

|  |
| --- |
| Data Warehouse  Издательский дом «МИФ» |
|  |

| Related Artifacts | |
| --- | --- |
| Ref. | Name |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Abbreviations and Acronyms | |
|  |  |
|  |  |

Contents

[1 Business Description 5](#_Toc500068480)

[1.1 Business background 5](#_Toc500068481)

[1.2 Problems because of poor data management 5](#_Toc500068482)

[1.3 Benefits from implementing a Data Warehouse 5](#_Toc500068483)

[2 Dimensions of a Business 6](#_Toc500068484)

[2.1 Business process 6](#_Toc500068485)

[2.2 The grain 6](#_Toc500068486)

[2.3 Dimensions 6](#_Toc500068487)

[2.3.1 Dim\_Time\_Day 6](#_Toc500068488)

[2.3.2 Dim\_Customers 7](#_Toc500068489)

[2.3.3 Dim\_Employees 8](#_Toc500068490)

[2.3.4 Dim\_Stores\_SCD 8](#_Toc500068491)

[2.3.5 Dim\_Products 9](#_Toc500068492)

[2.3.6 Dim\_Payments 10](#_Toc500068493)

[2.4 Fact table 10](#_Toc500068494)

[3 Logical Scheme 11](#_Toc500068495)

[3.1 Star scheme 11](#_Toc500068496)

[3.2 3NF scheme 12](#_Toc500068497)

[3.2.1 Employees 13](#_Toc500068498)

[3.2.2 Products 13](#_Toc500068499)

[3.2.3 Geodata 13](#_Toc500068500)

[3.2.4 Payments 14](#_Toc500068501)

[3.2.5 Customers 14](#_Toc500068502)

[3.2.6 Shops 14](#_Toc500068503)

[4 Data Flow 15](#_Toc500068504)

[4.1 SA\_SRC 16](#_Toc500068505)

[4.2 Staging layer 16](#_Toc500068506)

[4.2.1 BL\_WRK 16](#_Toc500068507)

[4.2.2 BL\_CL 16](#_Toc500068508)

[4.2.3 BL\_3NF 16](#_Toc500068509)

[4.3 Aggregation layer 16](#_Toc500068510)

[4.3.1 BL\_CL\_DM 16](#_Toc500068511)

[4.3.2 BL\_DM 16](#_Toc500068512)

[5 Source description 17](#_Toc500068513)

[5.1 Geodata 17](#_Toc500068514)

[5.2 Customers 17](#_Toc500068515)

[5.3 Employees 18](#_Toc500068516)

[5.4 Departments 18](#_Toc500068517)

[5.5 Stores 19](#_Toc500068518)

[5.6 Products 20](#_Toc500068519)

[5.7 Payments 20](#_Toc500068520)

[5.8 Categories 20](#_Toc500068521)

[6 Fact table 21](#_Toc500068522)

[6.1 Randomize\_max func 21](#_Toc500068523)

[6.2 Randomize func 22](#_Toc500068524)

[6.3 Virtual columns 23](#_Toc500068525)

[7 CLS-tables 23](#_Toc500068526)

[7.1 Geodata 23](#_Toc500068527)

[7.2 Customers 26](#_Toc500068528)

[7.3 Employees 26](#_Toc500068529)

[7.4 Payments 27](#_Toc500068530)

[7.5 Catalog 27](#_Toc500068531)

[7.6 Adresses 28](#_Toc500068532)

[7.7 Stores\_SCD 28](#_Toc500068533)

[8 3NF-tables 29](#_Toc500068534)

[9 CL\_DM-TABLES 30](#_Toc500068535)

[10 DM-tables 30](#_Toc500068536)

[10.1 Dim\_Time\_day 30](#_Toc500068537)

[10.2 Итоговые скриншоты 31](#_Toc500068538)

[10.2.1 Dim\_Time\_Day 31](#_Toc500068539)

[10.2.2 Dim\_Customers 32](#_Toc500068540)

[10.2.3 Dim\_Employees 33](#_Toc500068541)

[10.2.4 Dim\_Stroes\_SCD 34](#_Toc500068542)

[10.2.5 Dim\_Products 35](#_Toc500068543)

[10.2.6 Dim\_Payments 36](#_Toc500068544)

[10.2.7 Fct\_sales 36](#_Toc500068545)

[11 Fact Table Partitioning Strategy 38](#_Toc500068546)

