生成

C_PHONE　　random n-string of 16 numbers

C_SINCE　　当生成 CUSTOMER 表时由系统给出日期/时间。

C_CREDIT = "GC". 在随机选择的 10%的行中，C_CREDIT = "BC"

C_CREDIT_LIM = 50,000.00

C_DISCOUNT　　random within [0.0000 .. 0.5000]

C_BALANCE = -10.00

C_YTD_PAYMENT = 10.00

C_PAYMENT_CNT = 1

C_DELIVERY_CNT = 0

C_DATA random a-string [300 .. 500]

对 CUSTOMER 表中的每一行：

➢ 在 HISTORY 表中有 1 行，其中：

H_C_ID = C_ID

H_C_D_ID = H_D_ID = D_ID

H_C_W_ID = H_W_ID = W_ID

H_DATE 当前的日期和时间

H_AMOUNT = 10.00

H_DATA random a-string [12 .. 24]

✧ 对于 District 表中每一行，在 ORDER 表中有 3,000 行，其中：

O_ID unique within [3,000]

O_C_ID selected sequentially from a random permutation of [1 .. 3,000]

O_D_ID = D_ID

O_W_ID = W_ID

O_ENTRY_D 由系统给定的当前日期/时间

O_CARRIER_ID 如果 O_ID < 2,101，在[1 .. 10]间随机选择，否则为 0

O_OL_CNT random within [5 .. 15]

O_ALL_LOCAL = 1

对 ORDER 表中的每一行：

★ ORDER-LINE 表中有数目为 O_OL_CN（平均为 10 行）的多行记录，并根据 New-Order 事务生成输入数据的规则生成记录：

OL_O_ID = O_ID

OL_D_ID = D_ID

OL_W_ID = W_ID

OL_NUMBER　　unique within [O_OL_CNT]

OL_I_ID　　random within [1 .. 100,000]

OL_SUPPLY_W_ID = W_ID

OL_DELIVERY_D = O_ENTRY_D　如果 OL_O_ID < 2,101, 否则为 0

OL_QUANTITY = 5

OL_AMOUNT = 0.00　如果 OL_O_ID < 2,101, 在[0.01 .. 9,999.99]间随机选择，否则取 0

OL_DIST_INFO　　random a-string of 24 letters

✧ 对于 DISTRICT 表中的每一行，在 NEW ORDER 表中要加入 900 行数据，这 900 行数据要对应于那个街区中最新的 900 条订单（也就是说，NO_O_ID 在 2,101 和 3,000 之间），其中：

NO_O_ID = O_ID

NO_D_ID = D_ID

NO_W_ID = W_ID

注释：允许含有“ORGINAL”的 S_DATA，含有“ORIGINAL”的 I_DATA，含有“BC”的 C_CREDIT 与目标基数有 5%的差异，这是为了解决在数据库的初始导入期间遇到的随机变化。

测试数据的初始化可以分两步操作，先将数据生成到数据文件中，然后再通过调用 SQL 语句将数据文件导入到数据库表中，这样操作是因为每次测试初始化的数据是相同的，为了避免多次重复的生成数据，造成时间浪费，将数据以文件形式预先存放起来，每次只需要进行数据导入操作即可。需要注意的是，数据规模不一样，需要生成的数据大小也不一样，因而不用仓库数的数据生成各自的数据分组存放。在测试集中，这些数据由 datagen 可执行程序生成，datagen 程序把按上述规则生成的数据放入与各表名称对应的.data 文件中，运行时需要指定仓库数以决定生成数据的规模，不同仓库数的数据生成到不同的文件夹中。

3.3 虚拟用户操作过程

TPC-C 基准测试中，所有的事务的发起均由 RTE（远程终端模拟器）模拟，

也就是说客户端均为虚拟用户，其所有操作也是由程序控制的，并记录事务的响应时间，下面介绍虚拟用户具体的操作过程及测量响应时间的方法。

术语解释：

菜单反应时间(menu RT)：menu response time，是菜单选项的最后一个字符被输入之前记录的时间戳和输入/输出屏幕的最后一个字符被接收之后记录的时间戳之间的时间（包括清除所有输入和输出域及显示固定域）。

等待时间(wait times)：包括按键时间(keying time)和思考时间(Think Time)。

按键时间(keying time)：用户在终端显示页面输入数据的时间。

思考时间(Think Time)：用户操作前的思考时间。

事务响应时间(RT)：是指最后一个输入字符被送出 RTE 和最后一个输出字符被显示在用户终端之间的时间。这个数据要根据测试的框架结构而定，如果这些输入数据或者输出数据在 RTE 端经过了处理，则这些处理时间也必须包含在响应时间内。图 3-2 表示了每个模拟用户执行的周期。

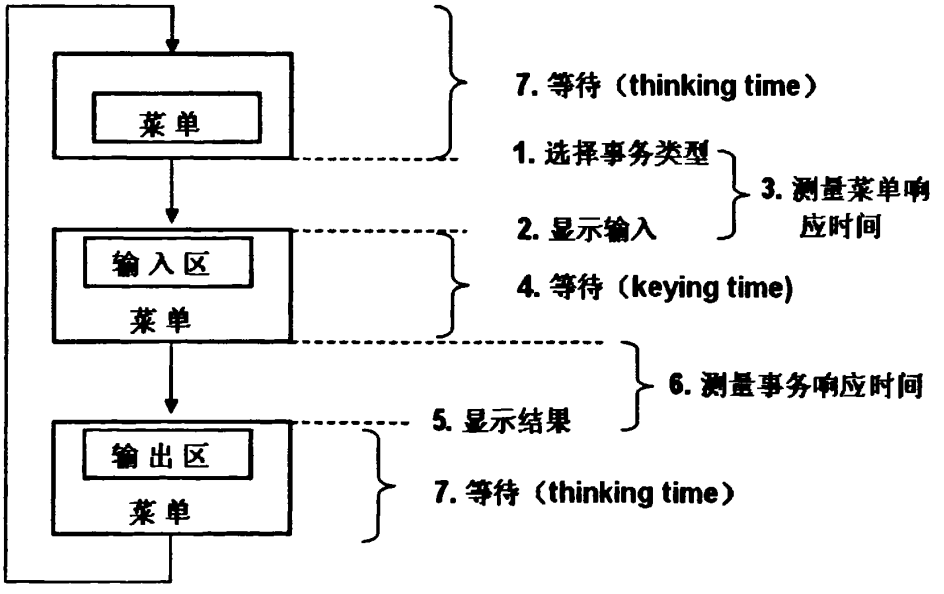

图 3-2　模拟用户执行周期

Figure 3-2 The executive cycle of emulate users

每个模拟用户执行由显示、等待时间和响应时间（RTs）组成的循环，具体操作过程如下：

1．根据加权分布从菜单中选择一个事务类型。

2．等待输入/输出屏幕显示。

3．测量菜单响应时间 Menu RT。

4．满足定义的最小按键时间基础上在输入域中输入要求的数字。

5．等待输出域要求的数字在输入/输出屏幕上显示。

6．测量事务响应时间 Txn RT。

7．等待当输入/输出屏幕显示时定义的最小思考时间。

在思考时间（第 7 步）结束时，模拟用户往回循环到从菜单选择事务类型（第 1 步）。

注释：从第 7 步返回到第 1 步被测系统 SUT 不要求任何动作。

3.4 事务处理过程

TPC-C 标准测试模拟了 5 种事务处理，通过这些事务处理来模拟真实的用户操作，新订单事务为其核心事务操作，其它四种事务分别为支付操作、订单状态查询、发货、库存状态查询。其执行的事务内容及特点如下：

1.新订单(New-Order)

对于任意一个客户端，从固定的仓库随机选取 5-15 件商品，创建新订单。其中 1%的订单要由假想的用户操作失败而回滚，其特点是处理的数据量为中级，读写频繁,要求响应快。

2.支付操作(Payment)

对于任意一个客户端，从固定的仓库随机选取一个辖区及其内用户，采用随机的金额支付一笔订单，并作相应历史纪录。其特点是数据量为中级，读写频繁，响应快。

3.订单状态查询 (Order-Status)

对于任意一个客户端，从固定的仓库随机选取一个辖区及其内用户，读取其最后一条订单，显示订单内每件商品的状态。其特点是处理的数据量为中级，读较少，响应快。

4.发货(Delivery)

对于任意一个客户端，随机选取一个发货包，更新被处理订单的用户余额，并把发货订单从新订单中删除。其特点是处理的数据量为轻级，读写频繁，响应快。

5.库存状态查询(Stock-Level)

对于任意一个客户端，从固定的仓库和辖区随机选取最后 20 条订单，查看订单中所有的货物的库存，计算并显示所有库存低于随机生成域值的商品数量。其特点是处理的数据量为重级，读较少，较宽松的响应时间。

通过上面这五种事务，对数据库进行大事务量的 TPC-C 测试。其中包括测试数据库的 tpmC(每分钟执行的事务数)以及回滚事务数。下面具体介绍每个事务的对各表的详细操作过程：

1.New-Order 事务

New-Order 事务主要是输入一张完整的订单。tpmC(每分钟的事务数)实际上就是计算每分钟执行的 New-Order 事务数。所以这项事务在 TPC-C 测试中占有着相当重要的地位。其执行步骤如下：

输入数据：

- 仓库号(W_ID)：发起事务的终端对应的仓库号
- 地区号(D_W_ID、D_ID)：D_W_ID=W_ID,D_ID=random[1..10]
- 客户号(C_W_ID、C_D_ID、C_ID)：C_W_ID=W_ID，C_D_ID=D_ID、C_ID=NURand(1023,1,3000)
- 订货种类数(*ol_cnt*)：random[5..15]
- 1%新订单模拟用户输出数据错误而回滚，rbk=ramdom[1..100]
- 货物号(OL_I_ID)：由函数 NURand(8191,1,100000)生成 *ol_cnt* 个 OL_I_ID，若 rbk=1，则取无用的货物号。
- 送货仓库号(OL_SUPPLY_W_ID)： x=random[]1..100]

  若 x>1 则本地供货，OL_SUPPLY_W_ID = W_ID

  x=1 则需要远程供货， OL_SUPPLY_W_ID 为远程 W_ID
- 每种货物订购数量(OL_QUANTITY)：random[1..10]

■ 订单日期(O_ENTRY_D)：当前系统日期

事务过程：

1) 生成的数据传送到 SUT：

仓库号(W_ID)，地区号(D_W_ID,D_ID)，客户号(C_W_ID,C_D_ID,C_ID)，订购种类数(*ol_cnt*)，货物号(OL_I_ID)，供货仓库号(OL_SUPPLY_W_ID)，每种货物订购数量(OL_QUANTITY)

2) 事务开始；

3) 找到表 WAREHOUSE 中的行 W_ID，取出仓库税 W_TAX；

4) 找到表 DISTRICT 中的行 (D_W_ID,D_ID)，取出地区税 D_TAX，取出新订单号 D_NEXT_O_ID,且 D_NEXT_O_ID 自加 1；

5) 找到表 CUSTOMER 中的行 (C_W_ID,C_D_ID,C_ID)，取出客户折扣率 C_DISCOUNT，客户名(C_LAST)，信用卡状态 C_CREDIT；

6) 在表 NEW-ORDER 和 ORDER 中插入新行，O_CARRIER_ID 设为 null，若本地供货，O_ALL_LOCAL=1，否则 O_ALL_LOCAL=0；

