МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: «Технологія оцінки якості даних в OLAP системах»

Виконав: студент групи УС-211М Рунов Андрій Олексійович

Керівник: доцент Моденов Юрій Борисович

Нормоконтролер Ігор РАЙЧЕВ

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології",122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології "

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Аліна САВЧЕНКО

"___"_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Рунов Андрій Олексійович

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Технологія оцінки якості даних в OLAP системах», затверджена наказом ректора від "29" вересня 2023р. за № 1976/ст.

2. **Термін виконання роботи :** з 02 жовтня 2023р. по 31 грудня 2023р.

3. **Вихідні дані до роботи:** огляд робочих методів тестування даних аналітичних систем.

4. **Зміст пояснювальної записки:** Проаналізовано сучасні технології та методи тестування програмного забезпечення. Проаналізована інформаційна інфраструктура підприємства. Описані методи тестування даних OLAP-системи. Проведено вибіркове тестування даних аналітичної системи по трьом методам.

5. **Перелік обов'язкового ілюстративного матеріалу:** слайди, презентація.

6. **Календарний план-графік:**

№ п/п	Завдання	Термін виконання	Підпис керівника
1	Аналіз і опрацювання літератури	02.10.2023 – 10.10.2023	
2	Провести консультацію з науковим керівником щодо розділів дипломної роботи	10.10.2023 – 15.10.2023	
3	Підготовка та написання розділу 1	15.10.2023 – 18.10.2023	
4	Підготовка та написання розділу 2	18.10.2023 – 24.10.2023	
5	Підготовка та написання розділу 3	24.10.2023 – 06.11.2023	
6	Оформлення пояснювальної записки	06.11.2023 – 19.11.2023	
7	Оформлення графічної частини роботи	19.11.2023 – 27.11.2023	
8	Подати дипломну роботу керівнику	11.12.2023	
9	Підготовка до захисту дипломної роботи	12.12.2023 – 20.12.2023	

7. **Дата видачі завдання:** 02.10.2023р.

Керівник дипломної роботи _____ Юрій МОДЕНОВ
(підпис керівника)

Завдання прийняв до виконання _____ Андрій РУНОВ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему: «Методика тестування даних OLAP-систем». Дипломна робота містить 92 сторінок, 13 рисунків та 10 використаних джерел.

Ключові слова: ТЕСТУВАННЯ, БАЗА ДАНИХ, МОДЕЛЬ ДАНИХ, ЗВІТ, АНАЛІТИЧНА СИСТЕМА, ТРАНЗАКЦІЙНА СИСТЕМА, ERP, OLAP.

Об’єкт дослідження: дані з транзакційної системи

Предмет дослідження: Особливості даних з транзакційної системи

Мета роботи: Описати методи тестування даних аналітичних систем

Методи дослідження: Аналіз області тестування даних аналітичних систем

Результати дипломної роботи планується використовувати: методи тестування великих обсягів даних аналітичної системи застосовуються для фактичного тестування описаних систем

Завдання дослідження:

1. Проаналізувати методи тестування та інфраструктуру підприємства.
2. Зробити опис методів тестування
3. Реалізувати три методи тестування

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ

ПЗ – програмне забезпечення

БД – База даних

ІС – Інформаційна система

ІТ – Інформаційні технології

СУБД – Системи управління базами даних

OLAP – Online Analytical Processing

ВСТУП

Найбільше зростання інтересу до напрямку тестування припало на 1990-ті роки і розпочалося у Сполучених Штатах Америки. Стрімкий розвиток автоматизованих систем розробки програмного забезпечення та мережевих технологій призвів до зростання ринку виробництва програмного забезпечення та перегляду питання забезпечення якості та надійності створюваних програм. Конкуренція між виробниками програмного забезпечення, що стрімко посилювалася, вимагала особливої уваги до якості створюваних продуктів, оскільки у споживача з'явився вибір: чимало компаній пропонували свої продукти і послуги за цілком прийнятними цінами, а тому можна було звернутися до тих, хто розробить програму не тільки швидко й дешево, але також якісно. Ситуація ускладнювалася ще й тим, що комп'ютеризації піддаються практично всі сфери людського життя. І питання якості програмного забезпечення стає все більш актуальним: сьогодні це вже не просто питання зручності роботи з конкретною програмою. Сьогодні програмне забезпечення керує обладнанням у лікарнях, системами управління в аеропортах, ядерними реакторами, космічними кораблями і т.д.

Усвідомлюючи, що забезпечення якості програмного забезпечення, яке вони розробляють, є реальним способом перевершити конкурентів, багато компаній по всьому світу інвестують все більше коштів у забезпечення якості своїх продуктів, створюючи власні команди та відділи тестування або віддаючи тестування своїх продуктів на аутсорсинг зовнішнім організаціям.

Великі компанії, які піклуються про свою репутацію і хочуть бути сертифікованими на найвищому рівні CMMI (Capability Maturity Model Integration), створюють власні системи управління якістю, спрямовані на постійне вдосконалення своїх виробничих процесів і безперервне поліпшення якості програмного забезпечення.

Нині тестування стало обов'язковою частиною процесу розробки програмного забезпечення. Його мета – виявити та усунути якомога більше помилок. Результатом такої діяльності є підвищення якості програмного забезпечення за всіма його

характеристиками.

Існуючі методи тестування програмного забезпечення не дозволяють однозначно і повністю усунути всі дефекти і помилки та встановити коректність програмного продукту. Тому всі існуючі методи тестування працюють в рамках формального процесу перевірки програмного продукту, що розробляється.

Такий формальний процес тестування або верифікації може довести, що з точки зору використовуваного методу дефекти відсутні. Іншими словами, не існує способу визначити або гарантувати відсутність дефектів у програмному забезпеченні, беручи до уваги людський фактор, присутній на всіх етапах життєвого циклу програмного забезпечення.

Існує багато підходів до вирішення проблеми тестування та верифікації програмного забезпечення, але ефективне тестування складних програмних продуктів – це надзвичайно творчий процес, який не зводиться до дотримання або створення суворих і чітких процедур.

Кінцевою метою будь-якого процесу тестування є забезпечення досягнення сукупної концепції якості з урахуванням усіх або найбільш важливих елементів у конкретному випадку.

Тестування програмного забезпечення – це спроба визначити, чи працює програма так, як очікується. Як правило, жодне тестування не може дати стовідсоткової гарантії того, що програма працюватиме в майбутньому.

Метою тестування програмного забезпечення є зниження витрат на розробку шляхом раннього виявлення дефектів.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ОБЛАСТІ

1.1 Тестування

Системи з програмним забезпеченням від бізнес-додатків до споживчих товарів є невід'ємною частиною нашого життя. Багатьом людям доводилося мати справу з програмним забезпеченням, яке працювало не так, як початково очікувалося. Програмне забезпечення, яке не працює належним чином, може спричинити багато проблем, включаючи втрату фінансів, часу або ж ділової репутації, і навіть викликати травми або смерть.

Людина може припуститися помилки, яка спричиняє дефект у програмному коді або документі. Якщо несправний код буде виконано, система не зможе робити те, що вона повинна робити. Дефекти в програмному забезпеченні, системах або документах можуть призвести до збою, але не всі дефекти мають настільки фатальний ефект.

Дефекти виникають через те, що людям властиво помилятися, бракує часу, код складний, інфраструктура складна, технології змінюються і є багато системних взаємодій.

Збої також можуть бути спричинені умовами навколишнього середовища. Наприклад, радіація, пил або електромагнітні поля.

Ретельне тестування систем і документації може зменшити ризик виникнення проблем під час експлуатації та сприяє підвищенню якості програмної системи, якщо дефекти будуть виправлені до того, як система буде запущена в експлуатацію.

Тестування є можливим способом оцінки якості програмного забезпечення з точки зору виявлених дефектів, як для функціональних, так і для нефункціональних вимог та можливостей програмного забезпечення.

Тестування може створити впевненість у якості програмного забезпечення, якщо знайдено мало дефектів або їх взагалі не виявлено. Добре спланований тест, який пройшов успішно, знижує загальний рівень ризику в системі. Якщо під час тестування виявлено дефекти, якість програмних систем підвищується, коли дефекти виправлені.

Розуміння першопричин дефектів, виявлених в інших проектах, дає змогу вдосконалити процеси, що, в свою чергу, має запобігти повторному виникненню цих дефектів і, в кінцевому підсумку, підвищити якість майбутніх систем. Це і є підхід забезпечення якості.

Тестування – це не просто виконання тестів. Тестування включає в себе як до-, так і післятестову діяльність. Ця діяльність включає планування та управління, вибір умов тестування, розробку та виконання тестових сценаріїв, перевірку результатів, оцінку критеріїв завершення, створення звітів про процес тестування та систему, що тестується, а також закриття або завершення пост-тестової діяльності. Тестування також включає перевірку документації.

Як динамічні, так і статичні тести використовуються для досягнення одних і тих же цілей, надаючи інформацію, яка може допомогти поліпшити як систему, що тестується, так і процес розробки тестів.

Завдання тесту:

- виявлення дефектів;
- підвищена впевненість у рівні якості;
- надання інформації для прийняття рішень;
- запобігання дефектам.

Цілі процесу та планування тестування на ранній стадії життєвого циклу розробки програмного забезпечення можуть допомогти запобігти потраплянню дефектів до коду. Перегляд документації, виявлення та вирішення проблем також допомагають запобігти потраплянню дефектів до коду.

Різні точки зору на тестування мають різні цілі. Наприклад, при тестуванні розробки основною метою може бути спричинення якомога більшої кількості збоїв, щоб виявити дефекти програмного забезпечення та виправити їх. У приймальному тестуванні головною метою може бути підтвердження того, що система працює так, як очікувалося, і підвищення впевненості в тому, що вона відповідає вимогам. У деяких випадках основною метою тестування може бути оцінка якості програмного забезпечення та інформування зацікавлених сторін про ризики, пов’язані з випуском системи вчасно. Тестування під час періоду супроводу – це, головним чином, перевірка на наявність нових дефектів, які могли бути внесені під час розробки змін. Під час експлуатаційного тестування основною метою може бути оцінка таких характеристик системи, як надійність або доступність.

1.1.1 Принципи тестування

Існує сім принципів тестування, які слугують загальними настановами для тестування загалом:

1. Тестування демонструє наявність дефектів. Тестування може показати, що дефекти присутні, але не може довести, що їх немає. Тестування зменшує ймовірність дефектів у програмному забезпеченні, але навіть якщо дефектів не виявлено, це не доводить, що програмне забезпечення є правильним.
2. Повне тестування недосяжне. Повне тестування з використанням усіх комбінацій вхідних даних і передумов фізично неможливе, за винятком тривіальних випадків. Замість вичерпного тестування слід використовувати аналіз ризиків та розстановку пріоритетів, щоб точніше сфокусувати зусилля з тестування.

3. Раннє тестування. Для того, щоб виявити дефекти якомога раніше, тестування повинно починатися на якомога більш ранній стадії життєвого циклу програмного забезпечення або системи і зосереджуватися на конкретних цілях.

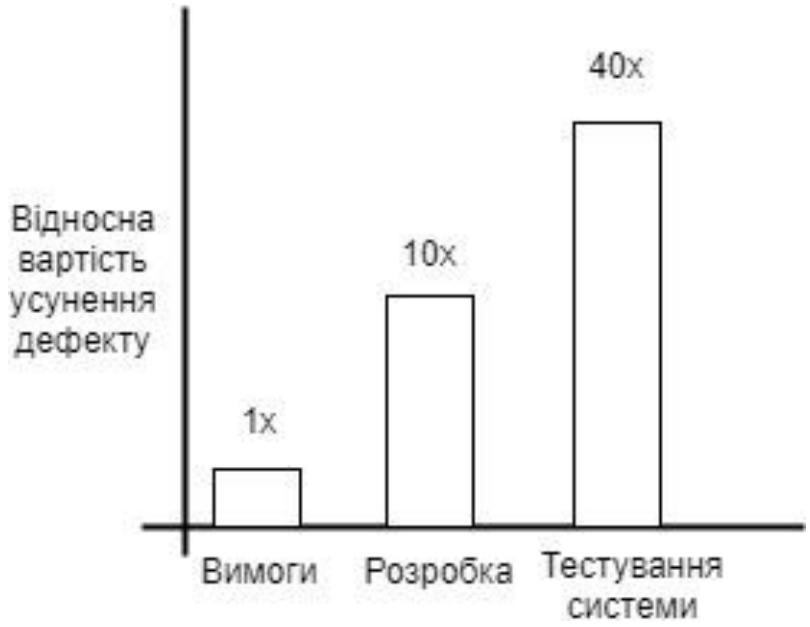


Рис. 1.1. Переваги раннього тестування

4. Накопичення дефектів. Зусилля з тестування повинні бути сконцентровані пропорційно очікуваній, а потім фактичній щільності дефектів в окремих модулях. Як правило, більшість дефектів, що виявляються під час тестування або спричиняють більшість відмов системи, знаходяться в невеликій кількості модулів.
5. «Парадокс пестицидів». Якщо одні й ті самі тести запускати багато разів, зрештою, цей набір тестових скриптів перестане знаходити нові дефекти. Щоб подолати цей «парадокс пестицидів», тестові скрипти повинні регулярно переглядатися і коригуватися, а нові тести повинні бути комплексними, щоб охоплювати всі компоненти програмного забезпечення або системи і знаходити якомога більше дефектів.
6. Тестування залежить від контексту. Тестування проводиться по-різному залежно від контексту. Наприклад, програмне забезпечення, де безпека є критично важливою, тестується інакше, ніж сайт електронної комерції.

7. Безпомилковість. Виявлення та виправлення помилок не допоможе, якщо створена система не влаштовує користувача і не відповідає його очікуванням та потребам.

1.1.2 Рівні тестування

Тестування компонентів

Тестова база:

- вимоги до компонентів;
- детальний дизайн;
- код.

Типові тестові об’єкти:

- компоненти;
- програми;
- перетворення даних та міграції даних;
- модулі баз даних.

Компонентне тестування (також відоме як модульне тестування) передбачає пошук дефектів і перевірку працездатності програмних модулів, програм, об’єктів, класів тощо. Це можна робити ізольовано від решти системи, залежно від контексту програмного забезпечення та життєвого циклу системи. У цьому процесі можуть використовуватися заглушки, драйвери та емулятори.

Тестування компонентів може варіюватися від функціонального тестування до специфічного нефункціонального тестування, такого як поведінка ресурсів (наприклад, виявлення витоків пам’яті) або тестування надійності, а також структурного тестування (наприклад, покриття коду). Сценарії тестування розробляються на основі артефактів процесу розробки, таких як специфікація компонента, дизайн або модель бази даних.

Рис. 1.2. Схематичний огляд тестування компонентів

Зазвичай компонентне тестування виконується з доступом до коду, що тестується, і за підтримки робочого середовища, такого як фреймворк для модульного тестування або утиліти для налагодження. На практиці компонентне тестування зазвичай виконується розробниками, які пишуть код. Дефекти зазвичай виправляються одразу після їх виявлення, без реєстрації в базі даних дефектів.

Один з підходів до тестування компонентів полягає у написанні автоматизованих тестових скриптів перед кодуванням. Це називається тест-керованою розробкою. Цей підхід складається з декількох ітерацій і базується на циклах створення тестових скриптів, написання та інтеграції невеликого фрагмента коду і запуску тестів компонентів, виправлення будь-яких проблем і запуску тестів до досягнення позитивного результату.

Інтеграційні тести

Базис тестування:

- проектування системи;
- архітектура;
- бізнес-процеси;
- варіанти використання.

Типові тестові об’єкти:

- база даних підсистем;
- інфраструктура;

− інтерфейси.

Конфігурація системи:

− дані конфігурації.

Інтеграційні тести перевіряють інтерфейси між компонентами, взаємодію різних частин системи, таких як операційні системи, файлові системи, апаратне забезпечення та інтерфейси між системами.

Інтеграційні тести можуть складатися з одного або декількох рівнів і можуть проводитися на тестових об’єктах різного розміру:

1. Тестування інтеграції компонентів перевіряє взаємодію між програмними компонентами і проводиться після тестування компонентів.

2. Тестування системної інтеграції перевіряє взаємодію між системами або обладнанням і може проводитися після тестування системи. У цьому випадку розробники можуть контролювати лише одну сторону інтерфейсу. Однак це може розглядатися як ризик. Бізнес-процеси можуть включати послідовність систем; відмінності між платформами можуть бути важливими.

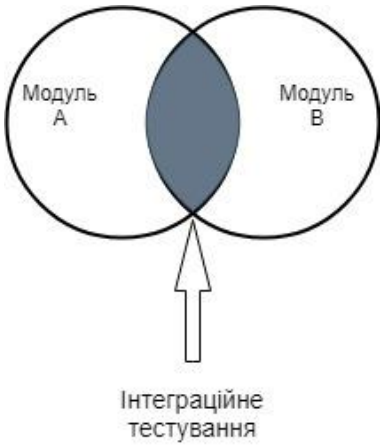


Рис. 1.3. Загальна схема інтеграційних тестів

Чим більший обсяг інтеграції, тим складніше стає ізолювати несправності в окремому компоненті або системі, що може призвести до підвищеного ризику і вимагати додаткового часу для вирішення проблем.

Стратегії тестування системної інтеграції можуть базуватися на архітектурі системи (наприклад, зверху вниз або знизу вгору), функціональних завданнях, послідовності обробки транзакцій або інших аспектах системи та її компонентів. Щоб спростити процес ізоляції збоїв і виявити несправності якомога раніше, інтеграцію слід виконувати поступово, а не за сценарієм «великого вибуху».

Тестування конкретних нефункціональних характеристик (наприклад, продуктивності) може бути включено в інтеграційне тестування разом з функціональним тестуванням.

На кожному етапі інтеграції тестувальники зосереджуються на самій інтеграції. Наприклад, якщо вони інтегрують модуль А з модулем В, вони тестують взаємодію модулів, а не функціональність кожного з них, оскільки це має бути перевірено під час тестування компонентів. Для тестування можна використовувати як функціональний, так і структурний підходи.

В ідеалі, тестувальники повинні розуміти архітектуру та її вплив на планування інтеграції. Якщо інтеграційне тестування планується до розробки компонентів або системи, ці компоненти можна розробляти в порядку, що забезпечує найбільш ефективне тестування.

Системне тестування

Базис тестування:

Вимоги до програмного забезпечення та система специфікацій

- варіанти використання;
- функціональна специфікація;
- звіти з аналізу ризиків.

Типові тестові об’єкти:

- інструкція з експлуатації системи;
- конфігурація системи;

Дані конфігурації

Системне тестування фокусується на поведінці об’єкта тестування як цілісної системи або продукту. Обсяг тестування повинен бути чітко визначений в генеральному плані тестування або плані тестування для певного рівня тестування.

Під час тестування системи тестове середовище має бути максимально наближеним до передбачуваного робочого середовища системи, щоб мінімізувати ризик пропущених збоїв, пов’язаних з робочим середовищем системи.

Системне тестування може включати тести на основі специфікацій ризиків або вимог, бізнес-процесів, варіантів використання системи або інших високорівневих текстових описів чи моделей поведінки системи, взаємодії з операційною системою та системними ресурсами.

Тестування системи повинно перевіряти функціональні та нефункціональні вимоги до системи, а також якість даних, що обробляються. Тестувальники також повинні бути здатні виконувати свої обов’язки у випадку неповних або недокументованих вимог. Тестування системи на відповідність функціональним вимогам починається з тестування на основі специфікацій (тестування «чорного ящика») для різних аспектів системи. Наприклад, можна створити таблицю рішень на основі бізнес-правил системи. Тестування на основі структури (тестування білої скриньки) можна використовувати для оцінки точності тестування конкретного структурного елемента, наприклад, структури меню або навігації веб-сайту.

Тестування системи найчастіше проводиться незалежною командою тестувальників.