[12 Strategy of Parallel Load 39](#_Toc500068547)

[13 Business analysis 40](#_Toc500068548)

# Business Description

## Business background

«Манн, Иванов и Фербер» — московское издательство деловой литературы, начавшее свою деятельность в [2005 году](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4). Основатели: [Игорь Манн](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B0%D0%BD%D0%BD,_%D0%98%D0%B3%D0%BE%D1%80%D1%8C_%D0%91%D0%BE%D1%80%D0%B8%D1%81%D0%BE%D0%B2%D0%B8%D1%87&action=edit&redlink=1" \o "Манн, Игорь Борисович (страница отсутствует)), Михаил Иванов и Михаил Фербер (по первым буквам фамилий основателей и произошло название). Издательство специализируется на издании книг по [маркетингу](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%B5%D1%82%D0%B8%D0%BD%D0%B3) и [менеджменту](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%BC%D0%B5%D0%BD%D1%82), [управлению временем](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B5%D0%BC), адресованных собственникам бизнеса, менеджерам, консультантам, студентам и слушателям программы [MBA](https://ru.wikipedia.org/wiki/MBA). Кроме того, развиваются направления книг о спорте, психологии и кулинарии.

На сегодняшний день издательство «МИФ» является одним из самых прогрессивных на рынке СНГ: ведет активную маркетинговую политику, внедряет передовые технологии и использует нестандартный подход в решении бизнес-задач.

## Problems because of poor data management

Неправильное хранение и управление данными может привести к большим потерям ресурсов компании: времени, клиентов, партнеров – все это ведет к значительным денежным убыткам.

Список проблем, которые возникли у заказчика:

* Потеря клиентов, в связи с устаревшей или некорректной информацией.
* Провальные рекламные кампании.
* Убытки, вызванные решениям, принятыми на основе неверных данных.
* Упущенная прибыль: неосуществленные возможности получения прибыли в связи с неудачным выбором способа действий.

## Benefits from implementing a Data Warehouse

* **Историчность**: позволяет хранить данные за все время операционной деятельности компании, что позволяет анализировать данные за различные промежутки времени, отслеживать тренды, а также делать прогнозы о будущем развитии, нуждах и возможных проблемах.
* **Консистенность данных**: все данные хранятся в едином формате, что позволяет сделать анализ максимально точным.
* **Принятие решений**: позволяет максимально быстро получить доступ к необходимым данным и на их основе принять взвешенное решение.
* **Эффективность работы:** увеличивает эффективность бизнес-процессов организации.

# Dimensions of a Business

## Business process

Бизнес-процессом для издательского дома «МИФ» является учет продаж изданных книг.

## The grain

Одна строка из таблицы фактов представляет собой одну транзакцию в чеке: определенную книгу, которую работник продал потребителю в конкретном магазине в какой-либо день со скидкой (или без), а также каким способом осуществлялась оплата.

## Dimensions

Хранилище данных будет включать следующие измерения:

* Даты (Dim\_Time\_Day)
* Клиенты (Dim\_Customers)
* Работники (Dim\_Employees\_SCD)
* Магазины (Dim\_Stores)
* Продукты (Dim\_Products)
* Оплата (Dim\_Payments)

Перед описанием каждого измерения будут указаны бизнес-правила, которые были применены (если таие имеются).

### Dim\_Time\_Day

* В данном измерении были использованы три индикатора (флага):
  + Weekend Indicator
  + Last Day Of Week Indicator
  + Last Day Of Month Indicator

Они могут принимать только два значения: true/false. Однако, были выбраны «говорящие» значения (напр. Weekend/weekday, а не Y/N), так как в дальнейшем такие значения будут удобнее для пользователей в процессе формирования отчетов, применения фильтров итд.