7) O_OL_CNT = *ol_cnt*，对于每种货物：

① 在表 ITEM 中找到行 I_ID (=OL_I_ID)，取出 I_PRICE，I_NAME,I_DATA，若找不到 I_ID，则事务回滚；

② 在表 STOCK 中找到行 S_I_ID (=OL_I_ID),S_W_ID (=OL_SUPPLY_W_ID)，提取 S_QUANTITY，S_DIST_xx，S_DATA；

若 S_QUANTITY-OL_QUANTITY>=10，S_QUANTITY=S_QUANTITY-OL_QUANTITY；

否则，S_QUANTITY=(S_QUANTITY-OL_QUANTITY)+91，

S_YTD = S_YTD + OL_QUANTITY，

S_ORDER_CNT = S_ORDER_CNT+1；

若此种货物是远程供货，则 S_REMOTE_CNT+1；

③ OL_AMOUNT = OL_QUANTITY * I_PRICE；

④ 检查字段 I_DATA 和 S_DATA，若都包含”ORIGINAL”，brand-generic = “B”；否则， brand-generic = “G”；

⑤ 在表 ORDER_LINE 中添加新行，OL_DELIVERY_D 设 null，OL_NUMBER=*ol_cnt*，OL_DIST_INFO 为 S_DIST_xx 的内容；

8) 订单总额：sum(OL_AMOUNT)*(1-C_DISCOUNT)*(1+W_TAX+D_TAX)。

9) 若没有回滚，事务提交,并在终端显示输出结果。

2. Payment 事务

数据输入：

- 仓库号(W_ID)：发起事务的终端对应的仓库号；
- 地区号(D_W_ID、D_ID)： D_W_ID=W_ID,random[1..10]；
- 客户(C_W_ID,C_D_ID,C_ID/C_LAST)：

  x,y = random[1..100]

① 若 x<=85，本地付款，C_D_ID=D_ID、C_W_ID=W_ID；

② 若 x>85，C_D_ID=random[1..10]，C_W_ID≠W_ID，随机选其他仓库；

③ 若 y<=60，以客户名字(C_LAST) 选择，C_LAST 由函数 NURand(255,0,999) 生成；

④ 若 y>60，以客户号(C_ID) 选择，C_ID 由函数 NURand(1023,1,3000)生成；

- H_AMOUNT= random[1.00 .. 5,000.00]；
- H_DATE 为系统当前时间。

事务过程：

1）给 SUT 传送生成数据：

仓库号(W_ID)，地区号(D_W_ID,D_ID)，客户号(C_W_ID,C_D_ID,C_ID) 或客户名字(C_W_ID,C_D_ID,C_LAST)，交付金额(H_AMOUNT)；

2）事务开始；

3）在表 WARHOUSE 中找到行 W_ID，取出

W_NAME,W_STREET_1,W_STREET_2,W_CITY,W_STATE,W_ZIP,并且 W_YTD 增加 H_MOUNT；

4）在表 DISTRICT 中找到行 (D_W_ID,D_ID)，取出 D_NAME,D_STREET_1,D_STREET_2,D_CITY,D_STATE,D_ZIP,并且 D_YTD 增加 H_AMOUNT；

5）以客户号选择：

在表 CUSTOMER 中选择行 (C_W_ID,C_D_ID,C_ID)，取出 C_FIRST,C_MIDDLE,C_LAST,C_STREET_1 ，

C_STREET_2, C_CITY, C_STATE, C_ZIP, C_PHONE, C_SINCE, C_CREDIT, C_CREDIM_LIM, C_DISCOUNT, C_BALANCE;

C_BALANCE 减少 H_AMOUNT, C_YTD_PAYMENT 增加 H_AMOUNT, C_PAYMENT_CNT 增加 1;

以客户名字选择:

在表 CUSTOMER 中选择符合(C_W_ID, C_D_ID, C_LAST)的所有行，并以 C_FIRST 的升序排列, 行数为 n, 找到第 n/2 行(向下取整);

取出 C_FIRST, C_MIDDLE, C_LAST, C_STREET_1, C_STREET_2, C_CITY, C_STATE, C_ZIP, C_PHONE, C_SINCE, C_CREDIT, C_CREDIM_LIM, C_DISCOUNT, C_BALANCE;

C_BALANCE 减少 H_AMOUNT, C_YTD_PAYMENT 增加 H_AMOUNT, C_PAYMENT_CNT 增加 1;

6) 若 C_CREDIT = "BC", 则将 C_ID, C_D_ID, C_W_ID, D_ID, W_ID, H_AMOUNT 插入到 C_DATA 左边, C_DATA 如果超出 500 字节，右侧抛弃溢出;。

7) H_DATA 插入 W_NAME+4 空格+D_NAME;

8) 表 HISTORY 中插入新行: H_C_ID=C_ID, H_C_D_ID=C_D_ID, H_C_W_ID=C_W_ID, H_D_ID=D_ID, H_W_ID=W_ID;

9) 事务提交;

10) 输出数据在终端显示。

3. Order-Status 事务

数据输入:

- 仓库号(W_ID) : 发起事务的终端对应的仓库号;
- 地区号(D_ID) : 区号(D_W_ID、D_ID): D_W_ID=W_ID, random[1..10];
- 客户(C_W_ID, C_D_ID, C_ID/C_LAST):

  y = random[1..100]

① 若 y<=60, 以客户名字(C_LAST) 选择, C_LAST 由函数 NURand(255, 0, 999) 生成;

② 若 y>60, 以客户号(C_ID) 选择, C_ID 由函数 NURand(1023, 1, 3000)生成;

事务过程:

1) 将输入数据传送到 SUT：C_W_ID, C_D_ID,C_ID;

2) 事务开始;

3) 以客户号选择客户：

在表 CUSTOMER 中选择行 C_W_ID,C_D_ID,C_ID, 取出 C_BALANCE,C_FIRST,C_MIDDLE,C_LAST;

以客户名字选择客户：

在表 CUSTOMER 中选择符合(C_W_ID,C_D_ID,C_LAST)的所有行，并以 C_FIRST 的升序排列，行数为 n，找到第 n/2 行(向下取整);

取出 C_BALANCE,C_FIRST,C_MIDDLE,C_LAST;

4) 在表 ORDER 中选择行

O_W_ID(=C_W_ID),O_D_ID(=C_D_ID),O_C_ID(=C_ID),和最大的 O_ID

取出 O_ID,O_ENTRY_D,O_CARRIER_ID 的值;

5) 在表 ORDER-LINE 中选择行

OL_W_ID(=O_W_ID),OL_D_ID(=O_D_ID),OL_C_ID(=O_ID)

取出相应的一组

OL_I_ID,OL_SUPPLY_W_ID,OL_QUANTITY,OL_AMOUNT,OL_DELIVERY_D 的值;

6) 事务提交;

7) 输出数据在终端显示。

4. Delivery 事务

数据输入：

- 仓库号(W_ID) ：发起事务的终端对应的仓库号;
- 批次号(O_CARRIER_ID) ：random[1..10];
- 发货日期(OL_DELIVERY_D)：当前系统时间;

事务过程：

1) 从等待队列中取出初始值：W_ID, (D_W_ID,D_ID), O_CARRIER_ID;

2) 在上一订单发货事务结束后，事务开始;

3) 在表 NEW-ORDER 中找到行 NO_W_ID(=W_ID),NO_D_ID(=D_ID)和最小的 NO_O_ID，取出 NO_O_ID 的值，若没有找到，则此区跳过，如果出现几率大于 1%，需报告。然后寻找下一个区的最早的未发货订单的 NO_O_ID 值;

4) 表 NEW-ORDER 中相应的行删除；

5) 在表 ORDER 中选择行 O_W_ID(=W_ID),O_D_ID(=D_ID),O_ID(=NO_O_ID)，取出 O_C_ID 的值，更新 O_CARRIER_ID；

6) 在表 ORDER-LINE 中选择行 OL_W_ID(=O_W_ID),OL_D_ID(=O_D_ID),OL_O_ID(O_ID)，更新 OL_DELIVERY_D 为系统当前时间，取出 OL_AMOUNT；

7) 在表 CUSTOMER 中选择行 C_W_ID(=W_ID),C_D_ID(=D_ID),C_ID(=O_C_ID)，C_BALANCE 增加所有订货总额 sum(OL_AMOUNT) C_DELIVERY_CNT 增加 1；

8) 若此事务中所有订单均准备发货，事务提交；

9) 发货订单的信息记录到结果文件中。

5. Stock-Level 事务

数据输入：

- 仓库号(W_ID)：发起事务的终端对应的仓库号 W_ID；
- 地区号(D_ID)：发起事务的终端的 D_ID；
- 查询最后 20 个订单订购的货物的库存水平；
- 最小库存量：random[10 .. 20]；

事务过程：

1) 将输入数据传送给 SUT；

2) 事务开始；

3) 在表 DISTRICT 找到行(D_ID,W_ID),取出 D_NEXT_O_ID 的值；

4) 在表 ORDER-LINE 中选择行 OL_W_ID(=W_ID),OL_D_ID(=D_ID),OL_O_ID(D_NEXT_O_ID > OL_O_ID > D_NEXT_O_ID-20)；

5) 在表 STOCK 中选择所有符合 S_I_ID(=OL_I_ID),S_W_ID(=W_ID)的行，并计算是否小于最小库存；

6) 提交当前事务；

7) 在终端显示输出数据。

3.5 执行规则和约束条件

3.5.1 执行规则

对于 TPC-C 中的测试数据库，可以进行以下一些执行操作，但同时这些操作也必须满足一定的执行规则：

允许横向分割基本表。基本表中的行组可以被分配到不同的文件、磁盘或者区域中。如果执行这一分割那么必须要解释区分的细节。

允许纵向分割基本表。基本表中的属性组(columns)可以被分配到不同于表的其他属性的存储域的文件、磁盘或者区域中。如果执行这一分割那么必须要解释区分的细节。

所有表格允许复制。所有的基本表的备份必须符合所有的要求，这些要求包括原子性、一致性和隔离性。如果执行那么复制的细节必须要解释。

当一些变化不会改善数据库性能的时候属性可以从一个基本表复制到另一个基本表。每一个属性必需逻辑上分散并且要独立的被访问。例如:W_STREET1 和 W_STREET_2 就不能把它们合并成一个属性 W-STREET 来访问。每一个属性必须要作为一个独立的属性被访问，例如:S-DATA 如果被分成两个属性 S_DATA_1 和 S_DATA_2 就不能被执行，这样做肯定会在产生异常。所有的关于时间日期的属性(例如:C_SINCE, H_DATE, O_ENTRY_D, and OL_DELIVERY_D)都可以作为一个日期属性和一个时间属性的合并值。

每个基本表的主键不能直接表示行的任何空闲的物理磁盘地址或它的任何偏移量。应用程序不能够引用相对地址行因为它们只是简单的相对于起始的存储位置有偏移。这并不排除在正常处理过程中有增加、删除和修改记录预备的哈希表或其他文件组织。

应用程序中执行的任何计算的最小精度必须是该计算中所有单独条款的最大精度。不管对数据库进行了何种额外的操作，它都必须满足数据库完整性的要求。完整性标准如下:

1 在任何可能的情况下，在各基本表中主键的值必须是唯一的。

2 在任何可能的情况下，不许有问题的行存在于数据库中。当任何一个属性的值不能被决定时说明数据库中存在着有问题的行。

3.5.2 事务混合百分比

菜单上的每种事务类型对每个终端都是可用的。在测量间隔内，终端必须对每种事务类型的混合产生一个最小的百分比，如表 3-1：

表 3-1 事务混合百分比

Table 3-1 The percentage of transaction mix

事务类型	混合的最小%
新订单[1]	n/a
支付	43.0
订单状态	4.0
发货	4.0
库存级别	4.0

[1]对新订单事务没有最小值，因为它的测量比率是报告的吞吐量

对每种事务类型的混合的最小百分比的目的是对每个新订单事务执行大约一个支付事务，对每 10 个新订单事务执行大约一个订单状态事务、一个发货事务和一个库存级别事务。这些混合导致每个订单的完整商业处理。

可从中导出混合的最小百分比的事务总数，包括从菜单中挑选的和测量结果内完成的所有事务。在此次测试中，主要采取了以下这种技术来随机选择事务类型，以满足了各种事务的最小混合比例。

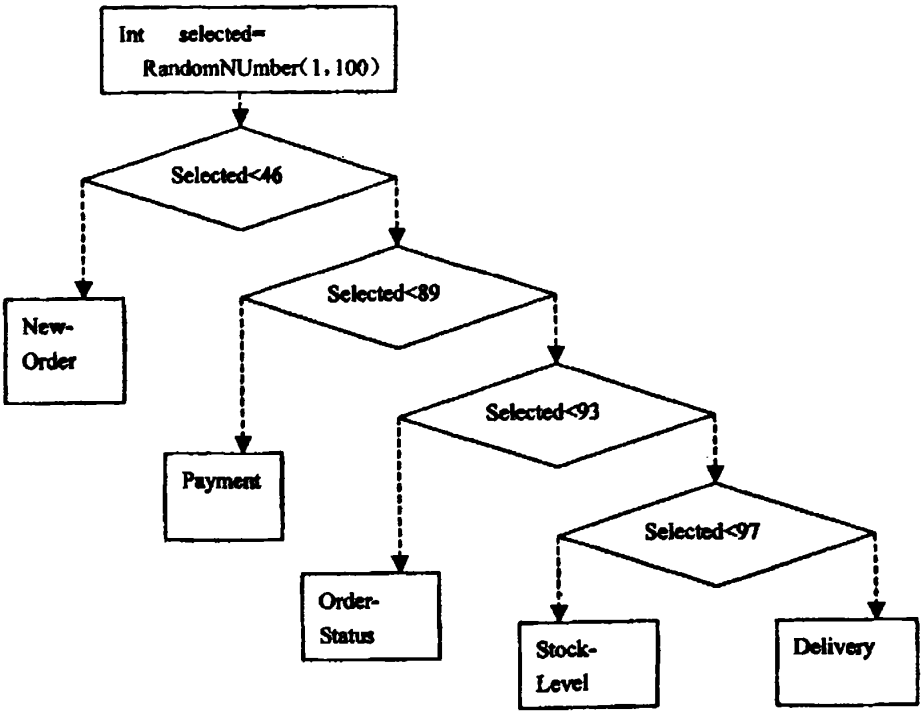

图 3-3 事务选取原理

Figure 3-3 The principle for transaction selection

如上图 3-3，将 1-100 的整数分成五段，代表五种事务类型：

$1 \leq Selected < 46$：	New-Order 事务
$46 \leq Selected < 89$：	Payment 事务
$89 \leq Selected < 93$：	Order-Status 事务
$93 \leq Selected < 97$：	Stock-Level 事务
$97 \leq Selected \leq 100$：	Delivery 事务

Selected 为 1-100 的整数中随机选择的一个数字，属于哪个数据段就执行相应的事务。

3.5.3 等待时间、响应时间和吞吐量

菜单响应时间菜单步骤是独立于事务的，所有菜单选项的至少 90%必须具有一个小于 2 秒的菜单反应时间。

对每种事务类型，按键时间是常数，且必须对新订单是最小值 18 秒，对支付是 3 秒，对订单状态、发货和库存级别均是 2 秒。

每种类型至少 90%的所有事务必须具有一个事务响应时间限制，它对新订单、支付、订单状态和发货均小于 5 秒，对库存级别小于 20 秒。

表 3-2 总结了事务混合、等待时间和响应时间的约束：

表 3-2 事务混合、等待时间和响应时间约束

Table 3-2 Transaction mix, wait times and response time constraint

事务类型	混合的最小百分比	最小按键时间	第 90 个百分点的响应时间约束	最小平均思考时间分布
新订单	n/a	18 秒	5 秒	12 秒
支付	43.0	3 秒	5 秒	12 秒
订单状态	4.0	2 秒	5 秒	10 秒
发货 [1]	4.0	2 秒	5 秒	5 秒
库存级别	4.0	2 秒	20 秒	5 秒

[1]：响应时间是对终端响应（事务排队）而言的，而不是对事务自身的执行而言的。事务的至少 90%必须在它们排队的 80 秒内完成。

吞吐量（MQTh），即 Maximum Qualified Throughput，是每分钟处理的订单数量，它测试的是事务吞吐量而非每分钟的事务执行效率。总的事务数要包括按各项事务的最小百分比从菜单中选择的所有事务，包括引起回滚的 New-Order 事务。TPC-C 报告中对系统吞吐量(MQTH)的度量是每分钟新增的新订单数，即 NOTPM，New-Order Transaction Per Minute，以 tpmC 表示。[16]

3.5.4 测试间隔

TPC-C 测试需保证在表示被测系统的真实的可持续吞吐量的稳定状态条件下进行。虽然测量间隔可以是 120 分钟，但是被测系统必须是能够在报告的 tpmC 基准上运行测试至少 8 小时的连续时间。例如，如果要求从任何单个的失败点恢复，那么必须配置用于存储至少 8 小时的日志数据的介质。

3.5.5 测试方法局限性

TPC-C 基准的目的是向工业用户提供相关的、客观的性能数据。为达到这个

目的，TPC 基准规范要求基准测试所反映得系统应该：

- 对用户而言总体上可用
- 与 TPC-C 基准模拟的商业模型相关联或相似
- 能够近似的实现模拟的大量用户

简而言之，用户使用 TPC-C 基准测试之前，应该考虑此基准模拟的商业应用环境是否适用于实际的业务环境，是否与实际市场环境相关，其相似度越高，测试结果参考值就越有价值，反之则相反。

本次测试集设计是针对 Linux 操作系统设计，其测试集的编译环境、SHELL 脚本的运行等均需要在 Linux 操作系统环境下实现，因而 不适用于 windows 操作系统环境。

3.6 本章小结

本章详细讲解了 TPC-C 基准测试的方案设计，首先介绍了测试系统是由 SUT（待测系统）、驱动系统和它们之间的通信接口构成的。其次，详细讲解了组成数据库的 9 张基本表的表结构，以及如何生成基本表的初始化数据，其中对每张表中字段值的生成方法都进行了一一描述，并简单介绍了测试集中代码的实现方法。再次，介绍了由 RTE（远程终端模拟器）模拟的虚拟用户发起事务操作的 7 个基本步骤，以及用户等待时间、菜单相应时间、事务相应时间如何度量；接着讲解了 TPC-C 中 5 个基本事务的具体处理过程，从生成发起事务需要提交的输入数据，到各表中相关数据的改变都作了详细介绍。最后，对执行 TPC-C 测试所要遵循的执行规则、测试过程中的约束条件以及测试方法的局限性作了阐述，约束条件中描述了定义的响应时间极限值，讲解了事务混合百分比控制方法、吞吐量的测量以及测试时间间隔要求。

第 4 章 TPC-C 测试实施

4.1 软硬件环境

此次数据库 TPC-C 基准性能测试，是基于红旗 Linux DC Server 5.0 操作系统上的 MySQL5.0 数据库 Linux 版本的性能测试。因为本人的工作是致力于全国产电子政务系统解决方案的测试与应用推广，因而选取了国产大品牌红旗 Linux 服务器操作系统。而选择开源数据库 MySQL5.0 的 Linux 版本作为数据库测试对象，是因为近年来国内发展比较快，已经趋于产品化的几家国产大品牌数据库厂商，如南大通用 GBase 数据库系统、人大金仓 KingBase ES 数据库系统等，是在完成对开源数据库内核实现机制的研究基础上，一步一步发展而来，最终形成了自己的产品。最初，主流开源数据库，如 MySQL，PostgreSQL 等，都是在 Linux 操作系统上实现。因而，大多数国产数据库也都是在 Linux 操作系统上研制实现，然后向 windows 操作系统上移植而形成其 windows 版本。在 Linux 操作系统上研究开源数据库的 TPC-C 基准测试集，有助于国产数据库在数据库 TPC-C 基准性能测试方面做参考，在应用环境和实现方式上，无须作较大改动，对国产数据库系统的 TPC-C 基准性能测试有一定的借鉴价值。

此次 TPC-C 测试集程序是针对国产操作系统上的开源数据库开发、设计的，测试程序主要采用标准的 C 为编写语言，同时通过 MySQL 数据库的提供的 C 接口访问数据，并用到了 SHELL 脚本语言。其软硬件环境具体如下：

操作系统：　红旗 Linux DC Server 5.0 操作系统

数据库：　MySQL 5.0.42 数据库

服务器：　Sun fire X2200

CPU：Dual-Core AMD Opteron(tm) 2210　1.8G

内存：2G

硬盘：SATA 160G

语言：　C 语言、SHELL 脚本语言

4.2 测试程序模块

我们可以将测试集简单的划分为 3 个模块：

4.2.1 数据装载模块

4.2.1.1 数据生成

数据生成模块(datagen)的主要功能是根据指定的数据规模(即 warehouse 个数) 根据 3.2.1 小节中的描述生成数据库的初始化数据到指定的文件夹中，生成的数据为.data 文件：

customer.data，district.data，history.data，item.data，new_order.data，order.data，order_line.data，stock.data，warehouse.data

其中每行中每个字段的值以“,”隔开，根据第 2 章中介绍，warehouse 数量为 1 时，将生成约 80M 左右的数据。

参数：

为了增强测试调整的灵活性，测试程序设置了一些调整参数，以供调整测试模式、规模等。具体的参数定义如下:

-w ：仓库数，调整此参数能够改变测试数据库的数据量

-c ：客户数，默认值为 3000

-I ：商品数，默认值为 100000

-o ：订单数，默认值为 3000

-n ：新增订单数，默认值为 900

-d ：生成数据文件到此路径下

4.2.1.2 数据导入

数据导入模块(build_db)主要功能包括建立数据库，建立 TPC-C 基准要求的 9 张表及相应索引，并将数据生成模块生成的初始化数据装载入建立的数据库中。装载数据的时间由仓库数(w)决定。

数据导入模块是通过 shell 脚本 build_db.sh 执行的。由于数据生成阶段已经生成了数据文件，每次初始化数据库数据时，只需利用 SHELL 脚本 build_db. sh 将生成好的数据导入到对应表中即可，build_db. sh 提供的“-w”参数指定生成数据的规模。build_db. sh 脚本中导入数据的实现方法为调用 MySQL 命令：

command_exec "$MYSQL *$DB_NAME* -e \"LOAD DATA $LOCAL INFILE \\\"*$DB_PATH/$FN.data*\\\" INTO TABLE *$TABLE* FIELDS TERMINATED BY '\t'\""