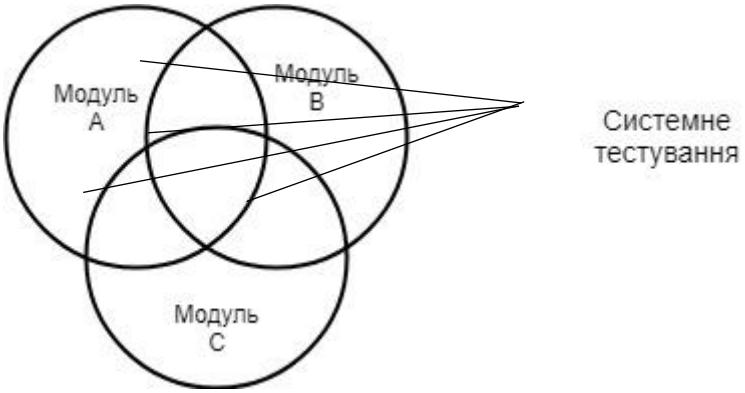


Рис. 1.4. Загальна схема системного тестування

Приймальні випробування

Базис тестування:

- вимоги до користувача;
- системні вимоги;
- варіанти використання;
- бізнес-процеси;
- звіти з аналізу ризиків.

Типові тестові об’єкти:

- бізнес-процеси в повністю інтегрованій системі;
- процеси експлуатації та обслуговування;
- процедури використання;
- звіти.

Конфігураційні дані.

Приймальні випробування системи найчастіше проводяться замовниками або користувачами системи, а також іншими зацікавленими сторонами.

Основною метою приймального тестування є перевірка функціональності системи, її частин або окремих нефункціональних особливостей системи. Пошук несправностей не є основною метою приймального тестування.

Приймальне тестування оцінює готовність системи до розгортання та використання, хоча воно не обов’язково є кінцевим рівнем тестування. Під час приймального тестування, наприклад, можуть бути проведені масштабні тести системної інтеграції.

Приймальні випробування можуть проводитися на різних етапах життєвого циклу продукту:

- у випадку з коробковим продуктом приймальні випробування можуть проводитися під час монтажу або інтеграції;

- приймальне тестування юзабіліті компонента може бути виконано під час тестування компонента;
- приймальне тестування нової функціональності може бути проведене до початку тестування системи.

1.1.3 Типи тестування

Групи тестових заходів можуть бути спрямовані на перевірку працездатності системи, виходячи з різних цілей і причин тестування.

Типи тестів визначаються цілями, які можуть бути наступними:

- функція, яку виконує додаток;
- нефункціональний атрибут якості, такий як надійність або зручність використання;
- структура або архітектура програми або системи.

Функції, які виконує система, підсистема або компонент, можуть бути описані в артефактах розробки, таких як специфікація вимог, варіанти використання системи або функціональна специфікація, або ж вони можуть бути не задокументовані. Ці функції описують, що саме робить система.

Функціональні тести розробляються на основі функцій і можливостей системи та їх взаємодії з конкретними системами і можуть проводитися на всіх рівнях тестування.

Методи розробки тестів на основі специфікацій використовуються для вилучення інформації про умови тестування та тестові сценарії з функціональності програми або системи. Функціональні тести аналізують зовнішню поведінку програмного забезпечення.

Нефункціональне тестування включає навантажувальне тестування, тестування продуктивності, стрес-тестування, юзабіліті-тестування, тестування відновлення, тестування надійності та тестування переносимості. Це тестування того, як працює система.

Нефункціональне тестування можна проводити на всіх рівнях тестування. Термін «нефункціональне тестування» описує тести, необхідні для оцінки характеристик систем і додатків, які можна виміряти кількісно, наприклад, час відгуку при тестуванні продуктивності. Нефункціональне тестування аналізує зовнішню поведінку програмного забезпечення і в більшості випадків використовує створення тестів «чорного ящика».

Структурне тестування (тестування «білого ящика») можна проводити на всіх рівнях тестування. Методи структурного тестування найкраще використовувати після методів розробки тестів на основі специфікацій, щоб виміряти точність тестування шляхом вимірювання покриття структури програми.

Покриття – частина структури програми, яка була покрита тестом, виражена у відсотках. Якщо покриття менше 100%, слід розробити додаткові тести для покриття відсутніх частин програми.

Регресійне тестування – це повторне тестування вже протестованих програм після внесення до них змін з метою виявлення дефектів, що з'явилися або були пропущені в результаті цих дій. Ці дефекти можуть бути присутніми в компоненті, що тестується, а також у пов'язаних або не пов'язаних з ним компонентах. Цей тип тестування виконується після внесення змін до програмного забезпечення або його оточення. Глибина тестування оцінюється на основі ризику не помітити дефекти в програмному забезпеченні, яке працювало раніше.

1.1.4 Методи проектування тестів

Методи на основі специфікацій або чорної скриньки

Еквівалентне розбиття. Вхідні дані в програмному забезпеченні або системі поділяються на групи, які повинні поводитися подібним чином, тобто оброблятися аналогічним чином. Еквівалентні області (або класи) можуть бути визначені як для валідних, так і для невалідних даних, тобто значень, які слід відкинути. Діапазони також можна визначити для вихідних даних, внутрішніх значень, значень, що

залежать від часу (наприклад, до або після події) і параметрів інтерфейсу (наприклад, під час інтеграційних тестів). Тести можуть бути розроблені таким чином, щоб включати всі допустимі і всі недопустимі класи.

Еквівалентне розбиття можна застосовувати на всіх рівнях тестування. Еквівалентне розбиття можна застосовувати як до вхідних, так і до вихідних даних. Його можна використовувати для ручного введення, для передачі даних через інтерфейси до системи або для перевірки параметрів інтерфейсу під час інтеграційного тестування.

Аналіз граничних значень. Поведінка на кордонах еквівалентних областей має найбільший шанс бути неправильною, тому такі кордони є потенційним джерелом дефектів. Мінімальне і максимальне значення сегмента є граничними значеннями. Граничне значення для правильного сегмента є правильним граничним значенням, для неправильного сегмента – неправильним граничним значенням. Тести можуть бути розроблені таким чином, щоб включати як правильні, так і неправильні граничні значення. При розробці тестових сценаріїв тести підбираються для кожного граничного значення.

Аналіз граничних значень можна використовувати на всіх рівнях тестування. Він відносно простий у використанні та ефективний у пошуку дефектів. Детальні специфікації надзвичайно корисні для визначення меж, що становлять інтерес.

Цей метод часто розглядається як доповнення до методу розбиття за еквівалентністю. Його можна використовувати для класів еквівалентності екранного введення, а також для класів еквівалентності часових діапазонів (наприклад, часових обмежень або вимог до продуктивності транзакцій) або розмірів таблиць.

Тестування таблиць рішень. Таблиці рішень є хорошим методом для збору системних вимог, що містять логічні умови та документування внутрішнього дизайну системи. Їх можна використовувати для запису складних бізнес-правил, які повинна реалізовувати система. Специфікації аналізуються і визначаються системні умови та

дії. Вхідні умови та дії часто формулюються таким чином, що вони можуть приймати логічні значення «істина» або «хибність».

Таблиця рішень містить умови запуску, як правило, комбінації істинних і хибних значень для всіх вхідних умов, а також відповідні дії для кожної комбінації умов. Кожен стовпець таблиці відповідає бізнес-правилу, яке визначає унікальну комбінацію умов і результат дій, пов'язаних з цим правилом. Стандартний обсяг тестування таблиць прийняття рішень зазвичай включає принаймні один тест для кожного стовпця, який, як правило, охоплює всі комбінації умов тригера.

Сильна сторона тестування за допомогою таблиці рішень полягає в тому, що воно створює комбінації умов, які не можуть бути протестовані в інший спосіб. Цей метод можна використовувати у всіх ситуаціях, коли робота програмного забезпечення залежить від декількох логічних альтернатив.

Тестування таблиці переходів. Система може відображати різні реакції в залежності від поточних умов або історії попередніх станів. Цей метод дозволяє тестувальнику розглядати систему з точки зору її станів, переходів між станами, входів або подій, які активують зміни станів (переходи) і дій, до яких ці переходи призводять. Стани системи або об'єкта тестування є роз'єднаними, умовними і кінцевими.

Таблиця станів показує зв'язок між станами та входами і може підказати можливі неправильні переходи.

Тести можуть бути створені для покриття типової послідовності станів, для покриття кожного стану, для виконання кожного переходу, для виконання певних послідовностей переходів або для перевірки неприпустимих переходів.

Тестування за допомогою таблиць переходів найчастіше використовується в індустрії програмного забезпечення та автоматизації. Однак ця техніка також підходить для моделювання бізнес-об'єктів, які мають певні стани, або для тестування діалогових послідовностей (наприклад, веб-додатків або бізнес-сценаріїв).

Тестування за допомогою кейсів використання. Тестування може ґрунтуватися на варіантах використання. Варіант використання описує взаємодію між акторами (включаючи користувачів і систему), яка призводить до корисних результатів для замовника або користувача системи. Варіанти використання можуть бути описані на рівні абстракції (бізнес-варіант використання, рівень бізнес-процесу, не пов'язаний з технологією) або на системному рівні (системний варіант використання на рівні функціональності системи). Кожен варіант використання має передумови, необхідні для успішного виконання сценарію. Кожен варіант використання закінчується кінцевою умовою, тобто спостережуваними результатами і кінцевим станом системи після виконання сценарію. Варіант використання зазвичай має основний (найбільш ймовірний) сценарій і альтернативні сценарії.

Варіанти використання описують «потоки процесів» у системі на основі типового передбачуваного використання. Отже, тестові сценарії, засновані на варіантах використання, найбільш корисні для виявлення дефектів у потоках процесів під час фактичного використання системи. Варіанти використання дуже корисні для розробки приймальних тестів разом із замовником або користувачами. Вони також можуть виявити дефекти інтеграції, спричинені взаємодією різних компонентів, які не виявляються при тестуванні окремих компонентів.

Проектування тестових сценаріїв на основі варіантів використання можна комбінувати з іншими методами, заснованими на специфікаціях.

Тестування на основі структури, або за методом «білого ящика»

Тестування операторів та покриття. При компонентному тестуванні покриття операторів – це їх частка, що перевірена під час виконання набору тестових сценаріїв. При тестуванні інструкцій тестові сценарії створюються для виконання конкретних інструкцій і зазвичай збільшують покриття операторів.

Покриття інструкцій визначається як відношення кількості виконуваних операторів, охоплених тестовими сценаріями (розробленими або виконаними), до загальної кількості операторів у коді, що тестується.

Тестування альтернатив і покриття. Покриття альтернатив, пов’язане з тестуванням розгалужень, – це відсоток альтернативних результатів (наприклад, True і False для операторів If), перевірених набором тестових сценаріїв. У методі тестування альтернатив тестові сценарії створюються для виконання певних альтернативних результатів. Гілки виходять з альтернативних точок програмного коду і показують передачу управління в різні частини коду.

Альтернативне покриття визначається відношенням кількості всіх альтернативних результатів, що покриваються розробленими або виконаними тестовими сценаріями, до кількості всіх можливих альтернативних результатів у коді, що тестується.

Тестування альтернатив – це тип тестування потоку управління, який описує проходження певного потоку через альтернативні точки. Покриття альтернатив є більш суворим, ніж покриття операторів: 100 відсотків покриття альтернатив гарантує 100 відсотків покриття операторів, проте не навпаки.

Інші структурні методи. Існують вищі рівні структурного покриття після альтернативного покриття. Наприклад, покриття станів і покриття декількох станів.

Принцип покриття можна застосовувати і на інших рівнях тестування. Наприклад, на рівні інтеграції відсоток модулів, компонентів або класів, протестованих набором тестових сценаріїв, можна виразити як покриття модулів, компонентів або класів.

Інструментальна підтримка надзвичайно корисна під час структурних випробувань.

Методи, засновані на досвіді

Коли тести розробляються на основі навичок, інтуїції та досвіду роботи тестувальника з подібними додатками або технологіями, це називається тестуванням на основі досвіду. Цей тип тестування корисний як доповнення до більш систематичних спеціальних методів розробки тестів, які не завжди очевидні при використанні більш формальних методів, особливо коли вони застосовуються після

таких методів. Однак корисність цього методу може значно відрізнятися в залежності від досвіду тестувальника.

Найпоширеніший метод, заснований на досвіді, – це припущення про наявність дефекту. Тестувальники часто очікують дефекти на основі свого досвіду. Організований підхід до пошуку дефектів полягає у створенні списку можливих дефектів і розробці тестів, які атакують ці дефекти. Цей підхід називається атака на дефекти або дефектоскопія. Списки дефектів і збоїв можуть бути створені на основі досвіду, доступної інформації про дефекти і збої та загального розуміння того, чому програмне забезпечення може давати збої.

Дослідницьке тестування – паралельна розробка, виконання, протоколювання та вивчення тестів на основі тестової концепції, яка включає тестові цілі та проводиться у визначені часові рамки. Цей підхід є найбільш корисним, коли специфікації є неповними або застарілими, а часові обмеження є жорсткими, або коли потрібно покращити чи доповнити більш формальні тести. Він може слугувати валідацією процесу тестування, щоб забезпечити виявлення ключових дефектів.

Вибір методів тестування

Вибір методу залежить від ряду факторів, включаючи тип системи, регуляторні стандарти, вимоги замовника або контракту, рівні ризику, типи ризику, цілі тестування, наявну документацію, знання тестувальника, час і бюджет, життєвий цикл програмного забезпечення, моделі використання і попередній досвід роботи з виявленими типами дефектів.

Деякі методи більше підходять для конкретних ситуацій, тоді як інші можна застосовувати на всіх рівнях тестування.

При розробці тестових сценаріїв тестувальники зазвичай використовують комбінацію методів тестування, включаючи методи, засновані на процесах, правилах і даних, щоб забезпечити адекватне покриття об’єкта тестування.

1.1.5 Філософія тестування

Тестування програмного забезпечення включає низку дій, які дуже схожі на послідовність процесів розробки програмного забезпечення. Сюди входять визначення тестової задачі, проектування, написання тестів, виконання тестів і, нарешті, виконання тестів та перевірка результатів тестування. Дизайн тестів відіграє ключову роль. Існує багато підходів до розробки філософії або стратегії проектування тестів. Щоб допомогти зорієнтуватися в стратегіях розробки тестів, корисно розглянути два крайні підходи, які знаходяться на кінцях спектра. Слід також зазначити, що багато людей, які працюють у цій сфері, часто впадають в одну або іншу крайність.

Підхід на лівій стороні спектру розробляє свої тести, вивчаючи зовнішні або інтерфейсні специфікації програми або модуля, що тестується. Він розглядає додаток як чорний ящик. Його позиція полягає в наступному: «Мене не цікавить, як виглядає програма і чи я виконав усі команди або пройшов усі шляхи. Я буду задоволений, якщо програма поводитиметься відповідно до специфікації». Його ідеал – перевірити всі можливі комбінації та значення на вході.

Підхід на іншому кінці спектру розробляє свої тести, вивчаючи логіку програми. Він починається з підготовки достатньої кількості тестів, щоб гарантувати, що кожна команда буде виконана хоча б один раз. Якщо тестувальник трохи досвідченіший, він розробляє тести так, щоб кожна команда умовного переходу виконувалася в кожному напрямку принаймні один раз. Його ідеал – перевірити кожен шлях, кожну гілку алгоритму. При цьому його зовсім (або майже зовсім) не цікавить специфікація.

Жодна з цих крайнощів не є доброю стратегією. Однак перша, а саме та, що розглядає додаток як чорний ящик, є кращою. На жаль, вона має той недолік, що може бути абсолютно непрацездатною. Розглянемо спробу протестувати тривіальну програму, яка приймає на вхід три числа і обчислює їх середнє арифметичне. Неможливо протестувати цю програму для всіх вхідних значень. Навіть на машині з відносно низькою точністю обчислень кількість тестів буде обчислюватися

мільярдами. Навіть якби у нас було достатньо обчислювальних потужностей, щоб виконати всі тести за розумний проміжок часу, ми б витратили на кілька порядків більше часу на їх підготовку, а потім на тестування. Ще гірша ситуація з такими програмами, як системи реального часу, операційні системи та програми управління даними, які зберігають «пам'ять» про попередні вхідні дані. Тут доведеться тестувати програму не лише для кожного вхідного значення, але й для кожної послідовності, кожної комбінації вхідних даних. Тому вичерпне тестування всіх вхідних даних будь-якої розумної програми неможливе.

Ці міркування підводять до другого фундаментального принципу тестування: тестування – це значною мірою економічна проблема. Оскільки вичерпне тестування неможливе, слід задовольнятися чимось меншим. Кожен тест повинен приносити максимальну віддачу порівняно з витратами. Результат вимірюється ймовірністю того, що тест виявить раніше невиявлену помилку. Вартість вимірюється часом і витратами на підготовку, виконання і перевірку результатів тесту. Припускаючи, що витрати обмежені бюджетом і графіком, можна стверджувати, що мистецтво тестування – це, по суті, мистецтво вибору тестів з максимально можливою віддачою. Крім того, кожен тест повинен бути репрезентативним для певного класу вхідних значень, щоб його правильне виконання давало впевненість у тому, що для певного класу вхідних даних програма працюватиме правильно. Зазвичай це вимагає певних знань алгоритму та структури програми, тому ми наближаємося до правого кінця спектру.

1.2 Інформаційна інфраструктура підприємства

1.2.1 Аналітична піраміда

Інформаційну інфраструктуру компанії можна представити у вигляді декількох ієрархічних рівнів, кожен з яких характеризується ступенем агрегації інформації та її роллю в процесі управління. Прикладом схематичного представлення інформаційної інфраструктури є аналітичний стек, запропонований компанією Gartner.

Аналітична піраміда - це ієрархічна структура, в якій різні класи інформаційних систем розташовані на різних рівнях. До цих рівнів відносяться:

- рівень торгових систем;
- рівні сховища даних;
- рівень представлення даних;
- рівень OLAP-систем;
- рівень аналітичних додатків.

В основі аналітичної піраміди лежать транзакційні системи, призначені для управління щоденними операціями і, таким чином, вони є основним джерелом інформації для аналізу. Коли піраміда рухається від основи до вершини, детальні операційні дані трансформуються в агреговану інформацію для підтримки прийняття управлінських рішень.

Рис. 1.5. Аналітична піраміда

Не завжди можна віднести програмне забезпечення до одного класу, оскільки багато систем дозволяють вирішувати аналітичні завдання в декількох категоріях.

Наприклад, OLAP-системи від різних виробників можуть виступати в ролі аналітичних додатків або використовуватися для побудови багатовимірних сховищ даних і сайтів.

1.2.2 Рівень торгових систем

До транзакційних систем належать системи планування ресурсів підприємства (ERP), автоматизовані банківські системи (АБС), розрахункові системи, бухгалтерські програми та деякі інші. Незважаючи на об'єктивні відмінності, всі ці системи мають спільну рису: вони призначені для обробки одиничних операцій (транзакцій). Тому для опису таких систем часто використовують термін OLTP (On-Line Transaction Processing).

Деякі транзакційні системи є складними і складаються з окремих модулів. Наприклад, модульна структура характерна для ERP-систем, основним завданням яких є об'єднання різних служб підприємства в єдиний управлінський потік. Крім того, такі системи завжди мають набір фінансово-бухгалтерських функцій. Тому транзакційні системи є джерелом базової інформації, яка використовується для подальшої аналітичної обробки. Дані з транзакційних джерел повинні бути зібрані, структуровані та представлені у формі, зручній для прийняття рішень. Самі транзакційні системи також містять деякі аналітичні можливості, але, строго кажучи, не підпадають під категорію аналітичних систем. Водночас вони є постачальниками інформації для систем бізнес-аналітики та аналітичних додатків. Дані можуть передаватися з транзакційних систем до аналітичних додатків або послідовно, через усі визначені рівні аналітичної піраміди, або коротшим шляхом, оминаючи один чи кілька рівнів (на схемі це показано стрілкою "bypass"). Спосіб передачі даних залежить як від технічних можливостей програмного забезпечення, так і від того, як ці дані будуть використовуватися.