| Dim\_Time\_Day | | | |
| --- | --- | --- | --- |
| # | Name | Type | Description |
| 1 | Full Date DT | DATE | Primary Key: 12-NOV-2017 |
| 2 | Full Date Description | VARCHAR2(20) | 12 November 2017 |
| 3 | Day Of Week | NUMBER(1) | День недели (1-7) |
| 4 | Day Of Month | NUMBER(2) | День месяца (1-31) |
| 5 | Day Of Year | NUMBER(3) | День в году (1-366) |
| 6 | Day Name | VARCHAR2(10) | Monday-Sunday |
| 7 | Weekend Indicator | VARCHAR2(40) | Weekend/Weekday |
| 8 | Last Day Of Week Indicator | VARCHAR2(40) | Last day of the week/Not last day of the week |
| 9 | Last Day Of Month Indicator | VARCHAR2(40) | Last day of the month/Not last day of the month |
| 10 | Week Of Month | NUMBER(1) | Номер недели в месяце (1-5) |
| 11 | Week Of Year | NUMBER(2) | Номер недели в году (1-52) |
| 12 | Week Ending DT | DATE | Дата конца недели (воскресенье) |
| 13 | Month Number | NUMBER(2) | Номер месяца в году (1-12) |
| 14 | Month Name | VARCHAR2(30) | Название месяца |
| 15 | Month Year | VARCHAR2(30) | Формат: November-2017 |
| 16 | Quarter Number | NUMBER(3) | Номер квартала (1-4) |
| 17 | Quarter Year | VARCHAR2(30) | Формат: 4-2017 |
| 18 | Year Number | NUMBER(4) | Год |

### Dim\_Customers

* Так как анализ клиента в геогрфическом разрезе очень важен для заказчика, все суррогатные ключи на географические данные были сохранены.

|  |  |  |  |
| --- | --- | --- | --- |
| Dim\_Customers | | | |
| # | Name | Type | Description |
| 1 | Customer ID | NUMBER(8) | Primary Key (уникален для каждой записи) |
| 2 | Customer code | VARCHAR2(15) | Natural Key (может повторяться) |
| 3 | Customer Name | VARCHAR2(35) | Имя клиента |
| 4 | Customer Surname | VARCHAR2(35) | Фамилия клиента |
| 5 | Customer Email | VARCHAR2(65) | Емейл клиента |
| 6 | Customer Phone | VARCHAR2(30) | Номер мобильного телефона |
| 7 | Customer Card | VARCHAR2(20) | Номер карты: 4929 7798 1456 9983 |
| 8 | Customer Region ID | NUMBER(8) | PK региона |
| 9 | Customer Region | VARCHAR2(50) | Название региона |
| 10 | Customer Country ID | NUMBER(8) | PK страны |
| 11 | Customer Country | VARCHAR2(50) | Название страны |
| 12 | Customer City ID | NUMBER(8) | PK города |
| 13 | Customer City | VARCHAR2(50) | Название города |
| 17 | Insert\_DT | DATE | Дата добавления |
| 18 | Update\_DT | DATE | Дата измений аттрибута(ов) |

### Dim\_Employees

* При заполнении данных будут добавлены значения по умолчанию для N/A (not applicable) и N/D (not defined).

|  |  |  |  |
| --- | --- | --- | --- |
| Dim\_Employees | | | |
| # | Name | Type | Decription |
| 1 | Employee ID | NUMBER(8) | Primary Key (уникален для каждой записи) |
| 2 | Employee code | VARCHAR2(15) | Natural Key (может повторяться) |
| 3 | Employee Name | VARCHAR2(35) | Имя работника |
| 4 | Employee Surname | VARCHAR2(35) | Фамилия работника |
| 5 | Employee Email | VARCHAR2(65) | Емейл работника |
| 6 | Employee Phone | VARCHAR2(30) | Номер мобильного телефона |
| 7 | Department Name | VARCHAR2(30) | Название отдела |
| 8 | Manager ID | NUMBER(8) | РК менеджера (Employee Key) |
| 9 | Manager Name | VARCHAR2(70) | Имя и фамилия менеджера |
| 10 |  |  |  |
| 11 | Store Name | VARCHAR2(35) | Название магазина |
| 12 | Store City | VARCHAR2(50) | Город, где находится магазин |
| 13 | Store Address | VARCHAR2(50) | Адрес, где находится магазин |
| 14 | Start DT | DATE | Дата, когда состояние вступило в силу |
| 15 | End DT | DATE | Дата, когда состояние изменилось |
| 16 | IsActive | VARCHAR2(10) | Y (действителен)/N (не действителен) |
| 17 | Insert\_DT | DATE | Дата добавления |

### Dim\_Stores\_SCD

* Это второй тип SCD, те здесь добавлена возможность точного отслеживания всех изменений.
* Если IsActive=Y, то end\_dt будет 31-DEC-9999’
* При заполнении данных будут добавлены значения по умолчанию для N/A (not applicable)и N/D (not defined).