其中*$DB_NAME* 为导入数据库的名称，*$DB_PATH* 为生成数据文件时指定的文件夹地址，$FN.data 即调用*$DB_PATH* 中对应表的初始化数据文件，*$TABLE* 为导入的表的名称，与$FN.data 对应。$TABLE 循环取各表的名称，上述命令将依次将 9 张表的数据从.data 文件导入到各表中，生成 TPC-C 测试需要的初始化数据。

参数：

-d ： <database name> 装载的数据库名称

-f ： <path to dataset files> 待装载数据文件的路径

-s ： <database socket> 数据库套接字

-h ： <database host> 主机名

-u ： <database user> 数据库用户名

-p ： <database password> 数据库密码

-e ： <storage engine: [MYISAM|INNODB|BDB]. (default INNODB)> 数据库引擎，默认为 INNODB

-w ： <scale factor> 缩放因素，即仓库数

4.2.2 事务处理模块

事务处理是整个测试过程的关键模块，它模拟了整个批发商的主要活动，产生输入数据，处理事务，显示输出，同时记录相关信息，系统的性能由这个过程的运行情况反映。

Client 模块主要模拟终端客户进程，原则上是每个用户端同被测系统建立一个连接，并且使用这个连接来访问数据库。并且模拟虚拟用户思考、等待菜单、在界面输入字符提交事务的过程，输入字符的长度与事务类型要求的最小 keying time 时间有关。首先根据事务混合比选择一个事务，用户输入字符由函数 parse_command(*command)控制[37]，此次测试按键时间取规定的最小 keying time，时间输入字符 command 为"exit"或"quit"时，parse_command(command)返回 EXIT_CODE，则下面的语句退出循环，用户在终端结束输入，事务发送给 Driver 模块处理。

……

do {

```
                scanf("%s", command);
                if (parse_command(command) = = EXIT_CODE) {
                        break;
                }
        } while(1);
……
```

Driver 模块是模块的核心，处理模拟客户端提交的事务请求，产生输入数据，处理事务，将处理结果显示返回到模拟用户终端，同时记录系统运行情况，并将其它统计信息一并记入 log(日志)文件，整个过程无需人工干预。在保证付款、发货、订单查询、发货四个事务最小比例混合，响应时间在约束范围内，ACID 属性满足要求的基础上，将新增订单及其他事务信息统计在 log 文件中，得出 TPC-C 最终的结果 tpmC 值。

测试过程中一个重要的任务是确定系统稳定运行的时间，以确定开始测试的时间点。本此测试模拟用户连接的时间间隔(client_conn_sleep)设置为 1000ms，因而取所有用户均建立连接时为测试的开始点，因而其爬坡时间(threads_start_time)为：

```
threads_start_time = (int) ((double) client_conn_sleep / 1000.0 *
                            (double) terminals_per_warehouse *
                            (double) (w_id_max - w_id_min));
```

其中 terminals_per_warehouse 为每个仓库的终端数，即 10。w_id_max 为仓库主键的最大值，w_id_min 为仓库主键的最小值。若为 10 个仓库，则其爬坡时间为 100 秒。

测试停止时间即为当前时间+测试运行时间+爬坡时间。

```
stop_time = time(NULL) + duration + threads_start_time;
```

上面语句中 time(NULL)为系统当前时间，duration 为指定的测试运行时间，此次测试时间中指定为 2 小时。

4.2.3 后处理模块

后处理(post-process)模块是将事务运行过程中的信息合理的记录在文件中，进行解析后，将需要在报告中显示的信息存储到结果文件 result.out 中。

其主要记录以下信息：

1．记录订购、付款、订单查询、发货、库存查询的响应时间，用于计算各自的 90%响应时间和平均响应时间。

2．记录各终端五个事务执行的次数，用于计算事务的总数及各事务的混合比例。

3．记录订购事务中，回滚事务的数目，用于计算订购事务回滚的比例，证明其符合基准的要求。

4．测试过程中系统资源消耗，如 CPU 占用率，内存占用率，磁盘 I/O 读写块数等。

图 4-1 为仓库数为 10，数据量约为 1.06G，爬坡时间为 95 秒，测试时间为分 5.1 钟的一次 TPC-C 预测试结果记录。

		Response Time (s)				
Transaction	%	Average :	90th %	Total	Rollbacks	%
Delivery	4.07	0.226 :	0.487	59	0	0.00
New Order	44.68	0.052 :	0.111	647	6	0.93
Order Status	4.00	0.046 :	0.136	58	0	0.00
Payment	43.16	0.041 :	0.094	625	0	0.00
Stock Level	4.07	0.427 :	1.479	59	0	0.00

```
126.06 new-order transactions per minute (NOTPM)
5.1 minute duration
0 total unknown errors
95 second(s) ramping up
```

图 4-1 10 个仓库数的 TPC-C 预测试结果

Figure 4-1 The TPC-C pre-test result for 10 warehouses scale test

4.3 测试实施

此次测试采用逐渐增大数据量的分级测试方法。TPC-C 的数据量规模以 1 个仓库为跨度，增加仓库表中的一个仓库数，则其他表的数据库数据按比例增长。此次测试选用仓库数 N 为测试点，通过不断加大 N 的值，考察数据库在不断增加的大量用户并发数下的性能表现。此次测试 N 分别采用 10、20、30，即约为 800M、1.6G、2.4G 的数据量进行测试。为求测试结果更加真实可靠，更大程度的体现数据库的事务处理能力，我们还将仓库数增加到 40、50 个，即约 3.2G、4.0 数据量，最大的表中数据达 1500 万条左右，500 用户并发提交事务，测试 120 分钟的稳定运行时间。红旗 Linux DC Server 5.0 操作系统使用默认的内核参数，MySQL5.0.42 数据库的配置文件选用默认的 my-huge.cnf。

以上提到的数据量为数据库表中的数据规模，不包括其他数据存储及日志开销，因而实际测试中数据库的数据量会更大。

4.3.1 10仓库数据量测试

4.3.1.1 装载数据

在装载数据前需要2个准备条件，一是启动MySQL数据库，二是在/tmp目录下，根据此次测试的仓库数，我们分别建立dbt2-w10、dbt2-w20、dbt2-w30等根据仓库数命名的文件夹用以存放生成的相应数据量的数据文件。

然后在测试集src目录下，运行数据生成命令：./datagen -w 10 -d /tmp/dbt2-w10, 其他几个数据文件的生成与此类似，只需更改-w参数为相应得仓库数，-d参数为其对应的文件夹即可。

最后运行数据生成Shell脚本命令生成数据到MySQL的dbt2数据库中，生成10个仓库数据的命令如下：

./build_db.sh -d dbt2 -f /tmp/dbt2-w10 -s /tmp/mysql.sock

数据生成执行过程如下图4-1，其中，“-d”参数指定数据库为dbt2，“-f”参数后面的/tmp/dbt2-w10为生成的10个仓库数的数据文件的路径，“-s”参数指定数据库的socket连接的位置。

```
[root@localhost mysql]# ./build_db.sh -d dbt2 -f /tmp/dbt2-w10-s /tmp/mysql.sock
Loading of DBT2 dataset located in /tmp/dbt2-w10 to database dbt2.

DB ENGINE:      INNODB
DB SCHEME:      OPTIMIZED
DB HOST:        localhost
DB USER:        root
DB_SOCKET:      /tmp/mysql.sock

Creating table STOCK
Creating table ITEM
Creating table ORDER_LINE
Creating table ORDERS
Creating table NEW_ORDER
Creating table HISTORY
Creating table CUSTOMER
Creating table DISTRICT
Creating table WAREHOUSE

Loading table customer
Loading table district
Loading table history
Loading table item
Loading table new_order
Loading table order_line
Loading table orders
Loading table stock
Loading table warehouse
```

图 4-2 初始化数据库

Figure 4-2 Database initialization

数据初始化完成后，还需要导入各事务的存储过程，通过存储过程保证事务的 ACID 属性，运行命令如下：

./mysql_load_sp –d dbt2

其中“-d”参数指定数据库。

4.3.1.2 运行测试

生成数据完成后，可以运行 Mysql 命令进入数据库操作界面，用 select count(*) from *Table_Name*; 查看表中行数，*Table_Name* 代表各表名，各表的行数应当符合 2.4 小节中描述的数据初始化要求，而且应当满足 ACID 属性中的一致性要求。

在测试集目录下，运行测试 Shell 脚本开始进行测试：

./run_workload -c 20 -d 120 -w 10

-c 参数表示允许的事务连接数，-d 参数表示稳定运行的分钟数，在此期间内度量 tpmC 值，-w 参数表示仓库数。

4.3.1.3 测试结果及系统资源利用率

测试完成后，测试结果和相关统计数据会存放在测试集目录下名为 output 的目录中，此目录下，第一次的测试结果存放在名为 0 的目录中，随测试次数增加，目录名称数字递增，其下一个可用目录数字存放在 run_number 文件中。

在 result.out 文件中，我们可以看到测试的最终结果，如下图所示：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	4.05	0.128 :	0.232	1390	0	0.00
New Order	45.31	0.048 :	0.070	15536	147	0.96
Order Status	3.90	0.031 :	0.044	1337	0	0.00
Payment	42.71	0.031 :	0.031	14643	0	0.00
Stock Level	4.03	0.072 :	0.005	1382	0	0.00

126.88 new-order transactions per minute (NOTPM)
120.1 minute duration
0 total unknown errors
97 second(s) ramping up