Здавалося б, торгові системи, які мають всю основну інформацію, можна використовувати як самодостатні аналітичні інструменти. Однак це справедливо лише в тому випадку, якщо ми говоримо про аналіз на рівні окремих угод. Якщо ж

аналітичні завдання виходять за межі управління транзакціями (на рівні тактичного і стратегічного управління), то такі завдання повинні ґрунтуватися на агрегованій інформації, можливо, отриманій з первинних даних з різних транзакційних систем. Крім того, важливою вимогою аналітики є багатоваріантність (можливість генерувати та оцінювати різні сценарії, в тому числі гіпотетичні), що також не забезпечується транзакційними системами. Тому для вирішення багатьох аналітичних завдань (в тому числі стратегічного та управлінського аналізу) рекомендується використовувати системи, розташовані на інших, більш високих рівнях аналітичної піраміди.

1.2.3 Системи бізнес-інтелекту

Поняття систем бізнес-інтелекту (Business Intelligence, BI) є досить широким і поєднує в собі різні способи аналізу та обробки даних в масштабах підприємства. BI-системи включають такі компоненти, як сховища і сайти даних, інструменти аналітичної обробки даних в режимі онлайн (OLAP-системи), інструменти виявлення знань, а також інструменти створення запитів і звітів. Багато аналітичних систем базуються на сховищах даних, які збирають, організовують і зберігають великі обсяги інформації з різних джерел. Тому має сенс почати зі сховища даних як «середньої ланки» аналітичної піраміди.

Сховища даних (СД) – це наступний рівень аналітичної піраміди після транзакційних систем. Один з найавторитетніших експертів у цій галузі Білл Інмон визначає сховища даних як предметно-орієнтовані, інтегровані, стабільні, чутливі до часу набори даних, організовані для підтримки управління, покликані діяти як «єдине і уніфіковане джерело істини», що забезпечує управлінців і аналітиків достовірною інформацією, необхідною для оперативного аналізу і прийняття рішень. Цінність сховищ даних полягає в тому, що вони є великими базами даних масштабу підприємства, які містять певну інформацію і забезпечують швидке представлення цієї інформації у формі, зручній для користувача або для подальшої обробки іншими аналітичними системами. Часто сховища даних структуровані з урахуванням

специфіки галузі, в якій працює організація. Однак дані, що містяться в сховищах даних, зазвичай недостатньо доступні для обробки в режимі реального часу, особливо коли вони великі. Ця проблема вирішується на наступному рівні ієрархії – на рівні вітрин даних та OLAP-систем.

Подібно до сховищ, вітрини даних є структурованими масивами інформації, але відрізняються тим, що є ще більш предметно-орієнтованими. Як правило, вітрина даних містить інформацію, пов'язану з певною предметною областю діяльності організації. Тому інформація у вітринах даних зберігається в спеціальній формі, найбільш придатній для вирішення конкретних аналітичних завдань або обробки запитів певної групи аналітиків.

Існує два погляди на вітрини даних. В одному випадку, вітрина даних – це частина сховища даних, яка оптимізована для запитів до даних у певній предметній області, в тому числі для надсилання цих даних для подальшої обробки в інші аналітичні системи. Іншими словами, візитна картка – це OLAP-куб або частина OLAP-куба, оптимізована для запитів користувачів до інформації в конкретній предметній області. Отже, з точки зору організації зберігання даних вітрини можуть бути як реляційними, так і багатовимірними, але в кожному випадку вони мають спільну властивість, таку як тематична спрямованість.

Наступний рівень аналітичної піраміди займають системи OLAP (OnLine Analytical Processing) – системи аналітичної обробки даних у режимі реального часу. OLAP-системи можуть надавати рішення для широкого спектру аналітичних задач, таких як KPI-аналіз, маркетинговий та фінансово-економічний аналіз, сценарний аналіз, моделювання, прогнозування тощо. Такі системи можуть працювати з усіма необхідними даними, незалежно від специфіки діяльності компанії.

Специфіка OLAP-систем полягає в багатовимірному зберіганні даних (на відміну від реляційних таблиць), а також у попередньому обчисленні агрегованих значень. Це дає користувачеві можливість будувати оперативні, нерегламентовані запити до даних з використанням декількох аналітичних областей. Крім того, OLAP-системи характеризуються змістовною (а не технічною) структурою інформації, що

дозволяє користувачеві оперувати знайомими економічними категоріями та поняттями.

Ще одним елементом BI-платформи, який часто виділяють в окрему категорію, є виявлення знань (Data Mining). Відповідне програмне забезпечення дозволяє виявляти закономірності в даних і виводити з них якісно нову інформацію. Така інформація може не міститися в джерелі даних у явному вигляді, тому в цьому випадку генеруються знання, керовані даними. Григорій П'ятецький-Шапіро, один з провідних експертів у цій галузі, визначає роботу таких систем як «процес виявлення у вихідних даних невідомих раніше нетривіальних, практично корисних та інтерпретованих знань, необхідних для прийняття рішень у різних сферах людської діяльності». Системи виявлення знань використовують такі методи аналізу даних, як фільтрація, дерева рішень, правила асоціацій, генетичні алгоритми, нейронні мережі та статистичний аналіз.

Нарешті, BI-системи включають в себе інструменти запитів і звітів. Такі системи дозволяють будувати запити до інформаційно-аналітичних систем у визначеному користувачем обсязі, з можливістю інтеграції даних з різних джерел, а також переглядати інформацію з можливістю деталізації та агрегування, будувати звіти та виводити їх на друк. Такі системи можуть використовуватися користувачами з «просунутими» технічними навичками. Професійні знання інформаційних технологій не є обов'язковими, але такі інструменти не завжди зручні для фахівців іншого профілю. Як правило, модулі, що містять функції запитів і звітів, входять до складу багатьох OLAP-систем, хоча існують і окремі програмні продукти цього класу.

Комерційні продукти включають Microsoft SQL Server Analysis Services, Hyperion Essbase, Cognos PowerPlay, BusinessObjects, MicroStrategy, SAP BW, Cartesis Magnitude, Oracle Express, OLAP Option, IBM Cognos TM1. Існує кілька рішень з відкритим вихідним кодом, зокрема Mondrian та Palo.

1.2.4 Аналітичні додатки

На вершині піраміди аналітики знаходяться аналітичні додатки. Як випливає з назви, вони призначені для виконання аналізу і в цьому сенсі принципово відрізняються від транзакційних систем, які зосереджені в першу чергу на обробці окремих транзакцій. Щоб ІТ-систему можна було вважати аналітичним додатком, вона повинна відповідати наступним критеріям:

− повинна забезпечувати структуризацію та автоматизацію процесів, які підвищують якість управлінської інформації, що, в свою чергу, призводить до покращення процесу прийняття рішень. Це досягається шляхом застосування політик, процедур і технологій, заснованих на відповідній методології і спрямованих на вирішення конкретних бізнес-завдань;

− повинна підтримувати аналітичні функції, тобто операції з аналізу даних, отриманих з різних джерел – внутрішніх або зовнішніх, фінансових та управлінських;

− це має бути самостійне програмне забезпечення, яке функціонує незалежно від транзакційних систем, але при цьому здатне взаємодіяти з ними в обох напрямках – як з точки зору отримання початкових транзакційних даних, так і з точки зору повернення результатів їх обробки.

Аналітичні програми часто мають справу з нестандартними, непередбачуваними або рідкісними ситуаціями. Такі ситуації можуть виникати, наприклад, під час впровадження нового продукту, моделювання нової корпоративної структури або створення нового відділу, а також при оцінці впливу злиттів і поглинань, перегляді бюджетів тощо.

Аналітичні програми часто базуються на багатовимірних базах даних (що також відрізняє їх від транзакційних систем, які використовують реляційні бази даних). Як наслідок, аналітичні програми можуть ефективно використовувати як всі необхідні дані, так і бізнес-правила, що описують їх взаємозв’язки з точки зору конкретних бізнес-завдань. Безумовно, переваги аналітичних систем повинні виражатися в прийнятті управлінських рішень, які позитивно впливають на бізнес. Це означає, що аналітичні системи повинні робити більше, ніж просто надавати

інформацію користувачам. Вони повинні слугувати «путівником» у процесі прийняття рішень. Ефект від використання аналітичних систем визначається низкою факторів, серед яких:

− скорочення відстані між аналітиком та особою, яка приймає рішення. У традиційному підході підтримка прийняття рішень – це процедура збору інформації (за допомогою технічних спеціалістів) і подальша передача її менеджеру. У цьому випадку користувач аналітичної системи не приймає рішення, а лише готує інформацію для інших менеджерів. Однак важко гарантувати, що надана інформація буде достатньо релевантною і що на її основі буде прийнято обґрунтоване рішення. Тому необхідно, щоб кінцевим користувачем аналітичної системи був не технічний спеціаліст чи клерк, а менеджер, який прийматиме рішення на основі аналітичної інформації;

− колегіальність у прийнятті рішень. Для того, щоб управлінське рішення було обґрунтованим, суб'єктивної точки зору одного менеджера часто недостатньо. В аналітичному середовищі прийняття рішень ґрунтується на консолідації думок, а самі рішення є результатом спільної роботи кількох менеджерів;

− підтримка рішень та оцінка їх ефективності. Спочатку BI-системи не були орієнтовані на підтримку прийняття рішень, але з часом розробники почали приділяти цьому аспекту належну увагу. В результаті аналітичні системи почали дозволяти оцінювати переваги рішення та його ефективність;

− спираючись на досвід лідерів. У кожній організації є відділи та команди, які можна вважати зразками для наслідування. Поширення та використання таких кращих практик забезпечує управління знаннями та збереження компетенцій, накопичених в організації. Здатність підтримувати процес управління накопиченими знаннями є однією з найважливіших функцій аналітичного програмного забезпечення;

− протидія нераціональним рішенням. Оптимізація прийняття управлінських рішень вимагає адекватної реакції на можливі нераціональні дії деяких менеджерів. Розробники аналітичних систем також беруть це до уваги.

Ці властивості аналітичних систем дають змогу значно покращити ефективність управлінської діяльності та забезпечити більш швидке повернення інвестицій в аналітичне ПЗ.

1.2.5 Характеристики OLAP-систем

Формальне визначення технології OLAP вперше було дано в статті Е.Ф. Кодда, яка була опублікована в 1993 році. Вона мала великий резонанс і привернула увагу до можливостей багатовимірного аналізу. У статті було описано дванадцять правил OLAP, до яких пізніше було додано ще кілька. Всі ці правила були розділені на чотири групи і названі «функціями».

Характеристики OLAP-систем включають в себе:

− ключові особливості: багатовимірна модель даних, інтуїтивно зрозумілі механізми використання даних, їх доступність, пакетне вилучення даних, клієнт-серверна архітектура, прозорість, багатокористувацька підтримка;

− особливості: обробка ненормованих даних, зберігання результатів окремо від вихідних даних, виділення пропущених даних, обробка пропущених значень;

− характеристики генерації звітів: гнучка генерація звітів, стабільна робота під час генерації звітів, автоматичне налаштування фізичного рівня;

− управління розмірами: загальна функціональність, необмежена кількість;

− вимірювання та рівні агрегації, а також необмежені операції між даними з різних вимірів.

Більшість сучасних OLAP-систем не можна однозначно класифікувати ані як програмні засоби, ані як готові додатки. З одного боку, їх використання не вимагає тривалого вивчення теорії та практики побудови аналітичних додатків. З іншого боку, вони не є готовими програмними продуктами для вирішення аналітичних завдань, оскільки вимагають певної кастомізації джерел даних, алгоритмів аналізу та форм представлення кінцевої інформації. Ця подвійність призводить до різноманітності варіантів впровадження, які можуть бути здійснені як системним інтегратором, так і кваліфікованими фахівцями компанії-користувача.

Універсальним критерієм, який визначає OLAP як аналітичний інструмент, є тест FASMI (Fast Analysis of Shared Multidimensional Information). Розглянемо докладніше кожен елемент зазначеної абревіатури.

Швидкість. Ця властивість означає, що OLAP-система повинна надавати відповідь на запит користувача в середньому за п’ять секунд, при цьому більшість запитів обробляється протягом однієї секунди, а найскладніші – протягом двадцяти секунд.

Аналіз. OLAP-система повинна бути здатна виконувати будь-який логічний і статистичний аналіз, типовий для бізнес-додатків, і забезпечувати зберігання результатів у формі, доступній для кінцевого користувача. Аналітичні інструменти можуть включати процедури аналізу часових рядів, розподілу витрат, конвертації валют, моделювання змін в організаційних структурах тощо.

Спільний доступ. Система повинна надавати широкий спектр можливостей для розмежування доступу до даних та одночасної роботи декількох користувачів.

Багатовимірність. Система повинна забезпечувати концептуально багатовимірне представлення даних, включаючи повну підтримку декількох ієрархій.

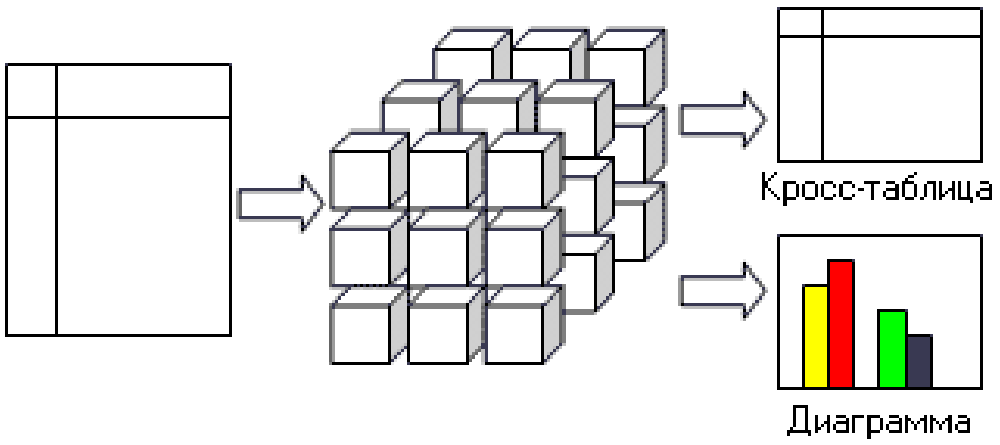


Рис. 1.6. Багатовимірне представлення даних

Інформація. Продуктивність різних програмних продуктів характеризується кількістю вхідних даних, які можуть бути оброблені. Різні OLAP-системи мають різну

продуктивність: найпотужніші можуть обробляти принаймні в тисячу разів більше даних, ніж найменш потужні. При виборі OLAP-інструменту необхідно враховувати низку факторів, зокрема дублювання даних, необхідний обсяг оперативної пам’яті, використання дискового простору, продуктивність, інтеграцію з інформаційними сховищами тощо.

Транзакційні системи оптимізовані для невеликих дискретних транзакцій. Однак запити на певну складну інформацію (наприклад, квартальна динаміка продажів певної моделі товару в певній галузі), характерні для аналітичних (OLAP) додатків, генеруватимуть складні з’єднання таблиць і перегляд цілих таблиць. Один такий запит займе багато комп’ютерного часу і ресурсів, сповільнюючи обробку поточних транзакцій. Тому дані передаються в OLAP-систему.

РОЗДІЛ 2
МЕТОДИ ТЕСТУВАННЯ ДАНИХ АНАЛІТИЧНИХ СИСТЕМ

2.1 Засоби переносу даних до сховищу даних

2.1.1 Реплікація бази даних

Реплікація бази даних – це, по суті, передача змін до інформації, зроблених в одному вузлі системи баз даних, на інші вузли. Реплікація характеризується автоматичною синхронізацією інформації, тобто без втручання користувача. З точки зору споживача інформації, добре організована реплікація означає, що дані доступні там і тоді, де і коли вони потрібні.

Сучасні системи СУБД підтримують різні типи реплікації даних. При виборі типу реплікації слід враховувати наступні фактори:

- Автономність (враховується рівень автономності або незалежності сервера для кожного вузла при розробці програми).
- Затримка (час, який витрачається на внесення змін на сервері публікації до того, як вони стануть доступними на сервері передплатника).
- Узгодженість даних:
    - збіжність даних (всі вузли виконують операції з однаковими результатами);
    - узгодженість транзакцій (дані на будь-якому сервері ідентичні, так, ніби всі транзакції виконуються на одному сервері).
- Узгодженість систем.

Головне завдання, однак, полягає в тому, щоб вибрати між двома основними критеріями для пошуку золотої середини:

- потреба в підключенні до мережі передачі даних на шкоду незалежності;
- необхідність мінімізації потенціалу конфліктів та максимальної автоматизації їх вирішення, якщо такі виникають.

Ось кілька популярних методів, кожен з яких має свої обмеження:

- Передача (знімок) і перезавантаження. Дані беруться з одного джерела і надсилаються одному або декільком одержувачам. Час між надходженням нових даних та їх розповсюдженням може вимірюватися годинами, днями та довшими періодами. Такий підхід забезпечує односторонню реплікацію без своєчасного віддаленого оновлення.
- Розповсюдження з двофазною перевіркою. У цьому випадку движок бази даних надсилає інформацію на цільові сервери під час кожної транзакції. Це значно збільшує час виконання транзакції, що робить такий підхід непридатним для завдань з високими вимогами до часу виконання.
- Знімки. У певні моменти часу робляться знімки бази даних, які потім надсилаються на один або кілька серверів одержувачів. У цьому випадку одержувачі працюють з відносно застарілими даними, що робить такий підхід неприйнятним у випадках, коли потрібна інформація в режимі реального часу. Крім того, цей метод не забезпечує віддаленого оновлення.

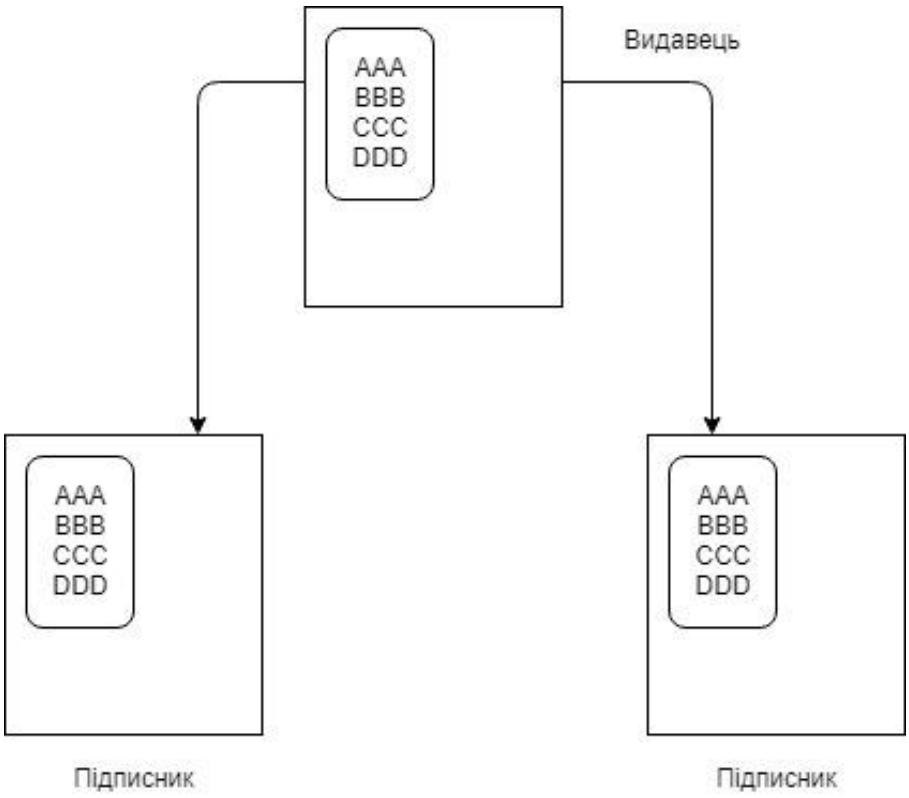


Рис. 2.1. Схема реплікації знімків

- Тригери без транзакцій. Тригер копіює дані одному або декільком віддаленим одержувачам без використання механізму транзакцій для забезпечення успішної доставки. Хоча накладні витрати зменшуються, це може легко призвести до ситуації, коли база даних не синхронізується, що може спричинити різні проблеми для одержувачів інформації.
- Тригери з транзакціями. Транзакції є перевіреним методом забезпечення цілісності при модифікації даних, а їх використання підвищує надійність тригерної реплікації. Ціною підвищення цілісності є зниження продуктивності програми через накладні витрати, пов’язані з виконанням транзакцій.
- На основі журналів. Цей метод, який використовується в Sybase Replication Server, є найбільш ефективним і економічно вигідним. Замість того, щоб споживати ресурси і сповільнювати роботу бази даних, реплікація на основі журналів безпосередньо зчитує онлайн журнали транзакцій.