|  |  |  |  |
| --- | --- | --- | --- |
| Dim\_Stores | | | |
| # | Name | Type | Decription |
| 1 | Store Key | NUMBER(8) | Primary Key (уникален для каждой записи) |
| 2 | Store ID | VARCHAR2(15) | Natural Key (может повторяться) |
| 3 | Store Name | VARCHAR2(35) | Название магазина |
| 4 | Store Email | VARCHAR2(65) | Емейл магазина |
| 5 | Store Phone | VARCHAR2(30) | Номер магазина |
| 8 | Manager Key | NUMBER(8) | РК менеджера (Employee Key) |
| 6 | Manager Name | VARCHAR2(70) | Имя и фамилия менеджера |
| 7 | Store Region ID | NUMBER(8) | PK региона |
| 8 | Store Region | VARCHAR2(50) | Название региона |
| 9 | Store Country ID | NUMBER(8) | PK страны |
| 10 | Store Country | VARCHAR2(50) | Название страны |
| 11 | Store City ID | NUMBER(8) | PK города |
| 12 | Store City | VARCHAR2(50) | Название города |
| 13 | Store Address ID | NUMBER(8) | PK адреса |
| 14 | Store Address | VARCHAR2(50) | Адрес |
| 15 | Start DT | DATE | Дата, когда состояние вступило в силу |
| 16 | End DT | DATE | Дата, когда состояние изменилось |
| 17 | IsActive | VARCHAR2(10) | Y (действителен)/N (не действителен) |
| 18 | Insert\_DT | DATE | Дата добавления |

### Dim\_Products

При заполнении данных будут добавлены значения по умолчанию для N/A (not applicable)и N/D (not defined)

|  |  |  |  |
| --- | --- | --- | --- |
| Dim\_Products | | | |
| # | Name | Type | Decription |
| 1 | Product ID | NUMBER(8) | Primary Key (уникален для каждой записи) |
| 2 | Product code | VARCHAR2(15) | Natural Key (может повторяться) |
| 4 | Category Name | VARCHAR2(25) | Printed book, audiobook, e-book. |
| 5 | Category Description | VARCHAR2(150) | Описание категории |
| 6 | Product Name | VARCHAR2(50) | Название книги |
| 8 | Product Description | VARCHAR2(150) | Описание книги |
| 9 | Product Genre | VARCHAR2(50) | Жанр |
| 10 | Product Author | VARCHAR2(50) | Автор |
| 11 | Product Weight (kg) | NUMBER(3,3) | Вес книги: только для печатных книг |
| 12 | Insert\_DT | DATE | Дата добавления |
| 13 | Update\_DT | DATE | Дата измений аттрибута(ов) |

### Dim\_Payments

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Type | Decription |
| 1 | Payment ID | NUMBER(8) | Primary Key (уникален для каждой записи) |
| 2 | Payment Type Group | VARCHAR2(25) | * Наличные * Кредитная карта, * Электронная валюта |
| 3 | Payment Type Description | VARCHAR2(25) | Наличные:   * BYN * USD   Кредитные карты:   * Visa, * MasterCard   Электронная валюта:   * Bitcoin * Etherum |
| 4 | Prepayment | VARCHAR2(25) | Предоплата:   * Prepayment * non-prepayment |
| 5 | Insert\_DT | DATE | Дата добавления |
| 6 | Update\_DT | DATE | Дата измений аттрибута(ов) |