图 4-3 10 个仓库数的 TPC-C 测试结果

Figure 4-3 The TPC-C test result for 10 warehouses scale test

根据统计 CPU 占用率如下图所示：

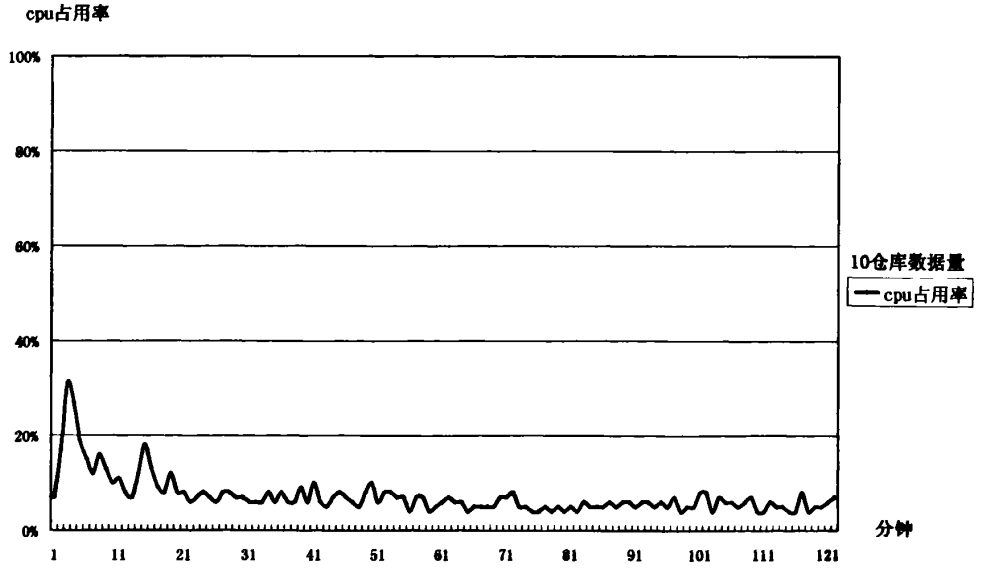

图 4-4 10 个仓库数测试的 CPU 占用率

Figure 4-4 The percentage of CPU use for 10 warehouses scale test

内存的占用情况如下图所示：

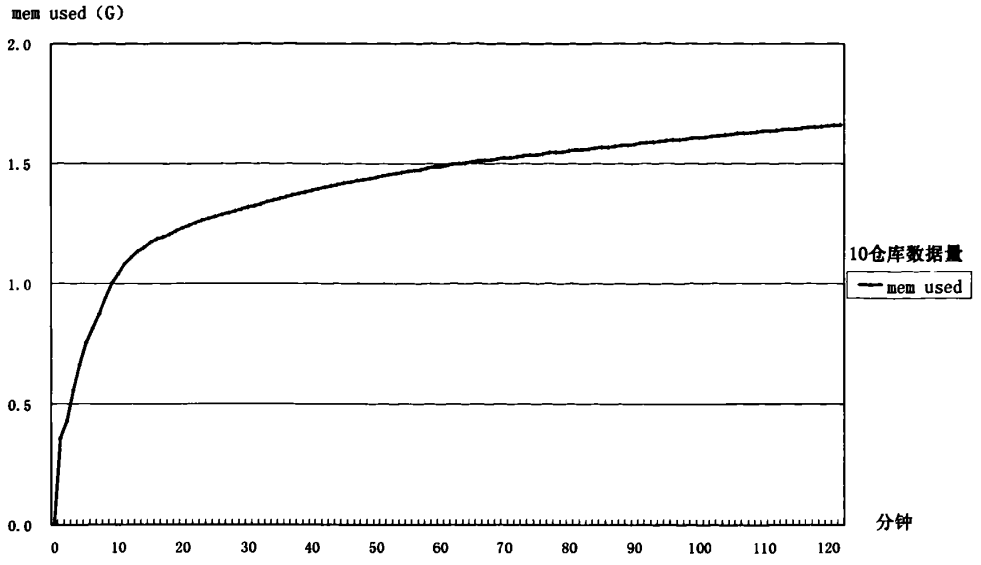

图 4-5 10 个仓库数测试的内存占用情况

Figure 4-5 The memory use for 10 warehouses scale test

从图 4-3、图 4-4 和图 4-5 来看，系统 CPU 占用率在 10%以下波动，系统内存占用缓慢的上升到 1.6G 左右，各事务的响应时间在 0.3 秒以下，最大的不超过 0.25 秒，系统负载较轻，可以承载更大的压力。测试结果文件显示出此次 TPC-C 测试的 tpmC 值为 126.88。

4.3.2 20 仓库数据量测试

10 仓库数测试完成之后，删除数据库 dbt2，重新导入 20 个仓库数的数据量，装载数据的方法参见 4.3.1 小节，再导入存储过程，并重新启动操作系统，以保持每次测试前的初始环境一致。

改变“-w”后面的参数为 20，运行./run_workload -c 20 -d 120 -w 20 命令测试 20 个仓库数的 TPC-C 测试。

20 个仓库数的 TPC-C 测试结果如下图所示：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	3.94	0.433 :	1.112	2687	0	0.00
New Order	45.18	0.294 :	0.887	30775	278	0.91
Order Status	4.06	0.154 :	0.448	2767	0	0.00
Payment	42.73	0.243 :	0.761	29102	0	0.00
Stock Level	4.08	0.366 :	1.043	2782	0	0.00

251.58 new-order transactions per minute (NOTPM)
120.1 minute duration
0 total unknown errors
197 second(s) ramping up

图 4-6 20 个仓库数的 TPC-C 测试结果

Figure 4-6 The TPC-C test result for 20 warehouses scale test

CPU 占用率如下图所示：

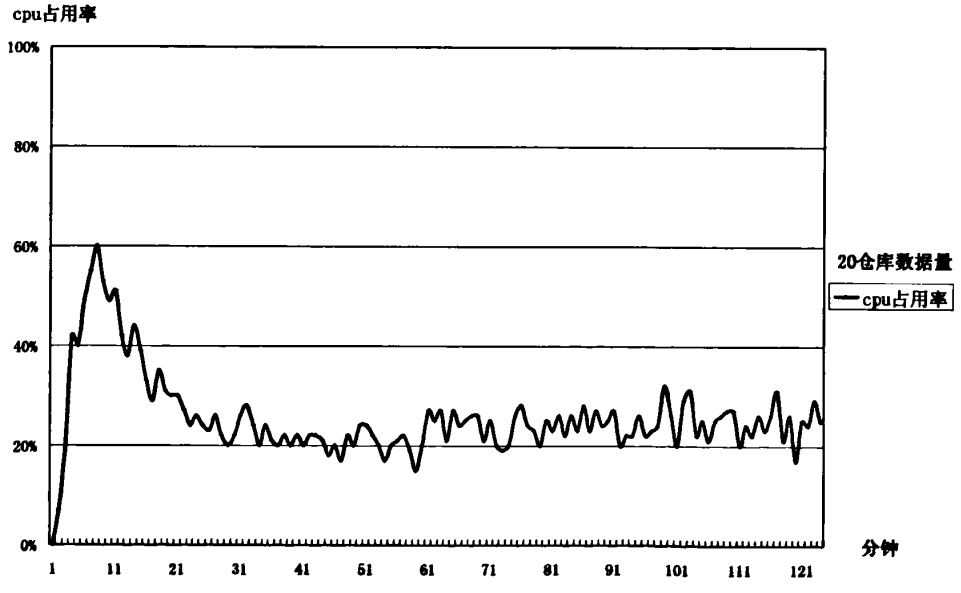

图 4-7 20 个仓库数测试的 CPU 占用率

Figure 4-7 The percentage of CPU use for 20 warehouses scale test

内存使用情况如下图所示：

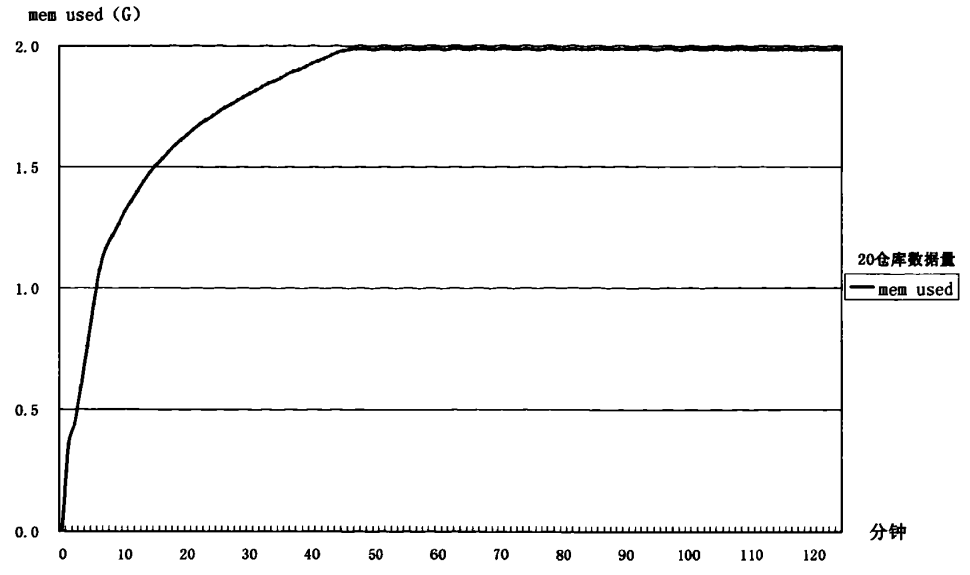

图 4-8 20 个仓库数测试的内存占用情况

Figure 4-8 The memory use for 20 warehouses scale test

其 CPU 占用率在 20%左右上下波动，内存占用达到了 1.9G，已经几乎耗尽了所有的内存资源，如图 4-6 可以看出各事务响应时间均在 1 秒左右，系统处于中等强度负载压力下工作。20 个仓库数据量的 TPC-C 测试为 251.58 tpmC。

4.3.3 30 仓库数据量测试

同样重复删除数据库中数据、装载数据、导入存储过程的步骤，初始化数据库完毕后，重新启动操作系统。改变“-w”参数为 30，运行命令./ run_workload -c 20 -d 120 -w 30，执行 30 仓库数的 TPC-C 测试。

30 个仓库数的 TPC-C 测试结果如下图所示：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	3.96	2.241 :	4.233	3753	0	0.00
New Order	45.43	1.988 :	3.790	43064	453	1.06
Order Status	3.92	1.286 :	2.217	3719	0	0.00
Payment	42.69	1.815 :	3.518	40467	0	0.00
Stock Level	4.00	2.031 :	3.533	3789	0	0.00

350.93 new-order transactions per minute (NOTPM)
120.1 minute duration
0 total unknown errors
294 second(s) ramping up