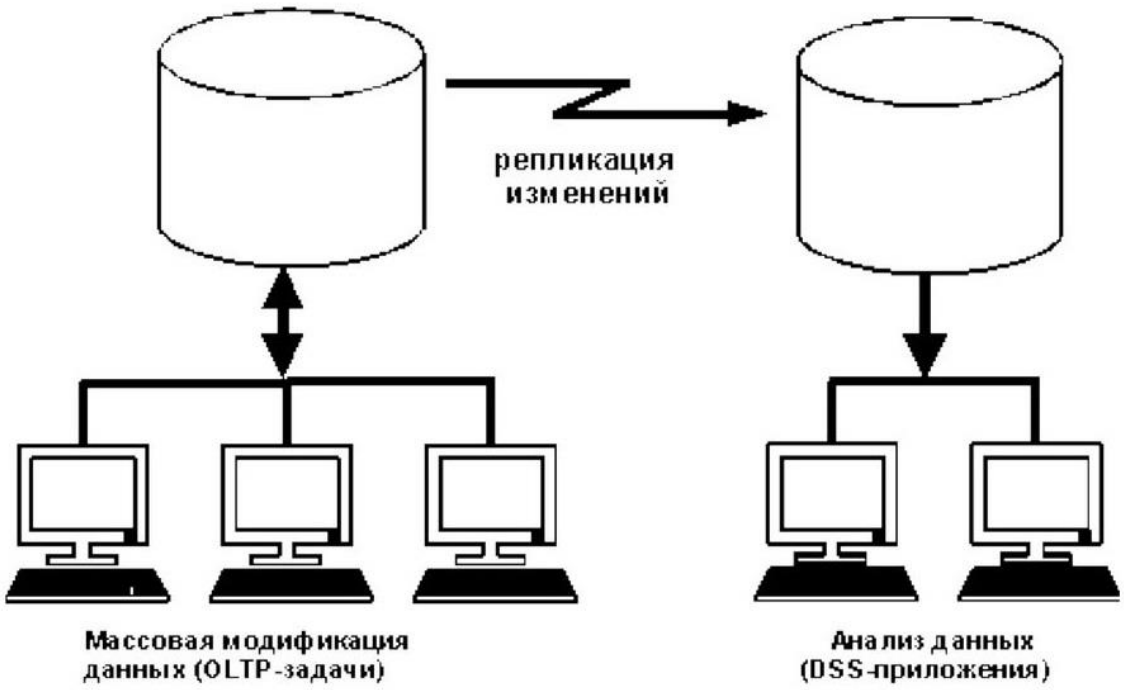


Рис. 2.2. Загальна схема реплікації даних

Після завершення реплікації копія даних стає доступною на сервері аналітичної системи. Виникає необхідність перевірити цілісність завантажених даних. Для цього дані необхідно протестувати. Існує кілька методів тестування таких даних.

2.1. ETL

ETL розшифровується як Extract-Transform-Load і являє собою процес завантаження даних з системи-джерела до сховища даних. Дані витягуються з бази даних OLTP, трансформуються відповідно до схеми сховища даних і завантажуються в базу даних сховища даних. Багато сховищ даних також містять дані з інших систем, крім OLTP, таких як текстові файли, застарілі системи та електронні таблиці.

Прикладом може бути роздрібний магазин, який має різні відділи, такі як продажі, маркетинг, логістика тощо. Кожен з них обробляє інформацію про клієнтів незалежно один від одного, і способи зберігання цих даних досить різні. Наприклад,

відділ продажів зберігає їх за іменами клієнтів, тоді як відділ маркетингу – за ідентифікатором клієнта.

Тепер, якщо вони матимуть бажання перевірити історію клієнта і з'ясувати, які різні продукти він купував у рамках різних маркетингових кампаній, це буде не дуже тривіальним завданням.

Рішення полягає у використанні сховища даних для зберігання інформації з різних джерел в єдиній структурі за допомогою ETL. ETL може перетворити різнорідні набори даних в єдину структуру. Пізніше, за допомогою інструментів бізнес-аналітики, з цих даних можна витягувати значущі дані та створювати звіти.

Рис. 2.3. ETL

1. Видалення

- Витяг відповідних даних

2. Конвертація

- Конвертація даних у формат DW (сховище даних)
- Ключі збірки. Ключ – це один або кілька атрибутів даних, які однозначно ідентифікують сутність. Існують різні типи ключів: первинний ключ, альтернативний ключ, зовнішній ключ, складений ключ, сурогатний ключ.

Сховище даних володіє цими ключами і ніколи не дозволяє іншим призначати їх.

- Очищення даних. Після вилучення даних відбувається наступний етап – очищення та узгодження даних. Під час очищення з даних видаляються пропуски, а також виявляються і виправляються помилки. Узгодження означає усунення невідповідностей між тими даними, які є несумісними, щоб їх можна було використовувати в корпоративному сховищі даних. Воно також створює метадані, які використовуються для діагностики проблем у вихідній системі та покращення якості даних.

3. Завантаження.

- Передача даних до DW (сховища даних)
- Створення агрегації. Агрегація – це об’єднання і зберігання даних, доступних у вигляді таблиці фактів, для підвищення продуктивності запитів кінцевих користувачів.

2.2 Тестування даних в таблицях

Процес тестування бази даних подібний до тестування інших додатків. Тестування бази даних можна описати за допомогою ключових процесів, перерахованих нижче.

- Конфігурація середовища
- Запуск тесту
- Перевірка результату тесту
- Підтвердження відповідності очікуваним результатам
- Повідомлення результатів відповідним зацікавленим сторонам.

Для створення тестових кейсів використовуються різні оператори SQL. Найпоширенішим оператором SQL, який використовується для тестування бази даних, є оператор Select. Крім того, можуть використовуватися різні оператори DDL, DML, DCL.

Валідація бази даних охоплює різні етапи життєвого циклу тестування бази даних відповідно до процесів тестування.

Основні етапи тестування бази даних:

- початкова перевірка стану;
- тестовий запуск;
- порівняння результатів з очікуваними результатами;
- видача результатів.

Першим етапом тестування бази даних є перевірка початкового стану бази даних перед початком процесу тестування. Потім поведінка бази даних тестується на конкретних тестових прикладах. Відповідно до результатів, тестові кейси коригуються.

Щоб успішно протестувати базу даних, слід дотримуватись наведеного робочого процесу для кожного тесту:

- очищення бази даних. Якщо база даних містить дані, які можна перевірити, її слід очистити;
- налаштування Fixture дозволяє вносити дані до бази даних і перевіряти поточний стан бази даних;
- запуск тесту, перевірка результатів та генерація результатів. Тест виконано і результат перевірено. Якщо результат відповідає очікуванням, наступним кроком буде генерація результатів на вимогу. В іншому випадку тест повторюється, щоб знайти помилки в базі даних.

Тестування схеми бази даних передбачає тестування кожного об’єкта у схемі. Потім тестуються бази даних пристроїв. Це включає в себе:

- перевірку імені бази даних;
- перевірку пристрою передачі даних, пристрою запису та пристрою скидання;
- перевірку наявності вільного місця для кожної бази даних;
- перевірку параметрів бази даних.

Потім перевіряються таблиці та стовпці. Щоб перевірити відмінності між фактичними та застосованими налаштуваннями, також перевіряються:

- назви всіх таблиць у базі даних;
- назви стовпців для кожної таблиці;
- типи стовпців для кожної таблиці;
- вказано або не вказано значення null;
- чи є стовпець таблиці неявно зв’язаним;
- визначення правил для корекції назв таблиць та прав доступу до них.

Також важливо перевірити валідність ключів та індексів. Слід перевірити ключі та індекси в кожній таблиці:

- первинний ключ для кожної таблиці;
- зовнішні ключі для кожної таблиці;
- типи даних між стовпцем зовнішнього ключа та стовпцем в іншій таблиці. Індекси, кластеризовані або некластеризовані, унікальні або неунікальні.

Тестування збережених процедур передбачає перевірку того, що збережена процедура визначена, і вихідні результати порівняні. При тестуванні збережених процедур перевіряються наступні елементи:

- Ім’я збереженої процедури
- Назви параметрів, типи параметрів тощо.
- Вихідний сигнал – вихідні дані містять багато записів. Виконуються нульові рядки або витягується лише декілька записів.
- Які функції виконує збережена процедура і чого не повинна робити збережена процедура?
- Надсилаються вибіркові вхідні запити для перевірки того, що збережена процедура містить коректні дані.
- Параметри збереженої процедури. Виклик збереженої процедури з граничними та допустимими даними. Кожен параметр перевизначається один раз відповідно до процедури.
- Значення, що повертаються – перевірка значень, що повертаються збереженою процедурою. У випадку невдачі повертається нульове значення.

- Перевірка повідомлень про помилки – внесення змін для того, щоб збережена процедура не запускалася і видавала кожне повідомлення про помилку хоча б один раз. Необхідно перевірити всі сценарії винятків, якщо немає умовного повідомлення про помилку.

Реплікація даних – це перший крок у побудові OLAP-моделі. Таке тестування дає можливість виявити помилки на ранній стадії проектування, ще до початку роботи над побудовою OLAP-моделі. В іншому випадку модель може бути побудована некоректно, що призведе до значних фінансових втрат.

2.3. Міграція моделі даних та її тестування

У компаніях, де впроваджуються OLAP-системи, швидше за все, вже існує звітність на основі даних з ERP-системи або застарілої OLAP-системи. Якщо програмне забезпечення майбутньої OLAP-системи підтримує автоматичну міграцію зі старої OLAP-системи, немає необхідності створювати нову систему. Іншими словами, переноситься вся модель даних аналітичної системи. Однак необхідно протестувати таку систему, щоб переконатися, що міграція працює належним чином.

При створенні нової моделі підхід полягає в тому, щоб порівняти загальні дані, наявні в старих системах, з новими. При міграції моделі даних важливими аспектами є контроль:

- Якість даних. Може статися так, що дані, взяті зі старої/застарілої системи, мають низьку якість у новій системі. У таких випадках якість даних необхідно покращити, щоб вони відповідали бізнес-стандартам. Ці фактори можуть призвести до низької якості даних. Це має наслідком високі операційні витрати, підвищений ризик інтеграції даних і відхилення від бізнес-цілей.
- Неузгодженість даних. Дані, перенесені із застарілої системи в нову/оновлену систему, можуть бути несумісними в новій системі. Це може бути пов’язано зі зміною типу даних, формату зберігання даних, мети використання даних. Це призводить до величезних зусиль з доопрацювання необхідних змін, щоб виправити невідповідні дані або прийняти їх і адаптувати для цієї мети.

- Втрата даних. Дані можуть бути втрачені під час перенесення із застарілої до нової/оновленої системи. Це може статися з обов'язковими або необов'язковими полями. Якщо втрачені дані стосуються необов'язкових полів, запис для них все одно буде дійсним і його можна буде оновити знову. Однак, якщо дані з обов'язкових полів будуть втрачені, сам запис стане недійсним і не може бути скасований. Це призведе до величезної втрати даних, і їх доведеться відновлювати з резервної копії бази даних або з журналів аудиту, якщо вони записані правильно.
- Обсяг даних. Величезні обсяги даних, які потребують багато часу для переміщення в рамках простою під час операції міграції. Наприклад: скретч-картки в телекомунікаційній галузі, користувачі на платформі розумних мереж тощо. Проблема полягає в тому, що застарілі дані видаляються, створюються величезні нові дані, які потрібно мігрувати знову. Автоматизація – це рішення для міграції величезних обсягів даних.

Під час тестування звіту, перенесеного із застарілої аналітичної системи, тестувальник порівнює всі релевантні дані в обох версіях системи, використовуючи описані вище принципи.

2.4. Тестування моделі даних на відповідність специфікаціям

Бувають випадки, коли потрібно створити абсолютно новий звіт. При цьому компанія не завжди може мати звіти зі старих OLAP або інших систем. У такому випадку складається специфікація моделі даних, яка допоможе розробникам створити модель даних.

На основі специфікацій, відповідно до яких створено звіт, необхідно його протестувати. Тестувальники створюють серію SQL-запитів, які покривають зазначені специфікації. Пізніше виконується SQL-скрипт, за допомогою якого тестувальники отримують дані про відповідність звіту специфікаціям.

2.5. Тестування моделі даних на основі legacy OLAP.

Інформація, що міститься у звіті, часто міститься в різних корпоративних джерелах даних. Ці джерела даних можуть бути на основі SQL (реляційні бази даних) або не на основі SQL, такі як XML і OLAP. Для того, щоб опублікувати змістовну інформацію, часто необхідно об’єднати дані з одного або декількох таких джерел даних. Наприклад, може знадобитися об’єднати наявні дані в реляційній базі даних з даними з багатовимірної бази даних, щоб порівняти тенденції та показники.

Цей тип тестування виконується, коли дані в новому звіті можна звірити з кількома старими звітами OLAP або ERP.

Загалом, цей процес тестування схожий на процес тестування перенесеної моделі даних. Однак цей процес довший і складніший, оскільки дані для нового звіту містяться в різних старих звітах.

2.6. Тестування моделі даних на основі декількох існуючих

Цей тест виконується, коли є спільні дані між новим звітом і раніше повністю протестованими звітами в новій OLAP-системі.

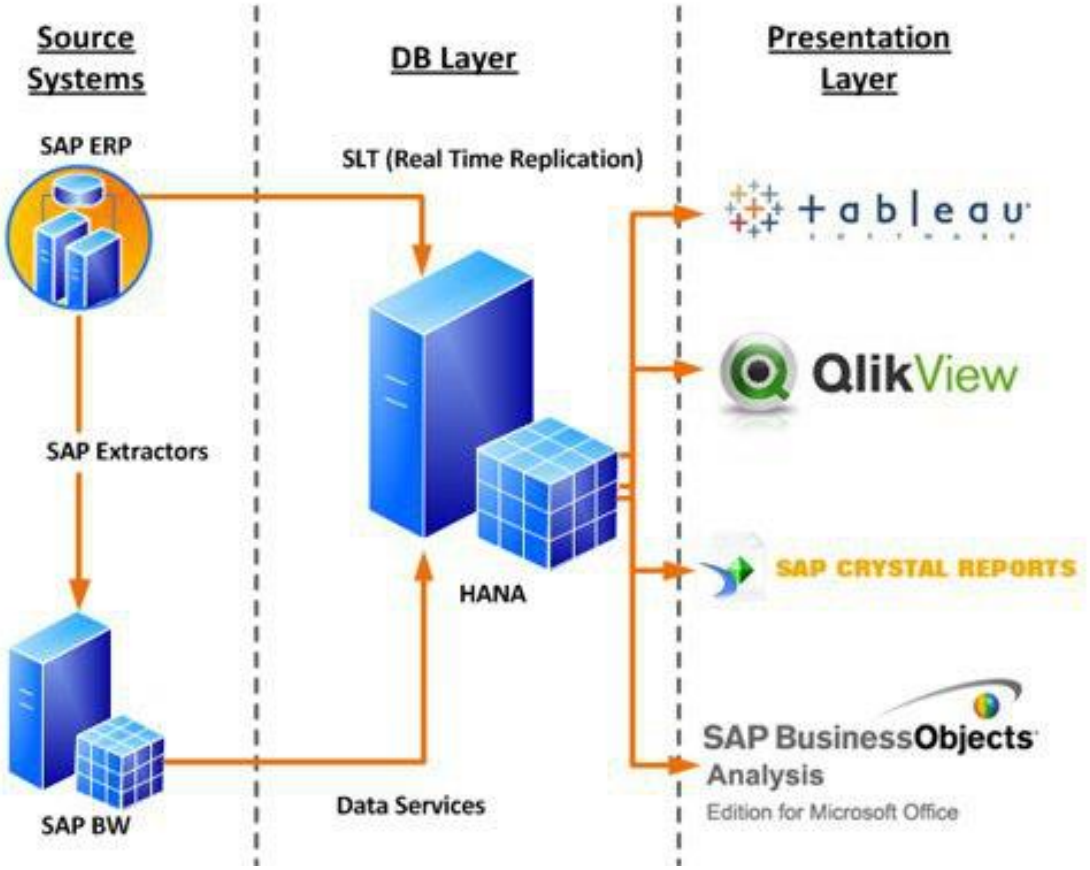
Тестування нового звіту з використанням іншого звіту, який був протестований в тій же базі даних, є кращим способом, аніж використання звітів з різних баз даних, оскільки це займає набагато менше часу.

РОЗДІЛ 3

Реалізаця методів тестування

3.1. Середовище – SAP

Для цього проекту були обрані наступні технології, які наразі використовуються та підтримуються існуючою інфраструктурою, та реалізовані з використанням класичного сховища даних (розмірне моделювання):



ETL : SAP SLT, SAP Business Objects Data Services 4.2

Очікувані джерела даних включають наступні платформи та контент:

Компоненти SAP HANA складаються з бази даних HANA в пам’яті та HANA Studio, яка є інструментом для розробки та моделювання даних. Дані зберігаються в таблицях бази даних HANA і можуть бути доступні через три типи представлень:

- Атрибут – початковий шар перегляду для відображення головної та довідкової таблиць
- Аналітичний – проміжний рівень подання для зв’язування таблиць транзакцій з головними та довідковими таблицями
- Обчислення – останній шар подання для застосування обчислень, підсумків і сполучників.

3.2. Тестування даних методом порівняння таблиць

Для реалізації цього методу було обрано одну з низки таблиць.

Така таблиця містить багато мільйонів записів. Тому порівняти всі дані вручну неможливо. Нижче наведено ту частину таблиці, яка є в ERP-системі.

IHPA (Plant Maintenance: Partners) – це стандартна таблиця в ERP-системах SAP.

Data Browser: Table IHPA Select Entries 6

Check Table...

MAN...	OBJNR	PARVW	COUNTER	OBT...	PARNR	INH...	ERDAT	ERZEIT
030	PR00053559	Z4	1	PRN	00213241		07/13/2011	11:32:24
030	PR00053559	Z4	2	PRN	00120309		07/13/2011	11:32:24
030	PR00053559	Z4	3	PRN	00258352		07/13/2011	12:13:24
030	PR00053559	ZC	37	PRN	00190632		05/26/2011	10:03:43
030	PR00053559	ZC	38	PRN	00224032		08/26/2011	10:56:23
030	PR00053559	ZL	12	PRN	00191109		05/26/2011	10:03:43

Рис. 3.1. Частина таблиці IHPA

Таблиця була вибірково реплікована в SAP HANA. Нижче наведено частину реплікації в SAP HANA:

MANDT	OBJNR	PARVW	COUNTER	OBTYP	PARNR	INHER	ERDAT	ERZEIT
010	PR02580598	ZC	000001	PRN	00237572		20171216	060146
010	PR02580643	ZC	000001	PRN	00237572		20171216	060146
010	PR03255200	ZC	000001	PRN	00237572		20171216	060306
010	PR03255201	ZC	000001	PRN	00237572		20171216	060306
010	PR03255202	ZC	000001	PRN	00237572		20171216	060306
010	PR03255203	ZC	000001	PRN	00237572		20171216	060306

Рис. 3.2. Частина таблиці-репліки IHPA

Щоб валідувати коректність реплікації даних, перевіряється назва поля, тип поля, кількість полів і ключів.

Існує два способи порівняння даних:

1. Одночасно відкриваються ідентичні вибірки даних з десятків рядків, застосувавши фільтри. Дані порівнюються візуально, аби переконатися, що реплікація пройшла успішно.
2. Використовуючи SAP HANA, деякі дані переносяться в EXCEL і порівнюються за допомогою функцій.

   Другий метод дозволяє порівнювати набагато більше даних, ніж перший.

Описаний набір дій повторюється для кожної реплікованої таблиці.

3.3. Тестування даних на основі legacy OLAP

У нашому прикладі SAP HANA використовується у сценарії «side-car», який передбачає перенесення всієї аналітики з SAP ERP та SAP BW до SAP HANA, що забезпечує можливості аналітики в режимі реального часу. Це означає, що для таких аналітичних звітів у SAP HANA будуть отримані ті самі звіти, що й у legacy системі SAP BW або SAP ERP. Така legacy аналітика є джерелом для узгодження новостворених аналітичних моделей в SAP HANA, що значно полегшує завдання тестування якості даних нових моделей.