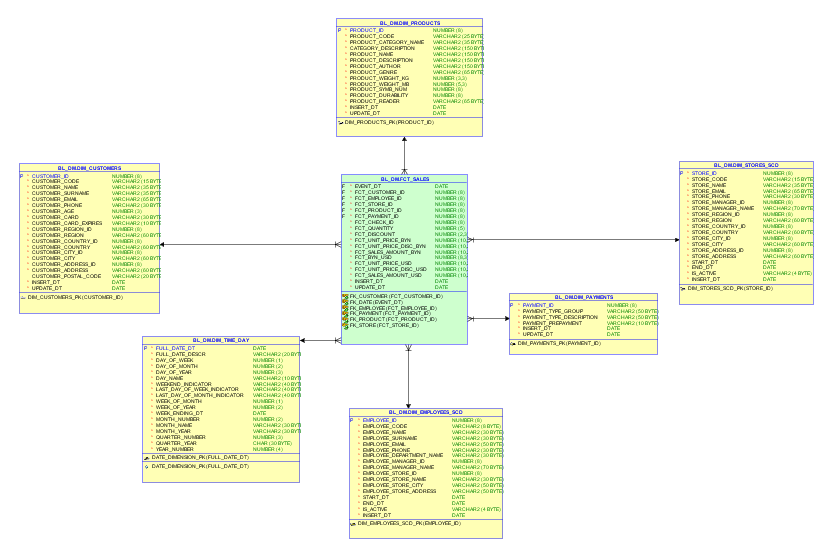
## Fact table

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Type | Decription |
| 1 | Date ID | NUMBER(8) | FK на измерение |
| 2 | Customer ID | NUMBER(8) | FK на измерение |
| 3 | Employee ID | NUMBER(8) | FK на измерение |
| 4 | Store ID | NUMBER(8) | FK на измерение |
| 5 | District ID | NUMBER(8) | FK на измерение |
| 6 | Product ID | NUMBER(8) | FK на измерение |
| 8 | Discount ID | NUMBER(8) | FK на измерение |
| 9 | Payment ID | NUMBER(8) | FK на измерение |
| 10 | Check ID | NUMBER(8) | Дегенеративное измерение |
| 11 | Quantity | NUMBER(5) | Кол-во |
| 12 | Unit Price BYN | NUMBER(10,2) | Цена за штуку |
| 13 | Unit Price With Discount BYN | NUMBER(10,2) | Цена за штуку со скидкой |
| 14 | Sales Amount BYN | NUMBER(10,2) | Кол-во\*Цену за штуку |
| 15 | BYN\_USD | NUMBER(8,3) | Курс доллара к белорусскому рублю |
| 16 | Unit Price USD | NUMBER(10,2) | Цена за штуку в долларах |
| 17 | Unit Price With Discount USD | NUMBER(10,2) | Цена за штуку со скидкой в долларах |
| 18 | Sales Amount USD | NUMBER(10,2 ) | Кол-во\*Цену за штуку в долларах |
| 19 | Insert\_DT | DATE | Дата добавления |

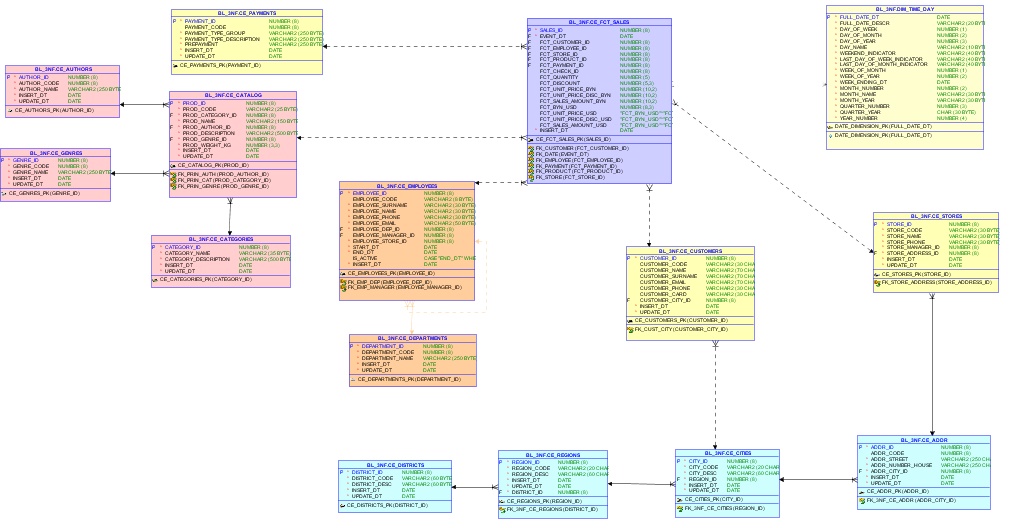
# Logical Scheme

## Star scheme

Была выбрана схема Звезда, так как:

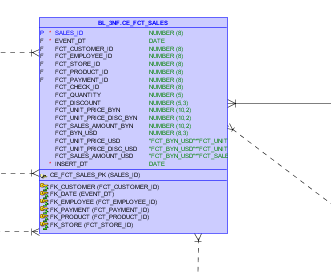
* Увеличивается производительность: так как все данные денормализованы, то уменьшается количество операций объединения таблиц (join) в запросах.
* Относительно небольшое количество данных позволяет хранить все в денормализованном виде.
* Быстрее осуществляется загрузка данных в таблицы: так как измерения и фактовые таблицы разделены, можно часто добавлять данные в фактовую таблицу, при этом записи в измерениях добавлять/изменять лишь при необходимости.  
    