图 4-9 30 个仓库数的 TPC-C 测试结果

Figure 4-9 The TPC-C test result for 30 warehouses scale test

CPU 占用率如下图所示：

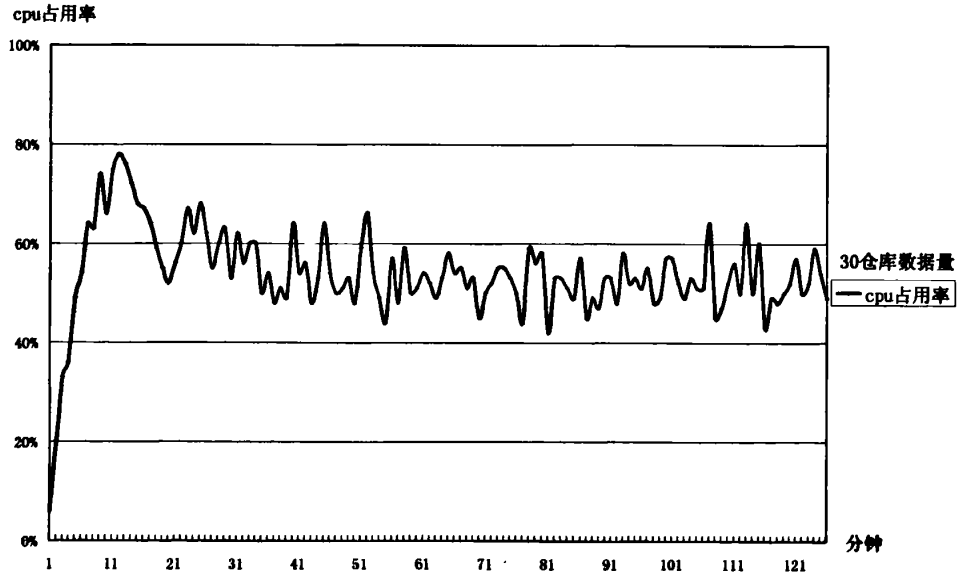

图 4-10 30 个仓库数测试的 CPU 占用率

Figure 4-10 The percentage of CPU use for 30 warehouses scale test

内存占用情况如下图所示：

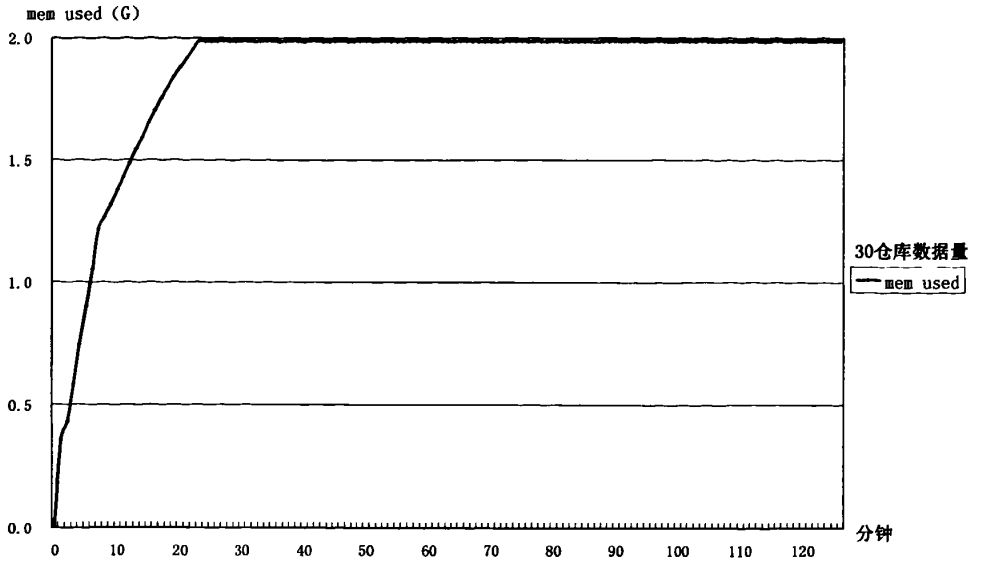

图 4-11 30 个仓库数测试的内存占用情况

Figure 4-11 The memory use for 30 warehouses scale test

由 CPU 和内存使用情况可以看出，此时系统负载较大，CPU 占用率在 50%左右，稳定运行 20 分钟后内存几乎全部被使用，系统事务响应时间虽然在 5 秒以内，但已经很接近 5 秒，刚刚满足了 TPC-C 基准规定的新订单、发货、订单状态查询、支付事务响应时间在 5 秒以内的约束条件。

4.3.3 30 个以上仓库数据量测试

为追求数据库系统的最大性能，我们把仓库数量提高到 40、50，在相同环境下，对系统进行更大压力的 TPC-C 基准性能测试，测试的结果如下。

40 个仓库数的 TPC-C 测试结果：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	3.92	21.550 :	34.651	2729	0	0.00
New Order	45.51	20.927 :	34.637	31681	316	1.01
Order Status	3.94	19.001 :	31.218	2746	0	0.00
Payment	42.65	20.504 :	33.794	29687	0	0.00
Stock Level	3.97	20.256 :	34.544	2766	0	0.00

258.45 new-order transactions per minute (NOTPM)
120.1 minute duration
0 total unknown errors
396 second(s) ramping up

图 4-12 40 个仓库数的 TPC-C 测试结果

Figure 4-12 The TPC-C test result for 40 warehouses scale test

50 个仓库数的 TPC-C 测试结果：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	3.92	45.260 :	58.958	2191	0	0.00
New Order	45.35	44.449 :	58.741	25335	228	0.91
Order Status	4.02	42.360 :	57.298	2244	0	0.00
Payment	42.79	43.954 :	58.583	23905	0	0.00
Stock Level	3.93	44.052 :	59.898	2193	0	0.00

207.04 new-order transactions per minute (NOTPM)
120.2 minute duration
0 total unknown errors
496 second(s) ramping up

图 4-13 50 个仓库数的 TPC-C 测试结果

Figure 4-13 The TPC-C test result for 50 warehouses scale test

由上两幅图我们可以看出，40、50 仓库数的 TPC-C 测试结果中，90%响应时间达到了 30 多秒和 50 多秒，tpmC 值为 258.45 和 207.04，比 30 仓库数据量的 tpmC 值 350.93 小很多，系统超负荷运转，性能下降，因而必须对数据库系统进行优化，以达到 TPC-C 关于事务响应时间的约束条件。

打开/etc 目录下 mysql 的全局参数文件 my.cnf，更改对 mysql 性能影响最大的 query_cache_size 等参数，具体更改如下：

query_cache_size = 64M [32M]

innodb_buffer_pool_size = 1228 MB [384]

innodb_log_file_size = 307 MB [100]

innodb_flush_log_at_trx_commit = 0 [1]

thread_concurrency = 4 [8]

等号后为更改后的数值，中括号内的为原来默认的数值。再次初始化数据库，重起系统，执行测试，得出 30、40、50 仓库数据量的 TPC-C 测试结果分别为：

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	3.94	2.382 :	3.727	3880	0	0.00
New Order	45.64	1.070 :	2.459	44923	469	1.06
Order Status	3.89	1.113 :	2.594	3827	0	0.00
Payment	42.64	0.742 :	1.386	41974	0	0.00
Stock Level	3.89	1.126 :	1.943	3832	0	0.00

```
366.08 new-order transactions per minute (NOTPM)
120.2 minute duration
0 total unknown errors
295 second(s) ramping up
```

图 4-14 优化后 30 个仓库数的 TPC-C 测试结果

Figure 4-14 The TPC-C test result for 30 warehouses scale test after tuning

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	4.00	4.150 :	4.303	5161	0	0.00
New Order	45.41	1.517 :	3.005	58615	615	1.06
Order Status	3.97	1.536 :	3.073	5118	0	0.00
Payment	42.59	1.024 :	1.899	54968	0	0.00
Stock Level	4.04	2.027 :	4.259	5210	0	0.00

```
477.68 new-order transactions per minute (NOTPM)
120.1 minute duration
0 total unknown errors
397 second(s) ramping up
```

图 4-15 优化后 40 个仓库数的 TPC-C 测试结果

Figure 4-15 The TPC-C test result for 40 warehouses scale test after tuning

Transaction	%	Response Time (s) Average :	90th %	Total	Rollbacks	%
Delivery	4.02	21.605 :	34.287	4573	0	0.00
New Order	45.71	10.738 :	18.749	51980	531	1.03
Order Status	3.86	10.800 :	18.633	4394	0	0.00
Payment	42.46	9.795 :	17.533	48285	0	0.00
Stock Level	3.94	12.689 :	22.788	4478	0	0.00

423.67 new-order transactions per minute (NOTPM)
120.2 minute duration
0 total unknown errors
496 second(s) ramping up

图 4-16 优化后的 50 个仓库数的 TPC-C 测试结果

Figure 4-16 The TPC-C test result for 50 warehouses scale test after tuning

由图中可以看出，30、40 个仓库数据量测试的 90%事务响应时间均在 TPC-C 基准约束的 5 秒范围内，当数据量加大到 50 个仓库数时，响应时间再次上升到 30 秒左右，事务混合比、回滚率等其他指标正常。可见优化后的 Mysql 数据库能够负载 40 仓库数据量的 TPC-C 测试，其测试结果为 477.68 tpmC，比优化前性能有了很大的提高，我们把优化前后的 tpmC 值归纳在下表。

表 4-1 TPC-C 测试结果

Table 4-1 The TPC-C test result

仓库数	tpmC	tpmC(tuning)
10	126.88	-
20	251.58	-
30	350.93	366.08
40	258.45	477.68
50	207.04	423.67

其变化曲线如下图所示：

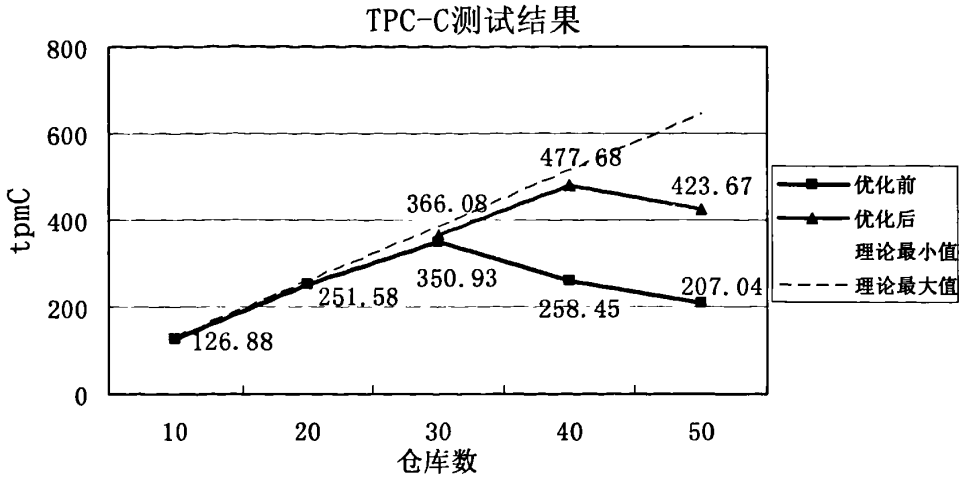

图 4-17 TPC-C 测试结果

Figure 4-17 The TPC-C test result

4.4 基准符合性分析

根据 4.3 小节中的测试结果图，我们从下面几个方面进行具体分析：

1. 响应时间(Response Times)

优化后的 40 个仓库数的 TPC-C 测试的响应时间如下表所示：

表 4-2 优化后 40 仓库数据量测试的响应时间

Table 4-2 The response times for 40 warehouses scale test after tuning

Transaction	Average	90th %	≤ %
Delivery	4.150	4.303	5
New Order	1.517	3.005	5
Order Status	1.536	3.073	5
Payment	1.024	1.899	5
Stock Level	2.027	4.259	20

由上表可以看出，上面的各种事务 90%响应时间均比其限制的最大的响应时间要小，满足 TPC-C 基准关于事务响应时间的约束要求。从事务 90%响应时间可以看出，事务响应时间随仓库数的增加，不断增大，系统负载不断增大的情况

下，事务处理时间也逐渐升高。40 个仓库数的数据规模测试中，响应时间在 4 秒左右，系统处于较高负载压力下工作，同时也证明了 MySQL 数据库可以承载 40 仓库数据量的 TPC-C 基准性能测试压力。

2. 事务混合比例(Transaction Mix)

以优化后的 40 个仓库数的测试为例，测试过程中，各种事务的混合比如下：

New-Order	45.41 %
Payment	42.59 %
Order_Status	3.97%
Delivery	4.00%
Stock-Level	4.04%

TPC-C 基准关于事务混合比例要求为 Payment>43%, Order-Status>4%, Delivery>4%,Stock-level>4%，但允许在测试过程中系统对事务比例的调整，但调整应在标准值的 5%范围之内。在上面的事务混合比例结果中，混合比均在标准值 5%范围内，因而符合标准的约束要求。40 个仓库数以下的测试事务混合比与之相近似，同样也符合标准规范要求，其 tpmC 值同样有效。

3. 吞吐量(MQTH)

由测试结果可以得出优化后的 40 个仓库数的测试过程中的总事务数为 129072，其中新增订单数量为 58615，测试的稳定运行时间为 120 分钟。回滚事务数为 615 个，回滚率为 1.06%，符合 TPC-C 标准的基准规定的 1%的事务回滚率。