Перед тестуванням нової моделі даних SAP HANA аналізуються старі звіти BW та ERP, щоб знайти ті, які містять ті самі показники, що й нова модель, яка тестується. Далі ці моделі, SAP HANA і старіші моделі, запускаються з параметрами, які генерують однакові набори даних, а потім набори даних порівнюються. Порівняння можна зробити візуально або завантажити дані з цих джерел, наприклад, у форматі Excel, а потім завантажити і порівняти їх за допомогою інструментів Excel.

Розглянемо такий звіт, як «Key Financial Ratios» (Основні фінансові показники). Цей звіт включений в ERP-систему, тому можна перевірити якість даних для цього звіту в SAP HANA, використовуючи існуючий звіт в ERP.

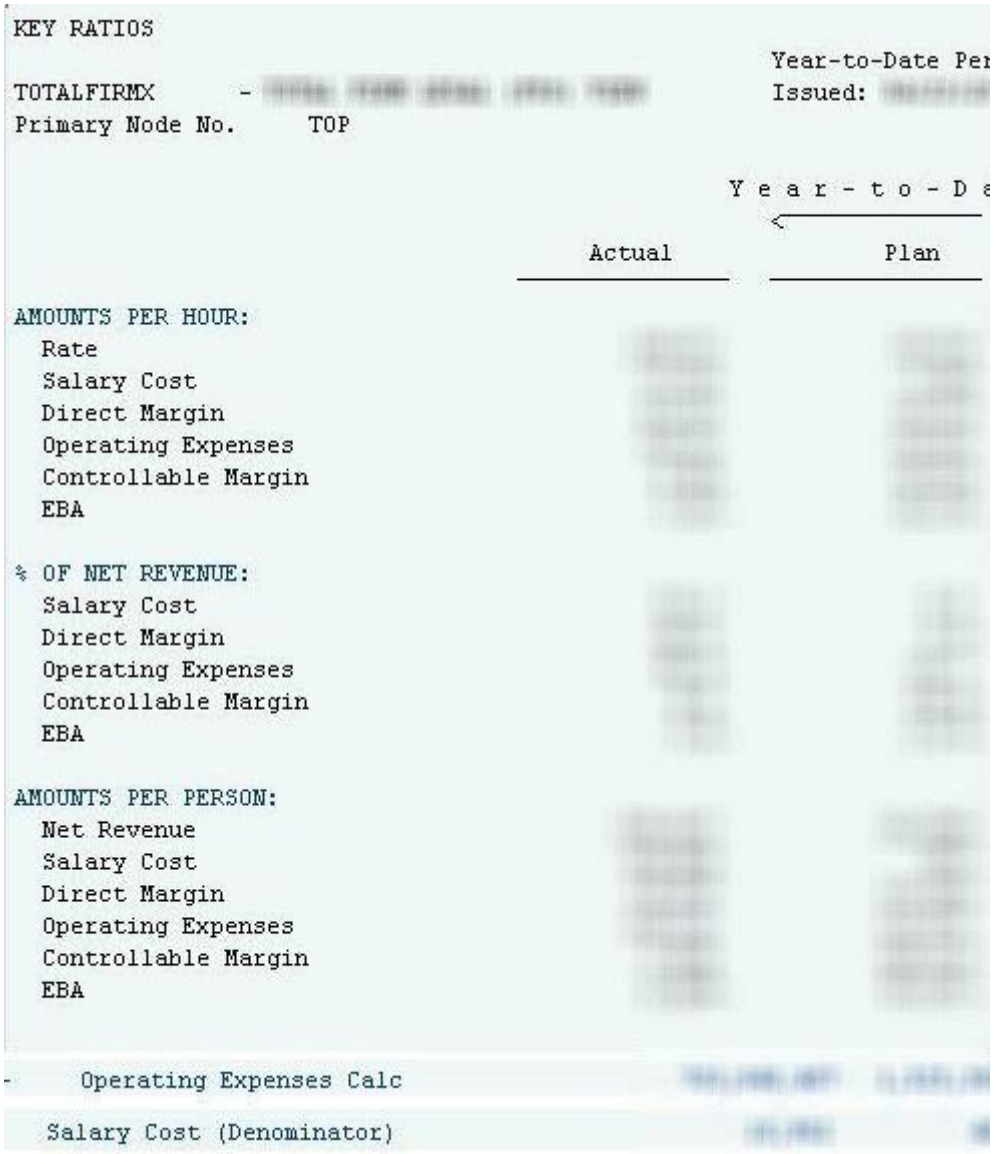
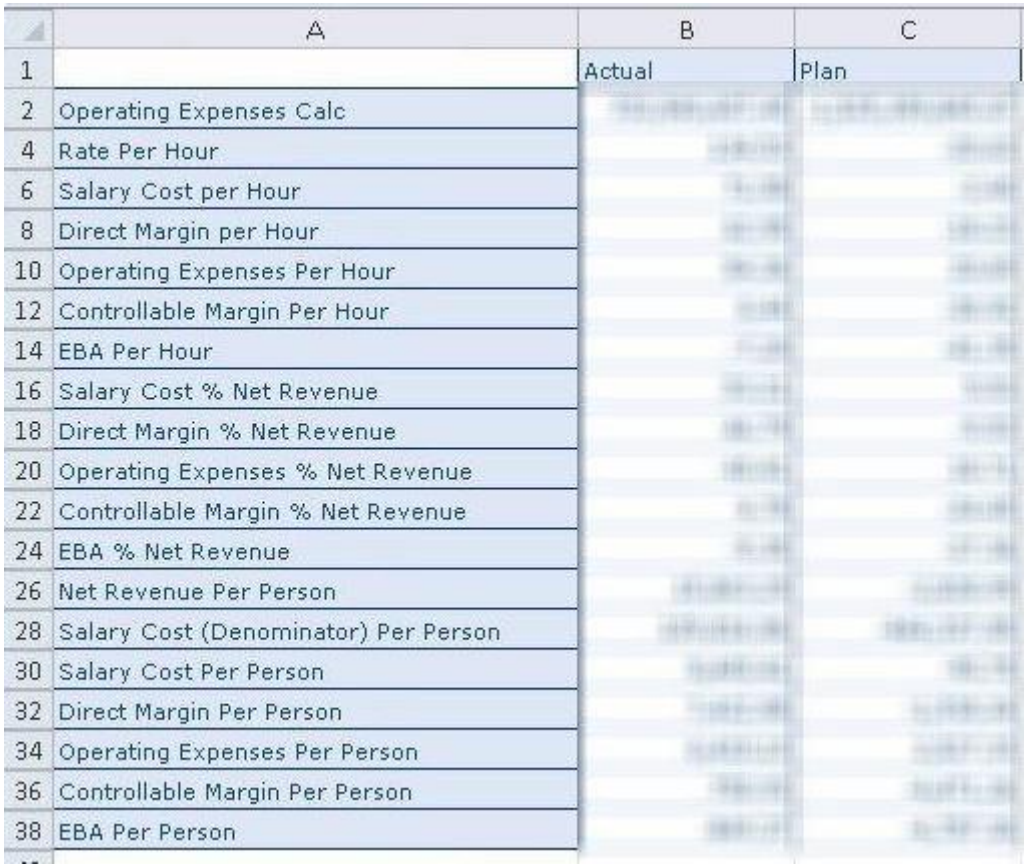


Рис. 3.3. Ключові фінансові показники зі звіту ERP

Звіт SAP HANA міститься у файлі Excel, створеному за допомогою SAP Aanalysis for Office.



	A	B	C
1		Actual	Plan
2	Operating Expenses Calc		
4	Rate Per Hour		
6	Salary Cost per Hour		
8	Direct Margin per Hour		
10	Operating Expenses Per Hour		
12	Controllable Margin Per Hour		
14	EBA Per Hour		
16	Salary Cost % Net Revenue		
18	Direct Margin % Net Revenue		
20	Operating Expenses % Net Revenue		
22	Controllable Margin % Net Revenue		
24	EBA % Net Revenue		
26	Net Revenue Per Person		
28	Salary Cost (Denominator) Per Person		
30	Salary Cost Per Person		
32	Direct Margin Per Person		
34	Operating Expenses Per Person		
36	Controllable Margin Per Person		
38	EBA Per Person		

Рис. 3.4. Ключові фінансові показники у звіті SAP HANA

Для прикладу розглянемо інші звіти: «Чисельність персоналу та використання ресурсів», «Графік доходів», «Інші ключові години».

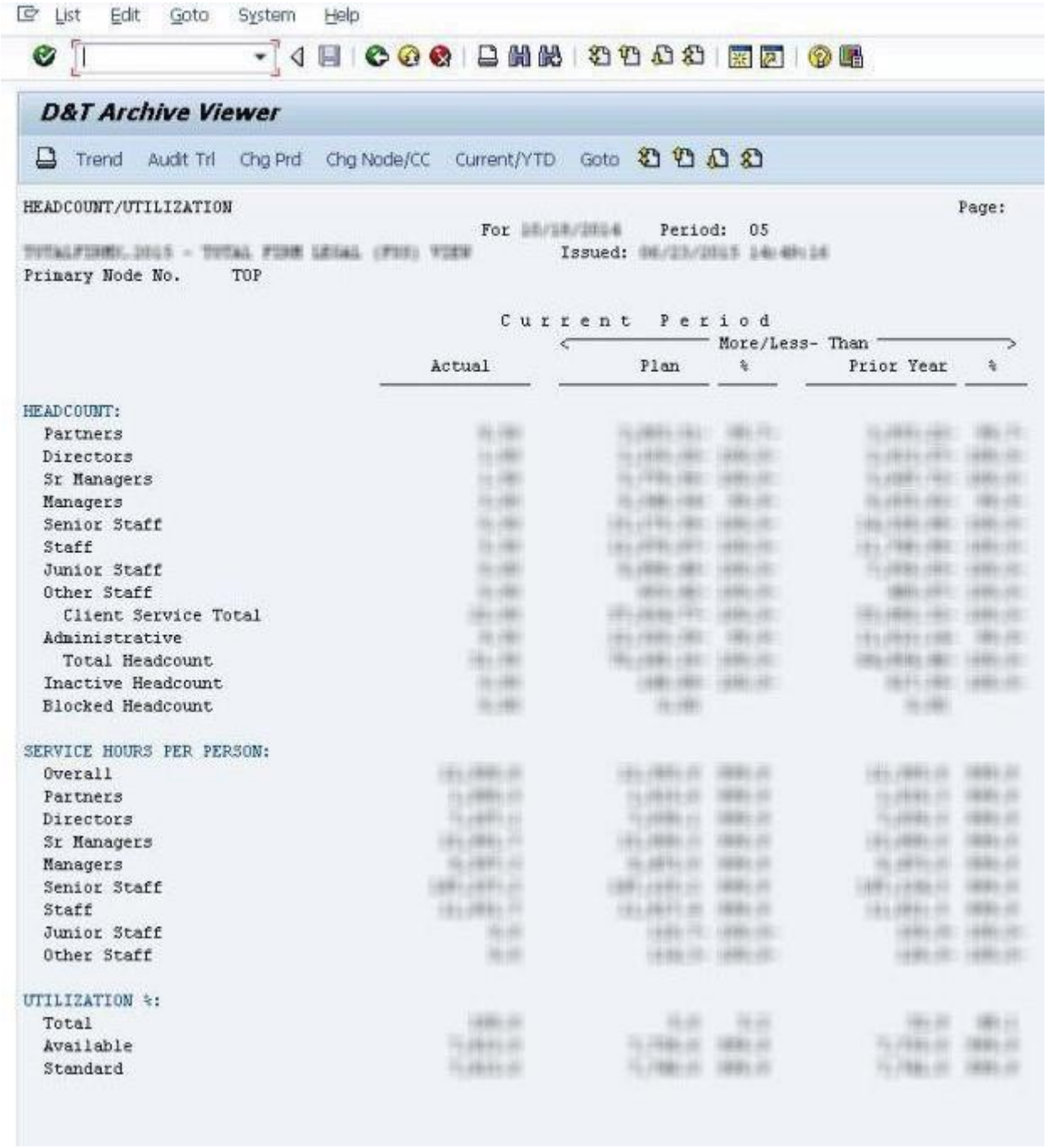
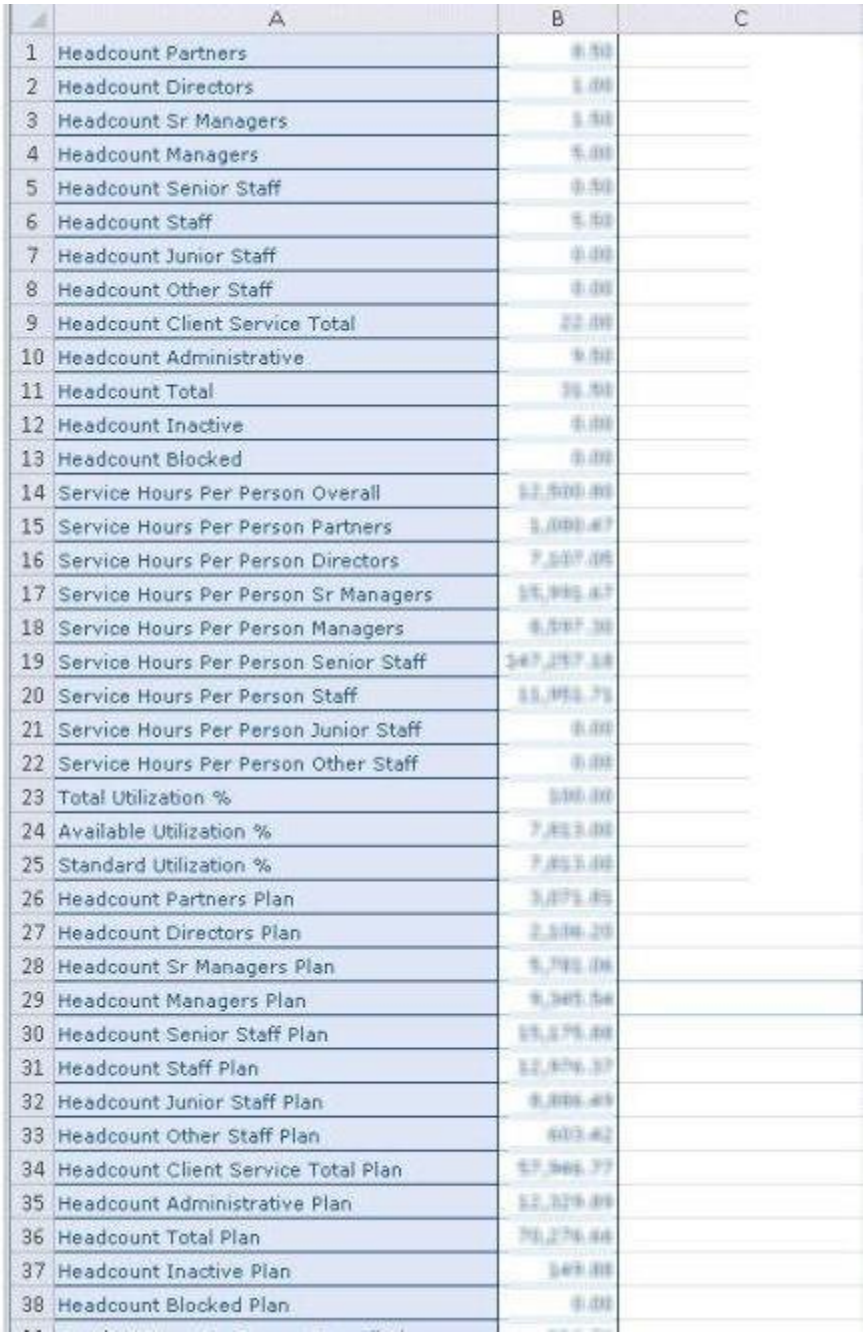


Рис. 3.5. Звіт «Зайнятість та використання ресурсів» з ERP-системи



	A	B	C
1	Headcount Partners		
2	Headcount Directors		
3	Headcount Sr Managers		
4	Headcount Managers		
5	Headcount Senior Staff		
6	Headcount Staff		
7	Headcount Junior Staff		
8	Headcount Other Staff		
9	Headcount Client Service Total		
10	Headcount Administrative		
11	Headcount Total		
12	Headcount Inactive		
13	Headcount Blocked		
14	Service Hours Per Person Overall		
15	Service Hours Per Person Partners		
16	Service Hours Per Person Directors		
17	Service Hours Per Person Sr Managers		
18	Service Hours Per Person Managers		
19	Service Hours Per Person Senior Staff		
20	Service Hours Per Person Staff		
21	Service Hours Per Person Junior Staff		
22	Service Hours Per Person Other Staff		
23	Total Utilization %		
24	Available Utilization %		
25	Standard Utilization %		
26	Headcount Partners Plan		
27	Headcount Directors Plan		
28	Headcount Sr Managers Plan		
29	Headcount Managers Plan		
30	Headcount Senior Staff Plan		
31	Headcount Staff Plan		
32	Headcount Junior Staff Plan		
33	Headcount Other Staff Plan		
34	Headcount Client Service Total Plan		
35	Headcount Administrative Plan		
36	Headcount Total Plan		
37	Headcount Inactive Plan		
38	Headcount Blocked Plan		

Рис. 3.6. Звіт «Зайнятість та використання ресурсів» з SAP HANA

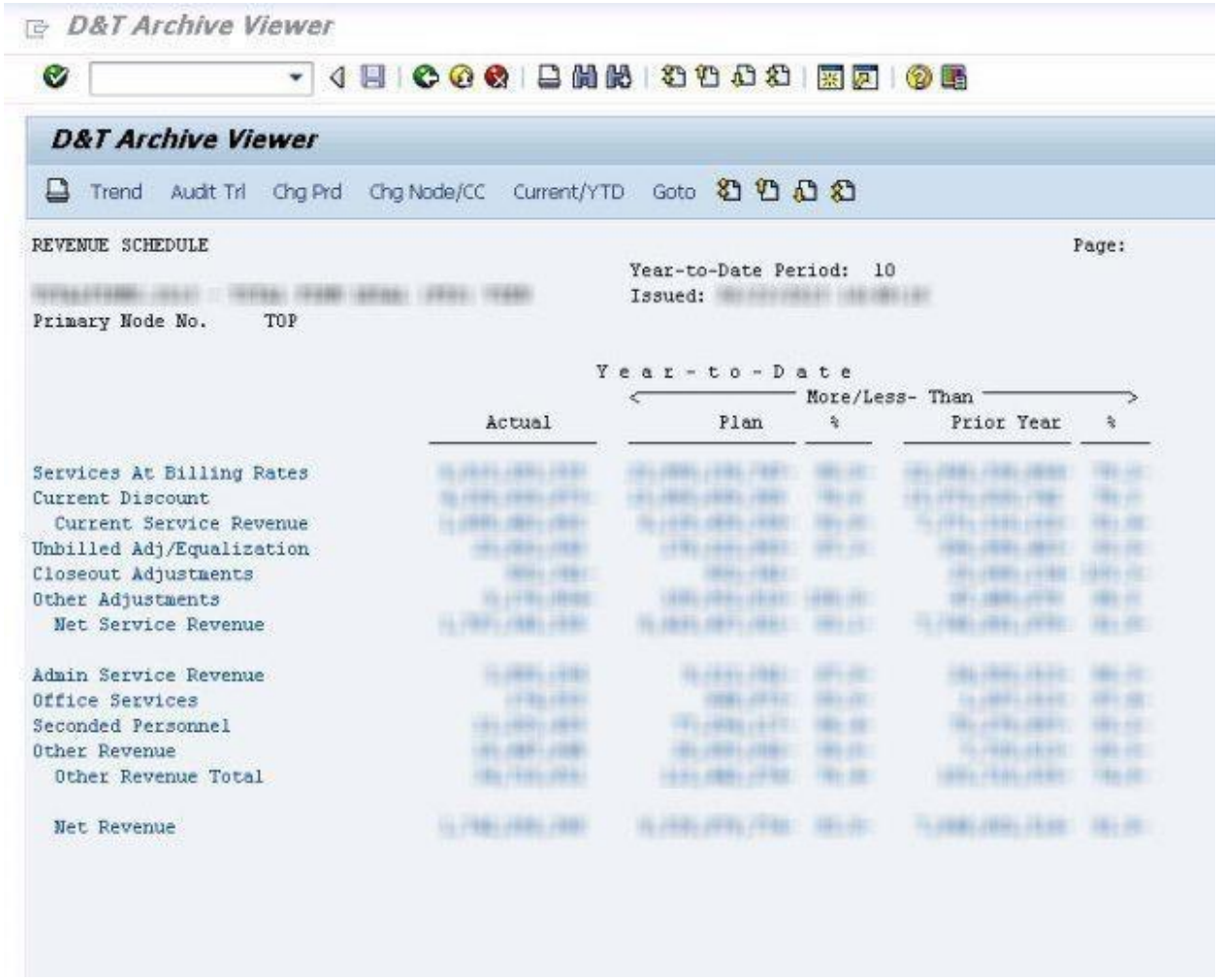


Рис. 3.7. Звіт «Графік надходжень» з ERP-системи

	A	B	C
1	Services At Billing Rates		
2	Current Discount		
3	Current Service Revenue		
4	Unbilled Adj/Equalization		
5	Closeout Adjustments		
6	Other Adjustments		
7	Net Service Revenue		
8	Admin Service Revenue		
9	Office Services		
10	Seconded Personnel		
11	Other Revenue		
12	Other Revenue Total		
13	Net Revenue		
14	Services At Billing Rates Plan		
15	Current Discount Plan		
16	Current Service Revenue Plan		
17	Unbilled Adj/Equalization Plan		
18	Closeout Adjustments Plan		
19	Other Adjustments Plan		
20	Net Service Revenue Plan		
21	Admin Service Revenue Plan		
22	Office Services Plan		
23	Seconded Personnel Plan		
24	Other Revenue Plan		
25	Other Revenue Total Plan		
26	Net Revenue Plan		