  

## 3NF scheme

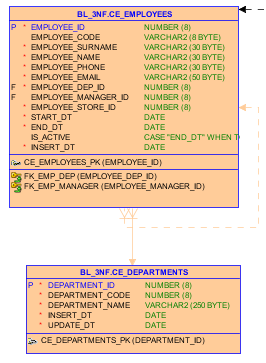


Одинаковым цветом обозначены сущности из одной логической единицы.

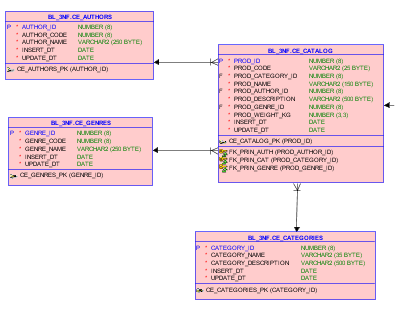
Желтым цветом обозначены логические единицы, которые сами по себе.



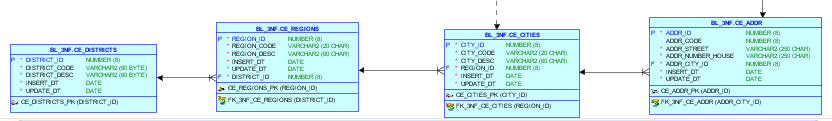
### Employees



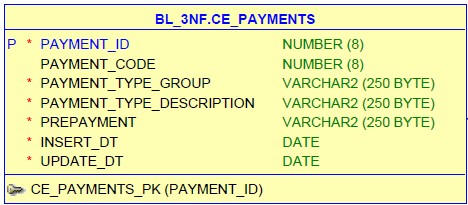
### Products



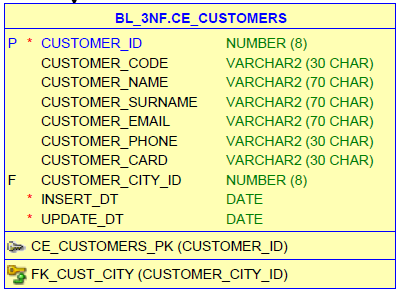
### Geodata



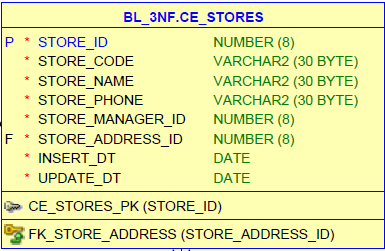
### Payments



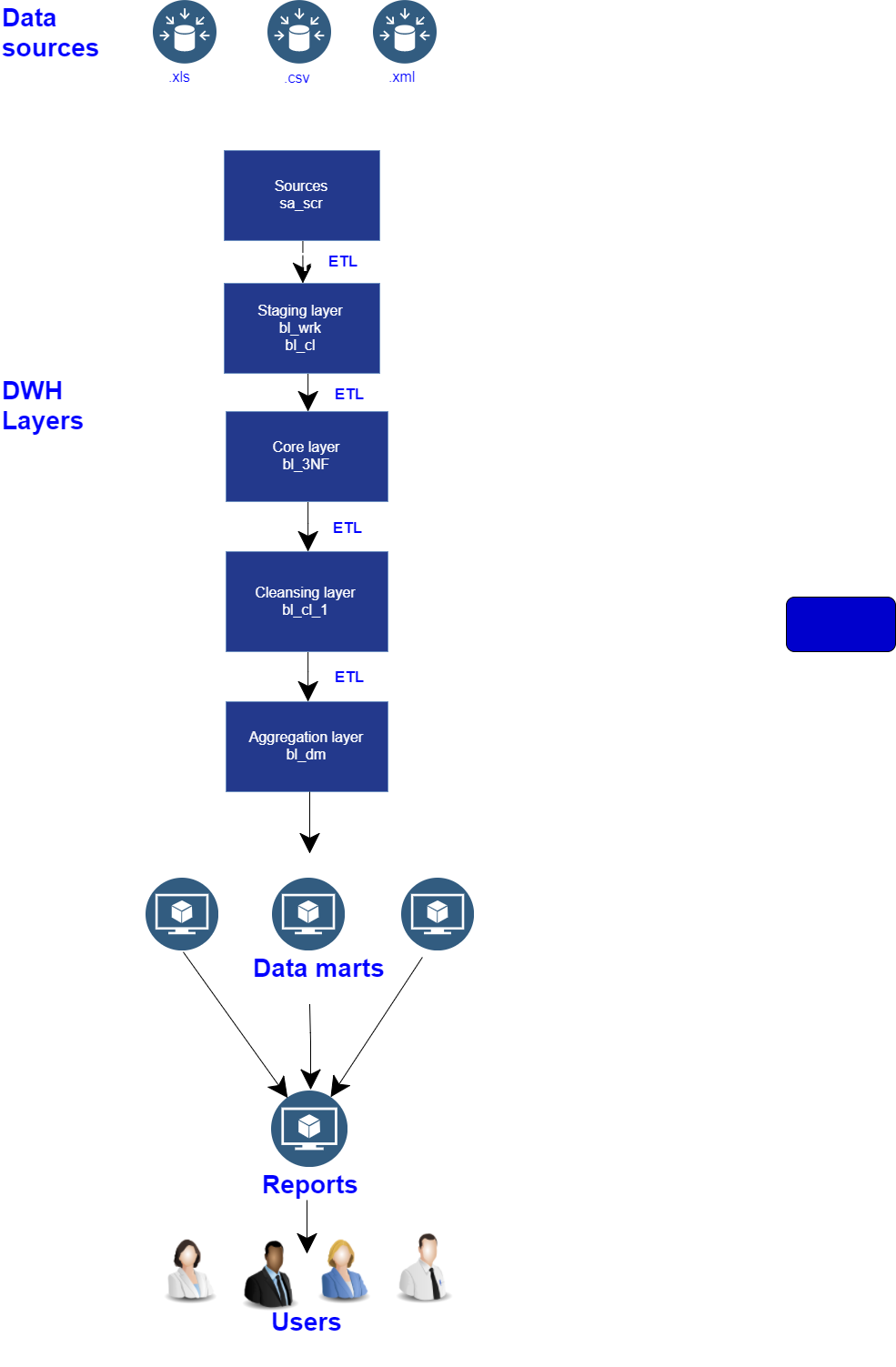
### Customers



### Shops



# Data Flow



## SA\_SRC

На этом этапе все источники сохраняются в виде внешних таблиц (таблицы, доступныу только для чтения, метаданные которых хранятся в базе данных, а данные – вне базы данных) добавляются в базу данных.