TPC-C 报告中对系统吞吐量(MQTH)的度量是每分钟新增的新订单数，以 tpmC 表示。

在本文描述的环境下，TPC-C 测试结果 tpmC 值为：477.68。

TPC-C 测试中的另外一个性能指标是性价比(prieP/eeribmrnace，简称 price/tpmC),该指标为测试系统价格(指以美元的报价)与吞吐量的比值，即获得一个 tpmC 在硬件系统上的花费情况。性价比越小越好，该指标通常用来评测一种

硬件或服务器硬件系统的整体性价比情况。

在本文进行的红旗 Linux 操作系统上 MySQL 数据库系统测试中，采用的是以 tpmC 作为系统测试结果的指标。

4．结果对比分析

(1) 与其他测试集的测试结果对比

为给此次测试的结果一个定量的比较，将基于 TPC-C 基准的其他测试集的测试结果与本次测试的测试结果进行了对比分析，以验证此次测试结果的合理性和可信性。当然，TPC-C 的测试结果受系统整体环境影响，此次对比尽量选取环境相似的其他 TPC-C 测试结果，增加可比性。

在麒麟操作系统上对 Oracle 数据库系统的 TPC-C 基准性能测试，其硬件环境为：2*Xeon 2.4GHz CPU、2* 37GHz Ultra 320 SCSI 硬盘、2GB 内存，与本论文采用的测试环境基本相似，其 31 个仓库数的 TPC-C 基准测试得出的测试结果为：354.7tpmC。本论文实施的 30 个仓库数的 TPC-C 基准测试结果为：350.93，与 Oracle 测试的结果相差仅 3.77tpmC。[42]

现有商业化操作系统 wndiows2000 SERVER 上对 Oracle 数据库系统进行的 TPC-C 测试，测试硬件环境为：2*Xeon 2.4GHz CPU、2* 37GHz Ultra 320 SCSI 硬盘、2GB 内存，与本论文采用的测试环境基本相似，测试系统工作负载 WAREHOUSE 数量为 30 个，测试结果为 343.23tpmC 的测试值，与本论文 30 个仓库数的 TPC-C 测试结果 350.93 相当。[42]

(2) 与理论值对比

为防止数据量过度扩张，TPC-C 规定每仓库数其 tpmC 有一个理论最小值为 9，事务响应时间为极小值 0 时，得到每仓库数其 tpmC 的理论最大值为 12.86。

因而我们可以得到 tpmC 理论范围为：$9 \leqslant NOTPM / \#W \leqslant 12.86$

其中 NOTPM，即 New-Order Transactions per minute，每分钟处理的新增订单数，也就是测试结果的 tpmC 值。#W 为仓库数，测试仓库数为 40，则#W 为 40。

因而 40 仓库数的 tpmC 理论范围为[360，514.4]，此次测试 tpmC 值为 477.68，在理论范围之内，并显示出较好的性能表现。

4.5 本章小结

本章主要描述了测试的具体实施过程，首先介绍了此次测试用到的软硬件环境，包括操作系统、数据库、服务器及设计过程中使用的编程语言。其次对测试程序的三大模块分别进行了讲解，数据装载模块主要是讲述初始化数据的生成和导入；事务处理模块是测试的关键模块，它负责处理模拟客户端提交的事务请求，系统的性能由这个过程的运行情况反映；后处理模块是将事务运行过程中的信息合理的记录在文件中，并将需要的信息在结果文件中显示出来。接着，以仓库数 N 为测试点，通过不断加大 N 的值，考察数据库在不断增加的数据量和大量用户并发下的性能表现，对测试的具体实施过程进行了详细讲解，从装载数据到运行测试，再到最后测试结果和系统资源利用率的查看，其需要运行的命令和提供的操作参数均作了详细说明，并对测试系统从 CPU 占用率、内存使用情况等方面进行了性能分析，随着仓库数的增加系统性能下降并超出了 TPC-C 基准约束条件后，对 Mysql 数据库进行了优化，再次以 30、40、50 仓库数的数据量作测试，对优化前后的测试结果作对比分析，确定数据库在相应系统环境下能够承受的最大压力。最后对测试结果进行了基准符合性分析，将响应时间和各事务混合比与基准中规定的极限值进行了对比，计算系统在测试稳定运行过程中的吞吐量，得出最终的测试结果 tpmC 值。最后，还将测试结果 tpmC 值与其他测试集在相似测试环境下的 tpmC 值进行了比较，而且与 TPC-C 理论值进行了对比，确定测试结果的有效性。

结论

本文在红旗Linux DC Server 5.0操作系统上对MySQL 5.0数据库进行TPC-C测试集的设计过程中，依据 TPC-C 的标准规范对数据库表结构、表数据进行初始化，严格按照 TPC-C 对五种事务的要求进行事务处理过程代码的编写，设置合适的测试爬坡时间和稳定运行测试时间间隔，采用分级测试的方法设计测试方案，不断扩大测试数据规模来测试系统性能水平。从测试结果分析中我们可以得出如下结论：

1．在红旗 Linux DC Server 5.0 操作系统上对 MySQL 5.0 数据库系统的进行不同数据量的 TPC-C 基准测试证明 MySQL 数据库系统在采用默认的 my-huge.cnf 配置文件时，能承载 30 仓库数据量的测试压力，其测试结果为 350.93 tpmC。

2．当仓库数到达 40 和 50 时，系统性能明显下降，测试结果也不符合 TPC-C 基准约束。对系统 CPU 占用率和内存使用情况的分析，我们可以看出，CPU 占用并非系统性能瓶颈，随着数据量增大，内存需求明显增大，cache 缓存需求是影响性能的主要因素；

3．在对 MySQL 配置文件中 query_cache_size 等参数值进行优化后，系统能够负载 40 仓库数据量，tpmC 值为 477.68，各指数均在 TPC-C 规范中规定的界限值内，满足 TPC-C 各约束条件限制；

4．本论文测试结果与其他 TPC-C 测试集在相似环境下的测试结果对比的相似性，以及与 TPC-C 理论值的对比结果表明了本论文对 MySQL 数据库的测试结果合理、真实、可信。

5．此测试集能够根据仓库数快捷的生成相应得测试数据并方便导入，爬坡时间和稳定运行测试时间设置合理，测试过程中对 TPC-C 基准规定的各种约束指标和系统运行信息进行了统计和分析处理，方便分析系统得性能瓶颈，确定测试结果的基准符合性。

总体说来，在国产 Linux 操作系统上对开源 MySQL 数据库进行的 TPC-C 测试证明了此 TPC-C 测试集是一套符合 TPC-C 规范的数据库基准性能测试工具，能够对 MySQL 进行不同的数据规模的 TPC-C 基准测试，而且此测试集对系统资源和性能指标的记录易于测试者对测试结果的分析。但此测试集也存在不足之

处，缺少图形化监控功能，使操作者不能实时的了解测试的情况，无法即时地查看系统运行情况。另外，测试集中也缺少了对数据库系统的 ACID 属性的自动测试功能。

鉴于此测试集的设计初衷是为了有助于国产数据库在国际标准方面的性能测试，促进国产数据库测试尽快与国际标准接轨，因而，在今后的工作中也会基于本论文的设计着重对国产主流数据库在国产 Linux 操作系统上运行的 TPC-C 测试集进行开发。由于主流国产数据库大部分由对开源数据库内核实现机制的研究发展而来，相信借鉴于此论文对开源数据库的 TPC-C 测试设计，针对国产数据库的 TPC-C 基准测试设计也不是难事，也希望此课题能够为国产数据库性能测试方面作些小小的贡献。

另外，需要强调的是，“国际通用”的度量可以作为参考值，而不应作为必要条件。尤其是一定要弄清这些流行度量的含义，是在什么样的系统环境中测得的，以及基准程序是否符合企业真实的业务流程和运作模式。中国的行业应该建立符合自己实际应用的基准程序和测试标准，建立独立的测试中心，制定符合中国企事业运作模式的性能测试标准，才能更好的助力中国软件行业甚至整个产业的发展。

参考文献

1 Diego R. Llanos, Belen Palop. An Open-Source TPC-C Implementation for Parallel and Distributed Systems, 2006

2 Transaction Processing Performance Council (TPC). TPC BENCHMARK™ C Standard Specification Revision 5.7, April 2006

3 萨师煊，王珊．数据库系统概论．第2版．高等教育出版社，1991

4 KevinK line, DanielKl ine．SQ L技术手册．中国电力出版社，2003

5 周之英，郑人杰．计算机软件测试技巧．清华大学出版社，1995:145-187

6 S T Leutenegger, D M Dias. A modeling study of the TPC-C Benchmark TMC Standard Specification Revision 3.5. New York: Morgan Kaufmann Publisher Inc., 2002:1-130

7 Transaction Processing Performance Council (TPC). TPC BENCHMARK™ C Standard Specification Revision 5.8.0, December 2006

8 L.Barroso, K.Gharachorloo and E.Bugnion. Memory system characterization of commercial workloads, In Proc. of the 25th Intl. Symposium on Computer Architecture(ISCA), Jun 1998

9 J.L.Lo,L.A.Barroso, S.J.Eggers, K.Gharachorloo, H.M.Levy. An analysis of database workload performance on simultaneous multithreaded processors,Proc.of the 25th I SCA, Jun 1998

10 Highleyman W.H. Performance Analysis of Transaction Processing Systems, Prentice Hall Inc.,1998

11 Yao S.B., Hevner A.R., Myers H.Y. Analysis of Database System Architectures Using Benchmarks, IEEE Transactionson Software Engineering, Jun 1987

12 Biton D., DeWit D.J, C.Turbyfil. Benchmarking Database Systems: A Systematic Approach,Proceedings of the 1983 Very Large Database Conference, Oct 2001

13 H.Boral, D.DeWitt. A methodology for database system performance valuation, Proc. of the 1984 SIGMOD Conference, June 2002

14 高文．服务器聚集系统中高可用性分析与设计方法.博士学位论文．中国科学

院计算技术研究所，2001.6

15 林闯．计算机网络和计算机系统的性能评价．清华大学出版社，2001:4,76 页

16 计算机世界专题报告．TPC 基准程序及 tpmC 值．中国计算机世界出版服务公司，2004

17 Transaction Processing Performance Council. TPC Benchmark C Standard Specification Rev.5.1, Oct 2002.

18 谷长勇．事务处理服务器的性能评价研究．中国科学院计算技术研究所，2001.12

19 Transaction Processing Performance Council. TPC BENCHMARK R (Decision Support) Standard Specification, Rev.1.1.0, Jun 1999.

20 Transaction Processing Performance Council. TPC BENCHMARK H (Decision Support) Standard Specification, Rev.1.1.0, Jun 1999.

21 Transaction Processing Performance Council. TPC BENCHMARK W (Web Commerce) Specification,Rev.1.1,Jun 2000.

22 Patrick O'Neil. A set query benchmark for large databases. Technical Report, 1989.

23 Patrick O'Neil. The Set Query Benchmark. The Benchmark Handbook for Database and Transaction Processing Systems, Jim Gray edit, Morgan Kaufmann, 2nd Ed. 1993.

24 H.Reza Taheri. An Analysis of the Neal Nelson Business Benchmark TM, Performance Evaluation Review, vol.18, No.3, Nov 1990.

25 S.Leutenegger, D.Diaz. A Modeling Study of the TPC-C Benchmark, Proceedings of the ACM SIGMOD International Conference on Management of Data, May 2003.