Рис. 3.8. Звіт «Графік надходжень» з SAP HANA

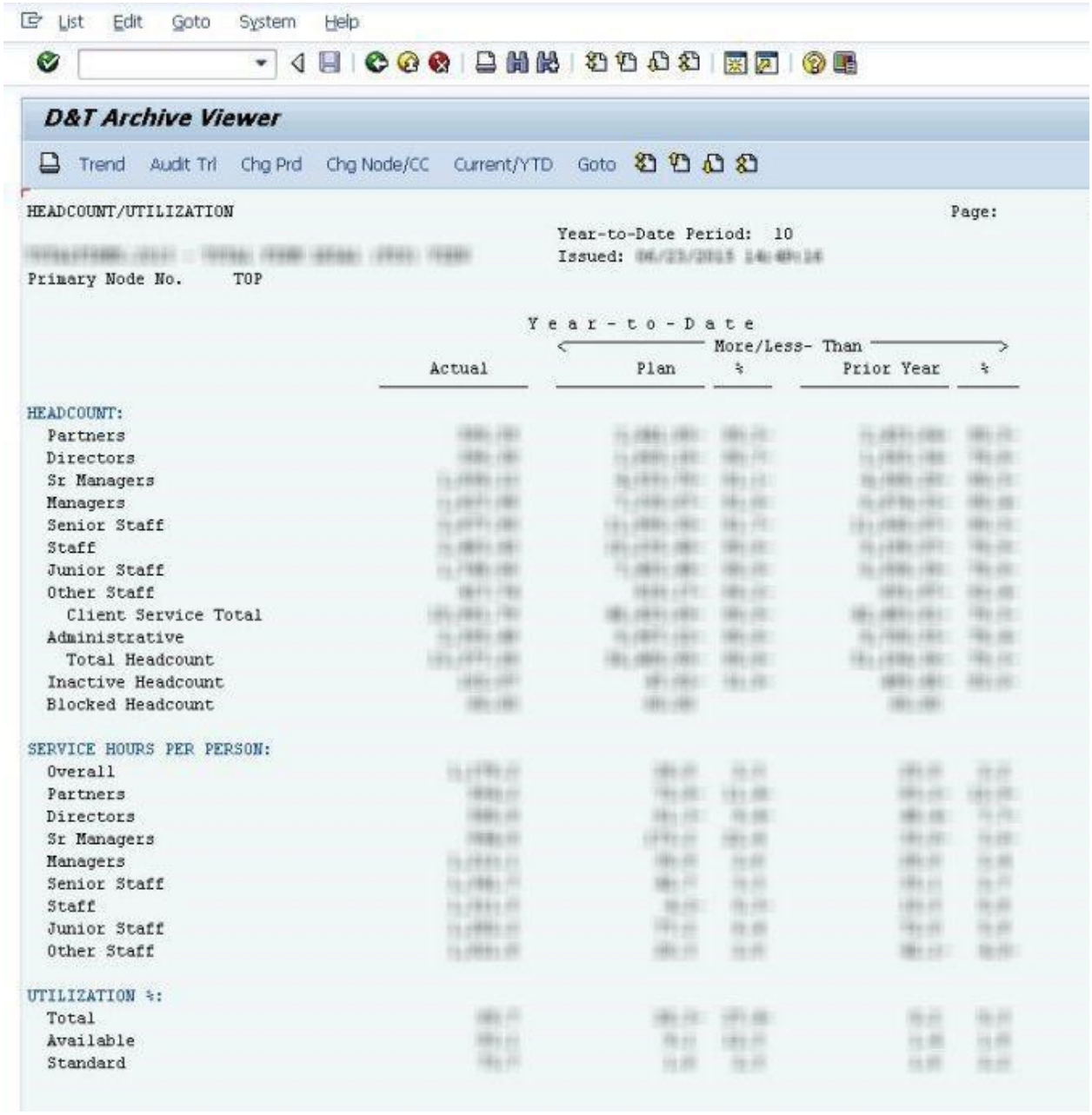


Рис. 3.9. Звіт «Зайнятість та використання ресурсів» з ERP-системи (інший період)

	A	B	C
1	Headcount Partners	[illegible]	
2	Headcount Directors	[illegible]	
3	Headcount Sr Managers	[illegible]	
4	Headcount Managers	[illegible]	
5	Headcount Senior Staff	[illegible]	
6	Headcount Staff	[illegible]	
7	Headcount Junior Staff	[illegible]	
8	Headcount Other Staff	[illegible]	
9	Headcount Client Service Total	[illegible]	
10	Headcount Administrative	[illegible]	
11	Headcount Total	[illegible]	
12	Headcount Inactive	[illegible]	
13	Headcount Blocked	[illegible]	
14	Service Hours Per Person Overall	[illegible]	
15	Service Hours Per Person Partners	[illegible]	
16	Service Hours Per Person Directors	[illegible]	
17	Service Hours Per Person Sr Managers	[illegible]	
18	Service Hours Per Person Managers	[illegible]	
19	Service Hours Per Person Senior Staff	[illegible]	
20	Service Hours Per Person Staff	[illegible]	
21	Service Hours Per Person Junior Staff	[illegible]	
22	Service Hours Per Person Other Staff	[illegible]	
23	Total Utilization %	[illegible]	
24	Available Utilization %	[illegible]	
25	Standard Utilization %	[illegible]	
26	Headcount Partners Plan	[illegible]	
27	Headcount Directors Plan	[illegible]	
28	Headcount Sr Managers Plan	[illegible]	
29	Headcount Managers Plan	[illegible]	
30	Headcount Senior Staff Plan	[illegible]	

Рис. 3.10. Звіт «Зайнятість та використання ресурсів» з SAP HANA (інший період)

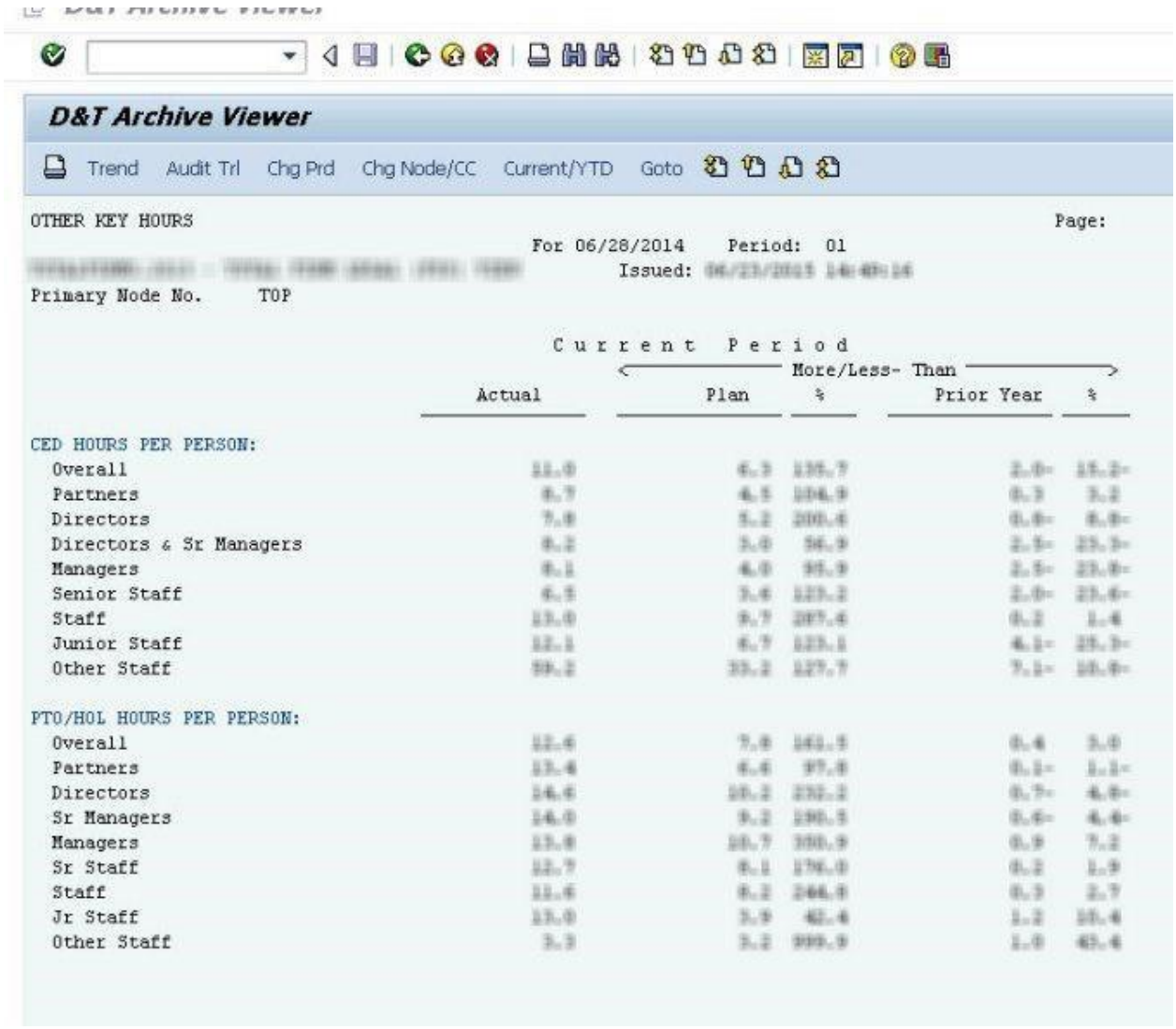


Рис. 3.11. Звіт «Інші ключові години» з ERP-системи

	A	B	C	D
1	CED Hours Per Person Overall			
2	CED Hours Per Person Partners			
3	CED Hours Per Person Directors			
4	CED Hours Per Person Sr Managers			
5	CED Hours Per Person Managers			
6	CED Hours Per Person Senior Staff			
7	CED Hours Per Person Staff			
8	CED Hours Per Person Junior Staff			
9	CED Hours Per Person Other Staff			
10	PTO/Holiday Hours Per Person Overall			
11	PTO/Holiday Hours Per Person Partners			
12	PTO/Holiday Hours Per Person Directors			
13	PTO/Holiday Hours Per Person Sr Managers			
14	PTO/Holiday Hours Per Person Managers			
15	PTO/Holiday Hours Per Person Sr Staff			
16	PTO/Holiday Hours Per Person Staff			
17	PTO/Holiday Hours Per Person Jr Staff			
18	PTO/Holiday Hours Per Person Other Staff			
19	CED Hours Per Person Overall Plan			
20	CED Hours Per Person Partners Plan			
21	CED Hours Per Person Directors Plan			
22	CED Hours Per Person Sr Managers Plan			
23	CED Hours Per Person Managers Plan			
24	CED Hours Per Person Senior Staff Plan			
25	CED Hours Per Person Staff Plan			
26	CED Hours Per Person Junior Staff Plan			
27	CED Hours Per Person Other Staff Plan			
28	PTO/Holiday Hours Per Person Overall Plan			
29	PTO/Holiday Hours Per Person Partners Plan			
30	PTO/Holiday Hours Per Person Directors Plan			
31	PTO/Holiday Hours Per Person Sr Managers Plan			
32	PTO/Holiday Hours Per Person Managers Plan			
33	PTO/Holiday Hours Per Person Sr Staff Plan			
34	PTO/Holiday Hours Per Person Staff Plan			
35	PTO/Holiday Hours Per Person Jr Staff Plan			
36	PTO/Holiday Hours Per Person Other Staff Plan			

Рис. 3.12. Звіт "Інші ключові години" з SAP HANA

Такий підхід до тестування розроблюваної моделі шляхом порівняння даних зі старими звітами з систем BW та ERP дозволяє перевірити якість даних з меншими витратами часу та ресурсів.

3.4 Тестування даних за специфікацією

У процесі розробки нових моделей даних нерідко виявляється, що не існує джерела, наприклад, існуючого звіту, на основі якого можна було б перевірити достовірність значень індикаторів нової моделі.

У таких ситуаціях для перевірки даних можна використовувати наступний підхід. За наявною специфікацією, відповідно до якої розробляються моделі даних, створюються тестові SQL-запити, які повторюють логіку специфікації для розрахунку показників. Результат виконання цих запитів потім порівнюється з результатом виконання моделі даних, що розробляється, для того ж набору даних.

Нижче наведено приклад такого запиту, який обчислює показник «Other Adjustments» (Інші коригування), використовуючи таблиці, що відповідають специфікації, і порівнює результат зі значенням, яке повертає тестована модель даних. Якщо дані в двох тестових джерелах збігаються, то запит не поверне жодних записів, інакше будуть повернуті записи з відмінностями.

```
SELECT COUNT(*) FROM (
SELECT RACCT, RCNTR, -SUM(KSL)
FROM "SOURCE_ERP_SLT"."ZZDTA"
WHERE RYEAR = 2016 AND POPER BETWEEN 9 AND 9 AND RRCTY =
0 AND RACCT IN
  (
    SELECT ACCT
    FROM
    "SOURCE_FINANCE"."company.fin::PCS_ACCOUNT"
    WHERE "REP_ID" = 'PER_ID_2017' AND "ROW" = 760
  )
```

```
      AND RCNTR IN
      (
        SELECT DISTINCT "COST_CENTER"
        FROM "_SYS_BIC"."company.fin/COSTCENTER_SNAP"
        WHERE "CC_HIER_NODE" = 'SOME NODE'
      )
    GROUP BY RACCT, RCNTR
    HAVING SUM("KSL")<>0
    )
    EXCEPT
    SELECT COUNT(*) FROM (
    SELECT
    RACCT,
    RCNTR,
    SUM("ROW760_ACTL") --Other Adjustments
    FROM "SOURCE_FINANCE"."company.fin/FINDATA_ACTUAL"
    WHERE "FYP" BETWEEN  '2016009' AND '2016009' AND "REP_ID"
= 'PER_ID_2017'
    AND RCNTR IN
      (
        SELECT DISTINCT "COST_CENTER"
        FROM "_SYS_BIC"."company.fin/COSTCENTER_SNAP"
        WHERE "CC_HIERARCHY_NODE_COMBINED" = 'SOME NODE'
```

```
  )
GROUP BY
RACCT,
RCNTR
HAVING SUM("ROW760_ACTL")<>0
```

Такий підхід до перевірки моделі, що розробляється, шляхом створення тестових запитів, які повторюють логіку, описану в специфікації, є ресурсо- та часозатратним, але дозволяє перевірити якість даних за відсутності джерела верифікації.

3.5. Тестування даних на основі декількох існуючих.

Для тестування якості даних новостворених моделей даних можна використовувати існуючі моделі, які вже були протестовані.

Можна навести такий приклад: у процесі роботи над проектом компанії була створена модель даних SAP HANA Calculation View для кінцевого користувача «Profitability Report» (Звіт про прибутковість), яка надалі була успішно протестована за допомогою цього ж звіту в SAP ERP. Потім був створений ще один звіт «Profitability Rolling 13 Report». У цьому випадку для тестування даних нового звіту можна використовувати звіт, який тестувався «Profitability Report» (Звіт про прибутковість), оскільки всі показники нового звіту, який тестується, можуть бути розраховані на основі звіту, який тестувався. Нижче наведено приклад скрипта, який тестує такі звіти.

Зразок сценарію можна знайти в Додатку А.

Тестування нового звіту за допомогою іншого звіту, який був протестований в тій же базі даних, є кращим способом, ніж використання звітів з різних баз даних, оскільки це займає набагато менше часу.

ВИСНОВКИ

У ході виконання дипломної роботи був розроблений веб-додаток для автоматизації роботи магазину, використовуючи сучасні технології та мікросервісну архітектуру. Цей процес включав декілька ключових етапів:

1. Огляд існуючих рішень для автоматизації торгівлі. У першому розділі дипломної роботи було проведено детальний аналіз існуючих рішень для автоматизації роботи магазинів. Були проаналізовані їхні переваги та недоліки, на основі чого було зроблено висновок про доцільність розробки специфікованого веб-додатку, який би відповідав конкретним потребам магазину.

2. Розробка веб-додатку. У другому розділі описано процес розробки веб-додатку, включаючи формулювання вимог до системи, проектування архітектури, бази даних та інтерфейсу, розробку основних функціональних модулів, а також тестування та верифікацію системи. Велика увага була приділена забезпеченню гнучкості та масштабованості системи.

3. Впровадження та порівняльний аналіз. У третьому розділі були наведені рекомендації щодо впровадження розробленого веб-додатку у практику роботи магазину. Також був проведений порівняльний аналіз переваг розробленого рішення у порівнянні з іншими універсальними платформами автоматизації.

Результатом дипломної роботи стало створення комплексного, спеціалізованого веб-додатку, який дозволяє автоматизувати основні бізнес-процеси магазину, підвищити його ефективність та конкурентоспроможність. Цей додаток був розроблений із зосередженням на специфічних потребах конкретного магазину, що робить його більш відповідним для цільового бізнесу, ніж універсальні рішення.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Байкарова О. О. Інформаційні технології – засіб оптимізації діяльності підприємств / О. О. Байкарова, Л. М. Тарасюк // Комп'ютерно-інтегровані технології : освіта, наука, виробництво. – 2013. – №11. – С. 177 – 182.

2. Вовчак І.С. Інформаційні системи та комп'ютерні технології в менеджменті : навч. посіб. / І. С. Вовчак. – Тернопіль : Карт-бланш, 2001. – 354 с.

3. Жежнич П.І. Технології інформаційного менеджменту : навч. посіб. / П.І. Жежнич. — Львів : Львівська політехніка, 2010. — 260 с.

4. Куденко Н. В. Менеджмент – управління інформацією / Навч. посібн. – К: КДТЕУ, 1999. – 313 с.

5. Лук'яненко Д. І. Розвиток інформаційного менеджменту як наукової категорії / Д. І. Лук'яненко // Науковий вісник Полтавського університету економіки і торгівлі. Сер.: Економічні науки. – 2013. – № 1. – С. 183–187.

6. Лисак В.М. Теоретичні аспекти автоматизації процесів збирання економічної інформації для управління підприємством // Вісник Хмельницького національного університету, - Хмельницький, 2008, № 5, Т. 2(119).

7. Принципи вибору програмного забезпечення для вирішення різних завдань управління персоналом. URL: https://studfile.net/preview/2398774/page:65/.