## Staging layer

### BL\_WRK

На этом этапе все данные из внешних таблиц загружаются в таблицы бд: это делается для увеличения производительности, так как считывание из бд быстрее, чем считывание с диска.

Таким образом, структуры таблиц sa\_src и bl\_wrk совпадают.

### BL\_CL

На этом этапе все данные очищаются,фильтруются. Именно на этом этапе осуществляется маппинг данных из разных таблиц.

Структура bl\_cl таблиц такая же как у bl\_3NF.

### BL\_3NF

На этом этапе уже полностью очищенные и подготовленные данные раскидываются по таблицам в третьей нормальной форме для их дальнейшей загрузки в DWH. Здесь уже появляются все необходимые ограничение (PK, FK) для поддержания ссылочной целостности.

## Aggregation layer

### BL\_CL\_DM

На этом этапе происходят все необходимые джоины, маппинг всех FK на значения из DWH

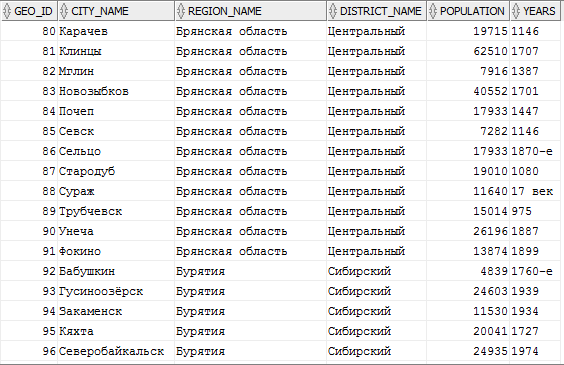
### BL\_DM

Непосредственно наше хранилище данных.