26 J.Piantedosi et al. Performance of TruCluster systems under the TPC-C benchmark, Digital Technical Journal, 2006:46--57

27 Kohler, W ., Shah, A., Raab, F. Overview of TPC Benchmark C: The Order-Entry Benchmark, Transaction Processing Performance Council, Dec, 1991.

28 黄恺，徐志伟．可扩展并行计算技术、结构和编程．机械工业出版社，2000

29 谭浩强．C 程序设计(第二版)．化学工业出版社印刷厂，1999

30 沈佩娟，汤荷美．数据库管理及应用开发．清华大学出版社，1995

31 晏子，黄杰湘．MySQL 中文参考手册．devoinfo 网络工作室，2002

32 BittonD, DeWitt D.J, C. Turbyfil. Benchmarking Database Systems: A Systematic Approach, Proceedings of the 1983 Very Large Database Conference, Oct 2003

33 Dina Biton, Carolyn Turbyfill. A retrospective on the Wisconsin benchmark, Michael Stonebraker, Readings in Database Systems. Morgan Kaufmann, second edition,2004.

34 K.Keeton. Computer architecture support for database applications, PhD dissertation, Univ. of California at Berkeley,July 1999.

35 Neil Matthew．Linux 高级编程．机械工业出版社，2002.1

36 Jim Gray. The Benchmark Handbook for Database and Transaction Processing Systems, Second Edition, Morgan Kauftnann Publishers, Inc., 1993.

37 徐千洋．Linux C 函数参考手册．中国青年出版社，2002

38 Fabio Avila. Performance Comparison between MySQL and PostgreSQL, Linux World conference & expo, 2006.2

39 Patrick O 'Neil E lizabeth O 'Neil．数据库原理，编程与性能(第二版)．机械工业出版社，2002.6

40 朱正华．DM3 的 TPC-C 性能测试研究．华中科技大学硕士学位论文．2002

41 Michael Kofler. MySQL5 权威指南．人民邮电出版社，2006.12

42 郭海峰．银河麒麟操作系统上基于 TPC-C 的 Oracle 调优研究与实现．国防科技大学工学硕士学位论文．2005：26，59

附录

1. 表结构

术语定义：

N unique IDs：N个唯一的ID，意味着属性必须能具有N个唯一的ID最小集中的任何一个ID，而不考虑属性的物理表示（例如，二进制、打包的十进制、字母等）。

Variable text, size N：可变的文本、大小N，意味着属性必须能具有最大长度为N的变化长度的任何字符串。如果属性被存储为一个固定长度的串，且它具有的串比N个字符短，那么它必须用空格填充。

Fixed text, size N：固定的文本、大小N，意思是属性列必须能支持固定长度为N的字符串。

date and time：日期和时间，意思是属性列必须能支持在之间的至少精确到秒的日期时间。

numeric, N digits：数字型，N个数字，意思是属性列必须能支持任何小数位数为N的值。数字域包含的货币单位(W_YTD, D_YTD, C_CREDIT_LIM, C_BALANCE, C_YTD_PAYMENT, H_AMOUNT, OL_AMOUNT, I_PRICE)必须能用数据类型给出精确的表达，要求精确到当前执行基准所规定的流通的最小货币单位。例如美元中的C_BALANCE可以表达(12.2)带符号的十进制位(带有固定的缩放比例)、精确到41bit的分币大小的整数或者精确到双精度实数64bit的分币。

Null：意思是给定属性的合法范围之外的值，且对此属性总是相同值。

Warehouse表（仓库）

域名	域定义	注释
W_ID	2*W个唯一的ID	有W个仓库
W_NAME	可变文本，大小为10	
W_STREET_1	可变文本，大小为20	
W_STREET_2	可变文本，大小为20	

W_CITY	可变文本，大小为 20	
W_STATE	固定文本，大小为 2	
W_ZIP	固定文本，大小为 9	
W_TAX	数字型，4 个数字	销售税
W_YTD	数字型，12 个数字	年对日期的收支平衡

主关键字：W_ID

District 表（地区）

域名	域定义	注释
D_ID	20 个唯一的 ID	每个仓库有 10 个
D_W_ID	2*W 个唯一的 ID	
D_NAME	可变文本，大小为 10	
D_STREET_1	可变文本，大小为 20	
D_STREET_2	可变文本，大小为 20	
D_CITY	可变文本，大小为 20	
D_STATE	固定文本，大小为 2	
D_ZIP	固定文本，大小为 9	
D_TAX	数字型，4 个数字	销售税
D_YTD	数字型，12 个数字	年对日期的收支平衡
D_NEXT_O_ID	10,000,000 个唯一的 ID	下一个可用的订单号

主关键字：（D_W_ID,D_ID）

D_W_ID 外关键字，引用 W_ID

Customer 表（客户）

域名	域定义	注释
C_ID	96,000 个唯一的 ID	每个区域有 3,000 个
C_D_ID	20 个唯一的 ID	
C_W_ID	20 个唯一的 ID	
C_FIRST	可变文本，大小为 16	
C_MIDDLE	固定文本，大小为 2	
C_LAST	可变文本，大小为 16	

C_STREET_1	可变文本，大小为 20	
C_STREET_2	可变文本，大小为 20	
C_CITY	可变文本，大小为 20	
C_STATE	固定文本，大小为 2	
C_ZIP	固定文本，大小为 9	
C_PHONE	固定文本，大小为 16	
C_SINCE	日期和时间	登记日期
C_CREDIT	固定文本，大小为 2	“GC”=good，“BC”=bad
C_CREDIT_LIM	数字型，12 个数字	透支限额
C_DISCOUNT	数字型，4 个数字	折扣
C_BALANCE	有符号数字型，12 个数字	欠款余额
C_YTD_PAYMENT	数字型，12 个数字	累计付款金额
C_PAYMENT_CNT	数字型，4 个数字	累计付款次数
C_DELIVERY_CNT	数字型，4 个数字	累计发货次数
C_DATA	可变文本，大小为 500	混杂信息

主关键字：（C_W_ID，C_D_ID，C_ID）

（C_W_ID，C_D_ID）外关键字，引用（D_W_ID，D_ID）

History 表（历史纪录）

域名	域定义	注释
H_C_ID	96,000 个唯一的 ID	
H_C_D_ID	20 个唯一的 ID	
H_C_W_ID	2*W 个唯一的 ID	
H_D_ID	20 个唯一的 ID	
H_W_ID	2*W 个唯一的 ID	
H_DATE	日期和时间	
H_AMOUNT	数字型，6 个数字	付款额度
H_DATA	可变文本，大小为 24	混杂信息

没有主关键字

（H_C_W_ID，H_C_D_ID，H_C_ID）外关键字，引用（C_W_ID，C_D_ID，

C_ID)

(H_W_ID, H_D_ID) 外关键字，引用 (D_W_ID, D_ID)

注释：历史表中的行没有主关键字，因为在基准的上下文中，不需要唯一确认表中的一行。

注意：TPC-C 应用程序不需要利用 C_ID 值超过 6,000 的增加范围。

New_Order 表（新订单）

域名	域定义	注释
NO_O_ID	10,000,000 个唯一的 ID	
NO_D_ID	20 个唯一的 ID	
NO_W_ID	2*W 个唯一的 ID	

主关键字：(NO_W_ID, NO_D_ID, NO_O_ID)

(NO_W_ID, NO_D_ID, NO_O_ID) 外关键字，引用 (O_W_ID, O_D_ID, O_ID)

Order 表（订单）

域名	域定义	注释
O_ID	10,000,000 个唯一的 ID	
O_D_ID	20 个唯一的 ID	
O_W_ID	2*W 个唯一的 ID	
O_C_ID	96,000 个唯一的 ID	
O_ENTRY_D	日期和时间	下单时间
O_CARRIER_ID	10 个唯一的 ID，或 null	货运代号
O_OL_CNT	5 到 15	订单行的数目
O_ALL_LOCAL	数字型，1 个数字	是否全部本地供货

主关键字：(O_W_ID, O_D_ID, O_ID)

(O_W_ID, O_D_ID, O_C_ID) 外关键字，引用 (C_W_ID, C_D_ID, C_ID)

Order_Line 表（订单分录）

域名	域定义	注释
OL_O_ID	10,000,000 个唯一的 ID	
OL_D_ID	20 个唯一的 ID	
OL_W_ID	2*W 个唯一的 ID	

OL_NUMBER	15 个唯一的 ID	分录代码
OL_I_ID	200,000 个唯一的 ID	商品代码
OL_SUPPLY_W_ID	2*W 个唯一的 ID	供货仓库代码
OL_DELIVERY_D	日期和时间，或 null	发货时间
OL_QUANTITY	数字型，2 个数字	订购数量
OL_AMOUNT	数字型，6 个数字	价格
OL_DIST_INFO	固定文本，大小为 24	

主关键字：（OL_W_ID，OL_D_ID，OL_O_ID，OL_NUMBER）

（OL_W_ID，OL_D_ID，OL_O_ID）外关键字，引用（O_W_ID，O_D_ID，O_ID）

（OL_SUPPLY_W_ID，OL_I_ID）外关键字，引用（S_W_ID，S_I_ID）

Item 表（商品）

域名	域定义	注释
I_ID	200,000 个唯一的 ID	有 100,000 件商品
I_IM_ID	200,000 个唯一的 ID	与商品相关的图象 ID
I_NAME	可变文本，大小为 24	
I_PRICE	数字型，5 个数字	
I_DATA	可变文本，大小为 50	商标信息

主关键字：I_ID

Stock 表（库存）

域名	域定义	注释
S_I_ID	200,000 个唯一的 ID	每个仓库有 100,000 个
S_W_ID	2*W 个唯一的 ID	
S_QUANTITY	数字型，4 个数字	
S_DIST_01	固定文本，大小为 24	
S_DIST_02	固定文本，大小为 24	
S_DIST_03	固定文本，大小为 24	
S_DIST_04	固定文本，大小为 24	
S_DIST_05	固定文本，大小为 24	
S_DIST_06	固定文本，大小为 24	

S_DIST_07	固定文本，大小为 24	
S_DIST_08	固定文本，大小为 24	
S_DIST_09	固定文本，大小为 24	
S_DIST_10	固定文本，大小为 24	
S_YTD	数字型，8 个数字	累计供货数量
S_ORDER_CNT	数字型，4 个数字	累计订单数量
S_REMOVE_CNT	数字型，4 个数字	累计其他仓库供货数量
S_DATA	可变文本，大小为 50	制作信息

主关键字：(S_W_ID，S_I_ID)

S_W_ID 外关键字，引用 W_ID

S_I_ID 外关键字，引用 I_ID

致谢

本文即将结束之际，首先要感谢我的导师沈琦副教授，在整个论文过程中对我进行了悉心的指导，同沈老师的交流对我的论文完成有很大的帮助。另外，在整个论文设计阶段，企业导师陈晟博士也给与我很多的指导，从设计的方法上给与我很大的启发。另外，还要感谢我工作单位的所有同事，在工作中与他们的讨论开阔了我的思路，同时也让我学到了很多知识。

另外，学院的所有老师和同学给与我学习上的帮助，在此也表示忠心的谢意！

最后，我还要感谢我的家人，他们不仅在生活和工作中给与我莫大的关怀和帮助，更给与我精神上的支持和鼓励，我唯有努力学习和工作，以报答他们对我的恩情。

基于TPC-C的数据库系统性能测试集的设计与实现

作者：文栋
学位授予单位：北京工业大学