8. Результати дослідження ринку CRM в Україні. URL: https://www.bitrix24.ua (дата звернення 10.01.2019).

9. CRM-системи стали найбільшим сегментом ринку в 2017 році. URL: https://news.finance.ua (дата звернення 09.01.2019).

10. Фадєєва І.Г. Розвиток концептуальних засад автоматизованого аналітичного управління бізнес-процесами // Вісник Хмельницького національного університету, - Хмельницький, 2008, № 5, Т. 2(119).

ДОДАТКИ

Додаток А

```
    SELECT

    SUM(SERVICE_HOURS_R13),

    SUM(SERVICE_HOURS_PR13),

    SUM(SERVICE_HOURS_R13)-SUM(SERVICE_HOURS_PR13)         AS
SERVICE_HOURS_R13_GROWTH,

    (SUM(SERVICE_HOURS_R13)-
SUM(SERVICE_HOURS_PR13))/NULLIF(SUM(SERVICE_HOURS_PR13),0)
AS SERVICE_HOURS_R13_PCT_GROWTH,

    SUM(TOTAL_HRS_R13),

    SUM(TOTAL_HRS_PR13),

    SUM(TOTAL_HRS_R13)-SUM(TOTAL_HRS_PR13)                 AS
TOTAL_HRS_R13_GROWTH,

    (SUM(TOTAL_HRS_R13)-
SUM(TOTAL_HRS_PR13))/NULLIF(SUM(TOTAL_HRS_PR13),0)         AS
TOTAL_HRS_R13_PCT_GROWTH,

    SUM(GROSS_SERVICES_R13),

    SUM(GROSS_SERVICES_PR13),

    SUM(GROSS_SERVICES_R13)-SUM(GROSS_SERVICES_PR13)       AS
GROSS_SERVICES_R13_GROWTH,

    (SUM(GROSS_SERVICES_R13)-
SUM(GROSS_SERVICES_PR13))/NULLIF(SUM(GROSS_SERVICES_PR13),0)
AS GROSS_SERVICES_R13_PCT_GROWTH,

    SUM(UNBILLED_ADJS_R13),
```

```
    SUM(UNBILLED_ADJS_PR13),

    SUM(UNBILLED_ADJS_R13)-SUM(UNBILLED_ADJS_PR13)          AS
UNBILLED_ADJS_R13_GROWTH,

    (SUM(UNBILLED_ADJS_R13)-
SUM(UNBILLED_ADJS_PR13))/NULLIF(SUM(UNBILLED_ADJS_PR13),0)
AS UNBILLED_ADJS_R13_PCT_GROWTH,

    SUM(NET_SERVICE_REVENUES_R13),

    SUM(NET_SERVICE_REVENUES_PR13),

    SUM(NET_SERVICE_REVENUES_R13)-
SUM(NET_SERVICE_REVENUES_PR13)                          AS
NET_SERVICE_REVENUES_R13_GROWTH,

    (SUM(NET_SERVICE_REVENUES_R13)-
SUM(NET_SERVICE_REVENUES_PR13))/NULLIF(SUM(NET_SERVICE_REVEN
UES_PR13),0) AS NET_SERVICE_REVENUES_R13_PCT_GROWTH,

    SUM(OTHER_REVS_R13),

    SUM(OTHER_REVS_PR13),

    IFNULL  (SUM(OTHER_REVS_R13),0)-SUM(OTHER_REVS_PR13)  AS
OTHER_REVS_R13_GROWTH,

    (IFNULL                          (SUM(OTHER_REVS_R13),0)-
SUM(OTHER_REVS_PR13))/NULLIF(SUM(OTHER_REVS_PR13),0)      AS
OTHER_REVS_R13_PCT_GROWTH,


    SUM(NET_REVENUES_R13),

    SUM(NET_REVENUES_PR13),

    SUM(NET_REVENUES_R13)-SUM(NET_REVENUES_PR13)          AS
NET_REVENUES_R13_GROWTH,
```

```
    (SUM(NET_REVENUES_R13)-
SUM(NET_REVENUES_PR13))/NULLIF(SUM(NET_REVENUES_PR13),0)   AS
NET_REVENUES_R13_PCT_GROWTH,

    SUM(US_SERVICE_LABOR_COSTS_R13),

    SUM(US_SERVICE_LABOR_COSTS_PR13),

    SUM(US_SERVICE_LABOR_COSTS_R13)-
SUM(US_SERVICE_LABOR_COSTS_PR13)                              AS
US_SERVICE_LABOR_COSTS_R13_GROWTH,

    (SUM(US_SERVICE_LABOR_COSTS_R13)-
SUM(US_SERVICE_LABOR_COSTS_PR13))/NULLIF(SUM(US_SERVICE_LABO
R_COSTS_PR13),0) AS US_SERVICE_LABOR_COSTS_R13_PCT_GROWTH

    SUM(R10_SERVICE_LABOR_COSTS_R13),

    SUM(R10_SERVICE_LABOR_COSTS_PR13),

    SUM(R10_SERVICE_LABOR_COSTS_R13)-
SUM(R10_SERVICE_LABOR_COSTS_PR13)                             AS
R10_SERVICE_LABOR_COSTS_R13_GROWTH,

    (SUM(R10_SERVICE_LABOR_COSTS_R13)-
SUM(R10_SERVICE_LABOR_COSTS_PR13))/NULLIF(SUM(R10_SERVICE_LA
BOR_COSTS_PR13),0) AS R10_SERVICE_LABOR_COSTS_R13_PCT_GROWTH,

    SUM(SERVICE_LABOR_COST_R13),

    SUM(SERVICE_LABOR_COST_PR13),

    SUM(SERVICE_LABOR_COST_R13)-SUM(SERVICE_LABOR_COST_PR13)
AS SERVICE_LABOR_COST_R13_GROWTH,

    (SUM(SERVICE_LABOR_COST_R13)-
SUM(SERVICE_LABOR_COST_PR13))/NULLIF(SUM(SERVICE_LABOR_COST_
PR13),0) AS SERVICE_LABOR_COST_R13_PCT_GROWTH,
```

```
    SUM(ADMIN_LABOR_COST_R13),

    SUM(ADMIN_LABOR_COST_PR13),

    SUM(ADMIN_LABOR_COST_R13)-SUM(ADMIN_LABOR_COST_PR13)   AS
ADMIN_LABOR_COST_R13_GROWTH,

    (SUM(ADMIN_LABOR_COST_R13)-
SUM(ADMIN_LABOR_COST_PR13))/NULLIF(SUM(ADMIN_LABOR_COST_PR13
),0) AS ADMIN_LABOR_COST_R13_PCT_GROWTH

    SUM(PROV_FOR_BAD_DEBT_R13),

    SUM(PROV_FOR_BAD_DEBT_PR13),

    SUM(PROV_FOR_BAD_DEBT_R13)-SUM(PROV_FOR_BAD_DEBT_PR13)
AS PROV_FOR_BAD_DEBT_R13_GROWTH,

    (SUM(PROV_FOR_BAD_DEBT_R13)-
SUM(PROV_FOR_BAD_DEBT_PR13))/NULLIF(SUM(PROV_FOR_BAD_DEBT_PR
13),0) AS PROV_FOR_BAD_DEBT_R13_PCT_GROWTH,

    SUM(INTEREST_CHARGE_R13),

    SUM(INTEREST_CHARGE_PR13),

    SUM(INTEREST_CHARGE_R13)-SUM(INTEREST_CHARGE_PR13)     AS
INTEREST_CHARGE_R13_GROWTH,

    (SUM(INTEREST_CHARGE_R13)-
SUM(INTEREST_CHARGE_PR13))/NULLIF(SUM(INTEREST_CHARGE_PR13),
0) AS INTEREST_CHARGE_R13_PCT_GROWTH,

    SUM(DIRECT_PROJECT_COSTS_R13),

    SUM(DIRECT_PROJECT_COSTS_PR13),

    SUM(DIRECT_PROJECT_COSTS_R13)-
SUM(DIRECT_PROJECT_COSTS_PR13)                              AS
DIRECT_PROJECT_COSTS_R13_GROWTH,
```

```
    (SUM(DIRECT_PROJECT_COSTS_R13)-
SUM(DIRECT_PROJECT_COSTS_PR13))/NULLIF(SUM(DIRECT_PROJECT_CO
STS_PR13),0) AS DIRECT_PROJECT_COSTS_R13_PCT_GROWTH,

    SUM(GROSS_CLIENT_MARGIN_R13),

    SUM(GROSS_CLIENT_MARGIN_PR13),

    SUM(GROSS_CLIENT_MARGIN_R13)-
SUM(GROSS_CLIENT_MARGIN_PR13)                                AS
GROSS_CLIENT_MARGIN_R13_GROWTH,

    (SUM(GROSS_CLIENT_MARGIN_R13)-
SUM(GROSS_CLIENT_MARGIN_PR13))/NULLIF(SUM(GROSS_CLIENT_MARGI
N_PR13),0) AS GROSS_CLIENT_MARGIN_R13_PCT_GROWTH,

    SUM(PRD_HOURS_R13),

    SUM(PRD_HOURS_PR13),

    SUM(PRD_HOURS_R13)-SUM(PRD_HOURS_PR13)                   AS
PRD_HOURS_R13_GROWTH,

    (SUM(PRD_HOURS_R13)-
SUM(PRD_HOURS_PR13))/NULLIF(SUM(PRD_HOURS_PR13),0)           AS
PRD_HOURS_R13_PCT_GROWTH

    SUM(US_PRD_LABOR_COSTS_R13),

    SUM(US_PRD_LABOR_COSTS_PR13),

    SUM(US_PRD_LABOR_COSTS_R13)-SUM(US_PRD_LABOR_COSTS_PR13)
AS US_PRD_LABOR_COSTS_R13_GROWTH,

    (SUM(US_PRD_LABOR_COSTS_R13)-
SUM(US_PRD_LABOR_COSTS_PR13))/NULLIF(SUM(US_PRD_LABOR_COSTS_
PR13),0) AS US_PRD_LABOR_COSTS_R13_PCT_GROWTH,

    SUM(R10_PRD_LABOR_COSTS_R13),
```

```
    SUM(R10_PRD_LABOR_COSTS_PR13),

    SUM(R10_PRD_LABOR_COSTS_R13)-
SUM(R10_PRD_LABOR_COSTS_PR13)                                    AS
R10_PRD_LABOR_COSTS_R13_GROWTH,

    (SUM(R10_PRD_LABOR_COSTS_R13)-
SUM(R10_PRD_LABOR_COSTS_PR13))/NULLIF(SUM(R10_PRD_LABOR_COST
S_PR13),0) AS R10_PRD_LABOR_COSTS_R13_PCT_GROWTH,

    SUM(PRD_LABOR_COST_R13),

    SUM(PRD_LABOR_COST_PR13),

    SUM(PRD_LABOR_COST_R13)-SUM(PRD_LABOR_COST_PR13)       AS
PRD_LABOR_COST_R13_GROWTH,

    (SUM(PRD_LABOR_COST_R13)-
SUM(PRD_LABOR_COST_PR13))/NULLIF(SUM(PRD_LABOR_COST_PR13),0)
AS PRD_LABOR_COST_R13_PCT_GROWTH,

    SUM(PRD_EXPENSE_R13),

    SUM(PRD_EXPENSE_PR13),

    SUM(PRD_EXPENSE_)-SUM(PRD_EXPENSE_PR13)                AS
PRD_EXPENSE_R13_GROWTH,

    (SUM(PRD_EXPENSE_R13)-
SUM(PRD_EXPENSE_PR13))/NULLIF(SUM(PRD_EXPENSE_PR13),0)     AS
PRD_EXPENSE_R13_PCT_GROWTH,


    SUM(PRD_LABOR_PLS_EXP_R13),

    SUM(PRD_LABOR_PLS_EXP_ R13PR13),

    SUM(PRD_LABOR_PLS_EXP_R13)-SUM(PRD_LABOR_PLS_EXP_PR13)
AS PRD_LABOR_PLS_EXP_R13_GROWTH,
```

```
    (SUM(PRD_LABOR_PLS_EXP_R13)-
SUM(PRD_LABOR_PLS_EXP_PR13))/NULLIF(SUM(PRD_LABOR_PLS_EXP_PR
13),0) AS PRD_LABOR_PLS_EXP_R13_PCT_GROWTH,

    SUM(PRD_CLIENT_MARGIN_R13),

    SUM(PRD_CLIENT_MARGIN_PR13),

    SUM(PRD_CLIENT_MARGIN_R13)-SUM(PRD_CLIENT_MARGIN_PR13)
AS PRD_CLIENT_MARGIN_R13_GROWTH,

    (SUM(PRD_CLIENT_MARGIN_R13)-
SUM(PRD_CLIENT_MARGIN_PR13))/NULLIF(SUM(PRD_CLIENT_MARGIN_PR
13),0) AS PRD_CLIENT_MARGIN_R13_PCT_GROWTH

    SUM(FSS_OVERHEAD_R13),

    SUM(FSS_OVERHEAD_PR13),

    SUM(FSS_OVERHEAD_R13)-SUM(FSS_OVERHEAD_PR13)          AS
FSS_OVERHEAD_R13_GROWTH,

    (SUM(FSS_OVERHEAD_R13)-
SUM(FSS_OVERHEAD_PR13))/NULLIF(SUM(FSS_OVERHEAD_PR13),0)   AS
FSS_OVERHEAD_R13_PCT_GROWTH,

    SUM(SHARED_OVERHEAD_R13),

    SUM(SHARED_OVERHEAD_PR13),

    SUM(SHARED_OVERHEAD_R13)-SUM(SHARED_OVERHEAD_PR13)    AS
SHARED_OVERHEAD_R13_GROWTH,

    (SUM(SHARED_OVERHEAD_R13)-
SUM(SHARED_OVERHEAD_PR13))/NULLIF(SUM(SHARED_OVERHEAD_PR13),
0) AS SHARED_OVERHEAD_R13_PCT_GROWTH,

    SUM(Audit_Claims_Insurance_Costs_R13),

    SUM(Audit_Claims_Insurance_Costs_PR13),
```

```
    SUM(Audit_Claims_Insurance_Costs_R13)-
SUM(Audit_Claims_Insurance_Costs_PR13)                    AS
Audit_Claims_Insurance_Costs_R13_GROWTH,

    (SUM(Audit_Claims_Insurance_Costs_R13)-
SUM(Audit_Claims_Insurance_Costs_PR13))/NULLIF(SUM(Audit_Cla
ims_Insurance_Costs_PR13),0)                              AS
Audit_Claims_Insurance_Costs_R13_PCT_GROWTH,

    SUM(TOTAL_OVERHEAD_R13),

    SUM(TOTAL_OVERHEAD_PR13),

    SUM(TOTAL_OVERHEAD_R13)-SUM(TOTAL_OVERHEAD_PR13)      AS
TOTAL_OVERHEAD_R13_GROWTH,

    (SUM(TOTAL_OVERHEAD_R13)-
SUM(TOTAL_OVERHEAD_PR13))/NULLIF(SUM(TOTAL_OVERHEAD_PR13),0)
AS TOTAL_OVERHEAD_R13_PCT_GROWTH,

    SUM(NET_CLIENT_MARGIN_R13),

    SUM(NET_CLIENT_MARGIN_PR13),

    SUM(NET_CLIENT_MARGIN_R13)-SUM(NET_CLIENT_MARGIN_PR13)
AS NET_CLIENT_MARGIN_R13_GROWTH,

    (SUM(NET_CLIENT_MARGIN_R13)-
SUM(NET_CLIENT_MARGIN_PR13))/NULLIF(SUM(NET_CLIENT_MARGIN_PR
13),0) AS NET_CLIENT_MARGIN_R13_PCT_GROWTH,

    SUM(PARTNER_SERVICES_LABOR_COSTS_R13),

    SUM(PARTNER_SERVICES_LABOR_COSTS_PR13),

    SUM(PARTNER_SERVICES_LABOR_COSTS_R13)-
SUM(PARTNER_SERVICES_LABOR_COSTS_PR13)                    AS
PARTNER_SERVICES_LABOR_COSTS_R13_GROWTH,
```

```
    (SUM(PARTNER_SERVICES_LABOR_COSTS_R13)-
SUM(PARTNER_SERVICES_LABOR_COSTS_PR13))/NULLIF(SUM(PARTNER_S
ERVICES_LABOR_COSTS_PR13),0)                                AS
PARTNER_SERVICES_LABOR_COSTS_R13_PCT_GROWTH,


    SUM(PARTNER_PRD_LABOR_COSTS_R13),

    SUM(PARTNER_PRD_LABOR_COSTS_PR13),

    SUM(PARTNER_PRD_LABOR_COSTS_R13)-
SUM(PARTNER_PRD_LABOR_COSTS_PR13)                           AS
PARTNER_PRD_LABOR_COSTS_R13_GROWTH,

    (SUM(PARTNER_PRD_LABOR_COSTS_R13)-
SUM(PARTNER_PRD_LABOR_COSTS_PR13))/NULLIF(SUM(PARTNER_PRD_LA
BOR_COSTS_PR13),0) AS PARTNER_PRD_LABOR_COSTS_R13_PCT_GROWTH,

    SUM(PARTNER_LABOR_COST_R13),

    SUM(PARTNER_LABOR_COST_PR13),

    SUM(PARTNER_LABOR_COST_R13)-SUM(PARTNER_LABOR_COST_PR13)
AS PARTNER_LABOR_COST_R13_GROWTH,

    (SUM(PARTNER_LABOR_COST_R13)-
SUM(PARTNER_LABOR_COST_PR13))/NULLIF(SUM(PARTNER_LABOR_COST_
PR13),0) AS PARTNER_LABOR_COST_R13_PCT_GROWTH,

    SUM(TOTAL_PTR_HRS_R13),

    SUM(TOTAL_PTR_HRS_PR13),

    SUM(TOTAL_PTR_HRS_R13)-SUM(TOTAL_PTR_HRS_PR13)          AS
TOTAL_PTR_HRS_R13_GROWTH,
```

```
    (SUM(TOTAL_PTR_HRS_R13)-
SUM(TOTAL_PTR_HRS_PR13))/NULLIF(SUM(TOTAL_PTR_HRS_PR13),0)
AS TOTAL_PTR_HRS_R13_PCT_GROWTH,

    SUM(CALC_DISC_PCT_R13),

    SUM(CALC_DISC_PCT_PR13),

    SUM(CALC_DISC_PCT_R13)-SUM(CALC_DISC_PCT_PR13)         AS
CALC_DISC_PCT_R13_GROWTH,

    SUM(CALC_GCM_PCT_R13),

    SUM(CALC_GCM_PCT_PR13),

    SUM(CALC_GCM_PCT_R13)-SUM(CALC_GCM_PCT_PR13)           AS
CALC_GCM_PCT_R13_GROWTH,

    SUM(CALC_PRD_CM_PCT_R13),

    SUM(CALC_PRD_CM_PCT_PR13),

    SUM(CALC_PRD_CM_PCT_R13)-SUM(CALC_PRD_CM_PCT_PR13)     AS
CALC_PRD_CM_PCT_R13_GROWTH,

    SUM(CALC_NCM_PCT_R13),

    SUM(CALC_NCM_PCT_PR13),

    SUM(CALC_NCM_PCT_R13)-SUM(CALC_NCM_PCT_PR13)           AS
CALC_NCM_PCT_R13_GROWTH,

    SUM(CALC_NET_REV_PER_HOUR_R13),

    SUM(CALC_NET_REV_PER_HOUR_PR13),

    SUM(CALC_NET_REV_PER_HOUR_R13)-
SUM(CALC_NET_REV_PER_HOUR_PR13)                            AS
CALC_NET_REV_PER_HOUR_R13_GROWTH,

    SUM(CALC_PARTNER_LEVERAGE_R13),
```

```
    SUM(CALC_PARTNER_LEVERAGE_PR13),

    SUM(CALC_PARTNER_LEVERAGE_R13)-
SUM(CALC_PARTNER_LEVERAGE_PR13)                                    AS
CALC_PARTNER_LEVERAGE_R13_GROWTH,

    SUM(CALC_STAFFING_INDEX_R13),

    SUM(CALC_STAFFING_INDEX_PR13),

    SUM(CALC_STAFFING_INDEX_R13)-
SUM(CALC_STAFFING_INDEX_PR13)                                      AS
CALC_STAFFING_INDEX_R13_GROWTH,

    SUM(CALC_USI_PCT_OF_HOURS_R13),

    SUM(CALC_USI_PCT_OF_HOURS_PR13),

    SUM(CALC_USI_PCT_OF_HOURS_R13)-
SUM(CALC_USI_PCT_OF_HOURS_PR13)                                    AS
CALC_USI_PCT_OF_HOURS_R13_GROWTH


    FROM

    (

    SELECT WBS,

    SUM(Admin_Labor_Cost) AS Admin_Labor_Cost_R13,

    SUM(Audit_Claims_Insurance_Costs)                              AS
Audit_Claims_Insurance_Costs_R13,

    SUM(Direct_Project_Costs) AS Direct_Project_Costs_R13,

    SUM(FSS_Overhead) AS FSS_Overhead_R13,

    SUM(Gross_Client_Margin) AS Gross_Client_Margin_R13,

    SUM(Gross_Services) AS Gross_Services_R13,
```

```
    SUM(Interest_Charge) AS Interest_Charge_R13,
    SUM(Net_Client_Margin) AS Net_Client_Margin_R13,
    SUM(Net_Revenues) AS Net_Revenues_R13,
    SUM(Net_Service_Revenues) AS Net_Service_Revenues_R13,
    SUM(Other_Revs) AS Other_Revs_R13,
    SUM(Partner_Labor_Cost) AS Partner_Labor_Cost_R13,
    SUM(Partner_PRD_Labor_Costs) AS
Partner_PRD_Labor_Costs_R13,
    SUM(Partner_Services_Labor_Costs) AS
Partner_Services_Labor_Costs_R13,
    SUM(PRD_Client_Margin) AS PRD_Client_Margin_R13,
    SUM(PRD_Expense) AS PRD_Expense_R13,
    SUM(PRD_Hours) AS PRD_Hours_R13,
    SUM(PRD_Labor_Cost) AS PRD_Labor_Cost_R13,
    SUM(PRD_Labor_pls_Exp) AS PRD_Labor_pls_Exp_R13,
    SUM(Prov_for_Bad_Debt) AS Prov_for_Bad_Debt_R13,
    SUM(R10_PRD_Labor_Costs) AS R10_PRD_Labor_Costs_R13,
    SUM(R10_Service_Labor_Costs) AS
R10_Service_Labor_Costs_R13,
    SUM(Service_Hours) AS Service_Hours_R13,
    SUM(Service_Labor_Cost) AS Service_Labor_Cost_R13,
    SUM(Shared_Overhead) AS Shared_Overhead_R13,
    SUM(TOTAL_HRS) AS TOTAL_HRS_R13,
    SUM(Total_Overhead) AS Total_Overhead_R13,
```

```
    SUM(TOTAL_PTR_HRS) AS TOTAL_PTR_HRS_R13,
    SUM(Unbilled_Adjs) AS UNBILLED_ADJS_R13,
    SUM(US_PRD_Labor_Costs) AS US_PRD_Labor_Costs_R13,
    SUM(US_Service_Labor_Costs)                                    AS
US_Service_Labor_Costs_R13,
    SUM(CALC_DISC_PCT) AS CALC_DISC_PCT_R13,
    SUM(CALC_GCM_PCT) AS CALC_GCM_PCT_R13,
    SUM(CALC_NCM_PCT) AS CALC_NCM_PCT_R13,
    SUM(CALC_NET_REV_PER_HOUR) AS CALC_NET_REV_PER_HOUR_R13,
    SUM(CALC_PARTNER_LEVERAGE) AS CALC_PARTNER_LEVERAGE_R13,
    SUM(CALC_PRD_CM_PCT) AS CALC_PRD_CM_PCT_R13,
    SUM(CALC_STAFFING_INDEX) AS CALC_STAFFING_INDEX_R13,
    SUM(CALC_USI_PCT_OF_HOURS) AS CALC_USI_PCT_OF_HOURS_R13,
    NULL AS Admin_Labor_Cost_PR13,
    NULL AS Audit_Claims_Insurance_Costs_PR13,
    NULL AS Direct_Project_Costs_PR13,
    NULL AS FSS_Overhead_PR13,
    NULL AS Gross_Client_Margin_PR13,
    NULL AS Gross_Services_PR13,
    NULL AS Interest_Charge_PR13,
    NULL AS Net_Client_Margin_PR13,
    NULL AS Net_Revenues_PR13,
    NULL AS Net_Service_Revenues_PR13,
```

```
NULL AS Other_Revs_PR13,

NULL AS Partner_Labor_Cost_PR13,

NULL AS Partner_PRD_Labor_Costs_PR13,

NULL AS Partner_Services_Labor_Costs_PR13,

NULL AS PRD_Client_Margin_PR13,

NULL AS PRD_Expense_PR13,

NULL AS PRD_Hours_PR13,

NULL AS PRD_Labor_Cost_PR13,

NULL AS PRD_Labor_pls_Exp_PR13,

NULL AS Prov_for_Bad_Debt_PR13,

NULL AS R10_PRD_Labor_Costs_PR13,

NULL AS R10_Service_Labor_Costs_PR13,

NULL AS Service_Hours_PR13,

NULL AS Service_Labor_Cost_PR13,

NULL AS Shared_Overhead_PR13,

NULL AS TOTAL_HRS_PR13,

NULL AS Total_Overhead_PR13,

NULL AS TOTAL_PTR_HRS_PR13,

NULL AS Unbilled_Adjs_PR13,

NULL AS US_PRD_Labor_Costs_PR13,

NULL AS US_Service_Labor_Costs_PR13,

NULL AS CALC_DISC_PCT_PR13,

NULL AS CALC_GCM_PCT_PR13,
```

```
    NULL AS CALC_NCM_PCT_PR13,
    NULL AS CALC_NET_REV_PER_HOUR_PR13,
    NULL AS CALC_PARTNER_LEVERAGE_PR13,
    NULL AS CALC_PRD_CM_PCT_PR13,
    NULL AS CALC_STAFFING_INDEX_PR13,
    NULL AS CALC_USI_PCT_OF_HOURS_PR13
    FROM "_SYS_BIC"."project/CV_PROFIT"
    WHERE WBS='some existing WBS' and FISCAL_YEAR_PERIOD
between '2013009' and '2014008'
    GROUP BY WBS
    UNION ALL
    SELECT WBS,
    NULL AS Admin_Labor_Cost_R13,
    NULL AS Audit_Claims_Insurance_Costs_R13,
    NULL AS Direct_Project_Costs_R13,
    NULL AS FSS_Overhead_R13,
    NULL AS Gross_Client_Margin_R13,
    NULL AS Gross_Services_R13,
    NULL AS Interest_Charge_R13,
    NULL AS Net_Client_Margin_R13,
    NULL AS Net_Revenues_R13,
    NULL AS Net_Service_Revenues_R13,
    NULL AS Other_Revs_R13,
```

```
NULL AS Partner_Labor_Cost_R13,
NULL AS Partner_PRD_Labor_Costs_R13,
NULL AS Partner_Services_Labor_Costs_R13,
NULL AS PRD_Client_Margin_R13,
NULL AS PRD_Expense_R13,
NULL AS PRD_Hours_R13,
NULL AS PRD_Labor_Cost_R13,
NULL AS PRD_Labor_pls_Exp_R13,
NULL AS Prov_for_Bad_Debt_R13,
NULL AS R10_PRD_Labor_Costs_R13,
NULL AS R10_Service_Labor_Costs_R13,
NULL AS Service_Hours_R13,
NULL AS Service_Labor_Cost_R13,
NULL AS Shared_Overhead_R13,
NULL AS TOTAL_HRS_R13,
NULL AS Total_Overhead_R13,
NULL AS TOTAL_PTR_HRS_R13,
NULL AS Unbilled_Adjs_R13,
NULL AS US_PRD_Labor_Costs_R13,
NULL AS US_Service_Labor_Costs_R13,
NULL AS CALC_DISC_PCT_R13,
NULL AS CALC_GCM_PCT_R13,
NULL AS CALC_NCM_PCT_R13,
```

```
NULL AS CALC_NET_REV_PER_HOUR_R13,

NULL AS CALC_PARTNER_LEVERAGE_R13,

NULL AS CALC_PRD_CM_PCT_R13,

NULL AS CALC_STAFFING_INDEX_R13,

NULL AS CALC_USI_PCT_OF_HOURS_R13,

SUM(Admin_Labor_Cost),

SUM(Audit_Claims_Insurance_Costs),

SUM(Direct_Project_Costs),

SUM(FSS_Overhead),

SUM(Gross_Client_Margin),

SUM(Gross_Services),

SUM(Interest_Charge),

SUM(Net_Client_Margin),

SUM(Net_Revenues),

SUM(Net_Service_Revenues),

SUM(Other_Revs),

SUM(Partner_Labor_Cost),

SUM(Partner_PRD_Labor_Costs),

SUM(Partner_Services_Labor_Costs),

SUM(PRD_Client_Margin),

SUM(PRD_Expense),

SUM(PRD_Hours),

SUM(PRD_Labor_Cost),
```

```
SUM(PRD_Labor_pls_Exp),
SUM(Prov_for_Bad_Debt),
SUM(R10_PRD_Labor_Costs),
SUM(R10_Service_Labor_Costs),
SUM(Service_Hours),
SUM(Service_Labor_Cost),
SUM(Shared_Overhead),
SUM(TOTAL_HRS),
SUM(Total_Overhead),
SUM(TOTAL_PTR_HRS),
SUM(Unbilled_Adjs),
SUM(US_PRD_Labor_Costs),
SUM(US_Service_Labor_Costs),
SUM(CALC_DISC_PCT),
SUM(CALC_GCM_PCT),
SUM(CALC_NCM_PCT),
SUM(CALC_NET_REV_PER_HOUR),
SUM(CALC_PARTNER_LEVERAGE),
SUM(CALC_PRD_CM_PCT),
SUM(CALC_STAFFING_INDEX),
SUM(CALC_USI_PCT_OF_HOURS)
FROM "_SYS_BIC"."project/CV_PROFIT"
```

```
    WHERE WBS='some existing WBS' and FISCAL_YEAR_PERIOD
between '2012009' and '2013008'
    GROUP BY WBS
    )
    UNION ALL
    SELECT
    SUM(SERVICE_HOURS_R13),
    SUM(SERVICE_HOURS_PR13),
    SUM(SERVICE_HOURS_R13_GROWTH),
    SUM(SERVICE_HOURS_R13_PCT_GROWTH),
    SUM(TOTAL_HRS_R13),
    SUM(TOTAL_HRS_PR13),
    SUM(TOTAL_HRS_R13_GROWTH),
    SUM(TOTAL_HRS_R13_PCT_GROWTH),
    SUM(GROSS_SERVICES_R13),
    SUM(GROSS_SERVICES_PR13),
    SUM(GROSS_SERVICES_R13_GROWTH),
    SUM(GROSS_SERVICES_R13_PCT_GROWTH),
    SUM(UNBILLED_ADJS_R13),
    SUM(UNBILLED_ADJS_PR13),
    SUM(UNBILLED_ADJS_R13_GROWTH),
    SUM(UNBILLED_ADJS_R13_PCT_GROWTH),
    SUM(NET_SERVICE_REVENUES_R13),
```

```
SUM(NET_SERVICE_REVENUES_PR13),
SUM(NET_SERVICE_REVENUES_R13_GROWTH),
SUM(NET_SERVICE_REVENUES_R13_PCT_GROWTH),
SUM(OTHER_REVS_R13),
SUM(OTHER_REVS_PR13),
SUM(OTHER_REVS_R13_GROWTH),
SUM(OTHER_REVS_R13_PCT_GROWTH),
SUM(NET_REVENUES_R13),
SUM(NET_REVENUES_PR13),
SUM(NET_REVENUES_R13_GROWTH),
SUM(NET_REVENUES_R13_PCT_GROWTH),
SUM(US_SERVICE_LABOR_COSTS_R13),
SUM(US_SERVICE_LABOR_COSTS_PR13),
SUM(US_SERVICE_LABOR_COSTS_R13_GROWTH),
SUM(US_SERVICE_LABOR_COSTS_R13_PCT_GROWTH),
SUM(R10_SERVICE_LABOR_COSTS_R13),
SUM(R10_SERVICE_LABOR_COSTS_PR13),
SUM(R10_SERVICE_LABOR_COSTS_R13_GROWTH),
SUM(R10_SERVICE_LABOR_COSTS_R13_PCT_GROWTH),
SUM(SERVICE_LABOR_COST_R13),
SUM(SERVICE_LABOR_COST_PR13),
SUM(SERVICE_LABOR_COST_R13_GROWTH),
SUM(SERVICE_LABOR_COST_R13_PCT_GROWTH),
```

```
SUM(ADMIN_LABOR_COST_R13),

SUM(ADMIN_LABOR_COST_PR13),

SUM(ADMIN_LABOR_COST_R13_GROWTH),

SUM(ADMIN_LABOR_COST_R13_PCT_GROWTH),

SUM(PROV_FOR_BAD_DEBT_R13),

SUM(PROV_FOR_BAD_DEBT_PR13),

SUM(PROV_FOR_BAD_DEBT_R13_GROWTH),

SUM(PROV_FOR_BAD_DEBT_R13_PCT_GROWTH),

SUM(INTEREST_CHARGE_R13),

SUM(INTEREST_CHARGE_PR13),

SUM(INTEREST_CHARGE_R13_GROWTH),

SUM(INTEREST_CHARGE_R13_PCT_GROWTH),

SUM(DIRECT_PROJECT_COSTS_R13),

SUM(DIRECT_PROJECT_COSTS_PR13),

SUM(DIRECT_PROJECT_COSTS_R13_GROWTH),

SUM(DIRECT_PROJECT_COSTS_R13_PCT_GROWTH),

SUM(GROSS_CLIENT_MARGIN_R13),

SUM(GROSS_CLIENT_MARGIN_PR13),

SUM(GROSS_CLIENT_MARGIN_R13_GROWTH),

SUM(GROSS_CLIENT_MARGIN_R13_PCT_GROWTH),

SUM(PRD_HOURS_R13),

SUM(PRD_HOURS_PR13),

SUM(PRD_HOURS_R13_GROWTH),
```

```
SUM(PRD_HOURS_R13_PCT_GROWTH),

SUM(US_PRD_LABOR_COSTS_R13),

SUM(US_PRD_LABOR_COSTS_PR13),

SUM(US_PRD_LABOR_COSTS_R13_GROWTH),

SUM(US_PRD_LABOR_COSTS_R13_PCT_GROWTH),

SUM(R10_PRD_LABOR_COSTS_R13),

SUM(R10_PRD_LABOR_COSTS_PR13),

SUM(R10_PRD_LABOR_COSTS_R13_GROWTH),

SUM(R10_PRD_LABOR_COSTS_R13_PCT_GROWTH),

SUM(PRD_LABOR_COST_R13),

SUM(PRD_LABOR_COST_PR13),

SUM(PRD_LABOR_COST_R13_GROWTH),

SUM(PRD_LABOR_COST_R13_PCT_GROWTH),

SUM(PRD_EXPENSE_R13),

SUM(PRD_EXPENSE_PR13),

SUM(PRD_EXPENSE_R13_GROWTH),

SUM(PRD_EXPENSE_R13_PCT_GROWTH),

SUM(PRD_LABOR_PLS_EXP_R13),

SUM(PRD_LABOR_PLS_EXP_PR13),

SUM(PRD_LABOR_PLS_EXP_R13_GROWTH),

SUM(PRD_LABOR_PLS_EXP_R13_PCT_GROWTH),

SUM(PRD_CLIENT_MARGIN_R13),

SUM(PRD_CLIENT_MARGIN_PR13),
```

```
SUM(PRD_CLIENT_MARGIN_R13_GROWTH),

SUM(PRD_CLIENT_MARGIN_R13_PCT_GROWTH),

SUM(FSS_OVERHEAD_R13),

SUM(FSS_OVERHEAD_PR13),

SUM(FSS_OVERHEAD_R13_GROWTH),

SUM(FSS_OVERHEAD_R13_PCT_GROWTH),

SUM(SHARED_OVERHEAD_R13),

SUM(SHARED_OVERHEAD_PR13),

SUM(SHARED_OVERHEAD_R13_GROWTH),

SUM(SHARED_OVERHEAD_R13_PCT_GROWTH),

SUM(Audit_Claims_Insurance_Costs_R13),

SUM(Audit_Claims_Insurance_Costs_PR13),

SUM(AUDIT_CLAIMS_INSURANCE_R13_GROWTH),

SUM(AUDIT_CLAIMS_INSURANCE_R13_PCT_GROWTH),

SUM(TOTAL_OVERHEAD_R13),

SUM(TOTAL_OVERHEAD_PR13),

SUM(TOTAL_OVERHEAD_R13_GROWTH),

SUM(TOTAL_OVERHEAD_R13_PCT_GROWTH),

SUM(NET_CLIENT_MARGIN_R13),

SUM(NET_CLIENT_MARGIN_PR13),

SUM(NET_CLIENT_MARGIN_R13_GROWTH),

SUM(NET_CLIENT_MARGIN_R13_PCT_GROWTH),

SUM(PARTNER_SERVICES_LABOR_COSTS_R13),
```

```
SUM(PARTNER_SERVICES_LABOR_COSTS_PR13),
SUM(PARTNER_SERVICES_LABOR_COSTS_R13_GROWTH),
SUM(PARTNER_SERVICES_LABOR_COSTS_R13_PCT_GROWTH),
SUM(PARTNER_PRD_LABOR_COSTS_R13),
SUM(PARTNER_PRD_LABOR_COSTS_PR13),
SUM(PARTNER_PRD_LABOR_COSTS_R13_GROWTH),
SUM(PARTNER_PRD_LABOR_COSTS_R13_PCT_GROWTH),
SUM(PARTNER_LABOR_COST_R13),
SUM(PARTNER_LABOR_COST_PR13),
SUM(PARTNER_LABOR_COST_R13_GROWTH),
SUM(PARTNER_LABOR_COST_R13_PCT_GROWTH),
SUM(TOTAL_PTR_HRS_R13),
SUM(TOTAL_PTR_HRS_PR13),
SUM(TOTAL_PTR_HRS_R13_GROWTH),
SUM(TOTAL_PTR_HRS_R13_PCT_GROWTH),
SUM(CALC_DISC_PCT_R13),
SUM(CALC_DISC_PCT_PR13),
SUM(CALC_DISC_PCT_R13_GROWTH),
SUM(CALC_GCM_PCT_R13),
SUM(CALC_GCM_PCT_PR13),
SUM(CALC_GCM_PCT_R13_GROWTH),
SUM(CALC_PRD_CM_PCT_R13),
SUM(CALC_PRD_CM_PCT_PR13),
```

```
    SUM(CALC_PRD_CM_PCT_R13_GROWTH),
    SUM(CALC_NCM_PCT_R13),
    SUM(CALC_NCM_PCT_PR13),
    SUM(CALC_NCM_PCT_R13_GROWTH),
    SUM(CALC_NET_REV_PER_HOUR_R13),
    SUM(CALC_NET_REV_PER_HOUR_PR13),
    SUM(CALC_NET_REV_PER_HOUR_R13_GROWTH),
    SUM(CALC_PARTNER_LEVERAGE_R13),
    SUM(CALC_PARTNER_LEVERAGE_PR13),
    SUM(CALC_PARTNER_LEVERAGE_R13_GROWTH),
    SUM(CALC_STAFFING_INDEX_R13),
    SUM(CALC_STAFFING_INDEX_PR13),
    SUM(CALC_STAFFING_INDEX_R13_GROWTH),
    SUM(CALC_USI_PCT_OF_HOURS_R13),
    SUM(CALC_USI_PCT_OF_HOURS_PR13),
    SUM(CALC_USI_PCT_OF_HOURS_R13_GROWTH)
    FROM "_SYS_BIC"."project/CV_PROFIT_R13"
    WHERE WBS='some existing WBS' AND FISCAL_YEAR='2014' AND
FISCAL_PERIOD='008'
```