# Source description

## Geodata

С интернета был скачен excel-файл с форматом данных:



## Customers

Данные генерировались на сайте <http://www.fakenamegenerator.com> на русском языке.

Было сгенирировано три архива по 150000 записей.

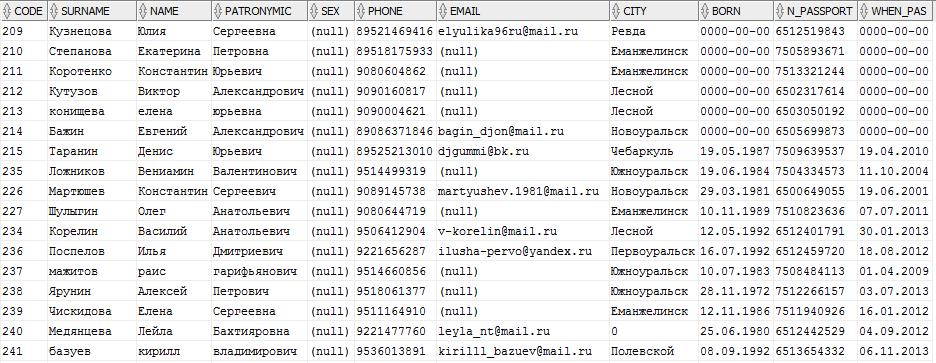
Города я добавляла вручную из геодаты:

* В первом файле перечислины все города – 50000 записей.
* Во втором только Москва – 50000 записей.
* В третьем Санкт-Петербург – 50000 записей.

(два самых крупных города России соответсвенно).

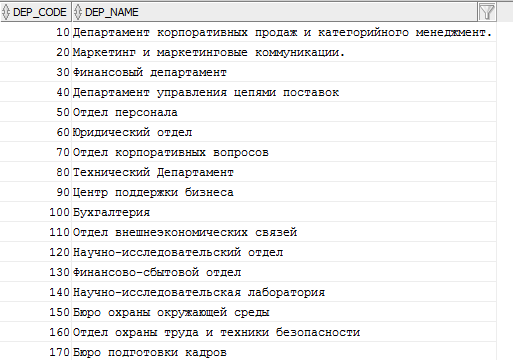
## Employees

Архив был скачен с сайта <https://ebaza.pro/category/7>.



## Departments

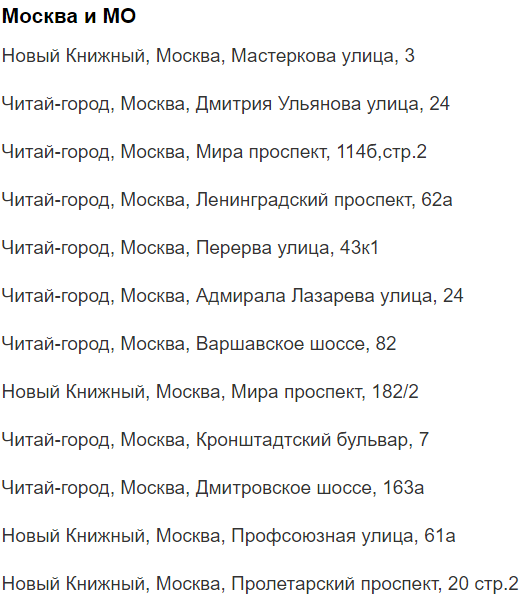
Документ составлялся вручную.



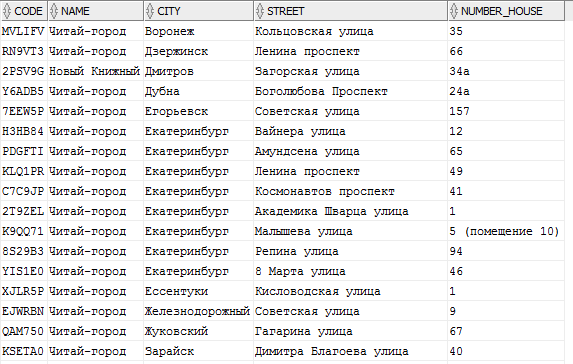
## Stores

Данные были взяты с официального сайта издательского дома «МИФ».

<https://www.mann-ivanov-ferber.ru/wtb/>

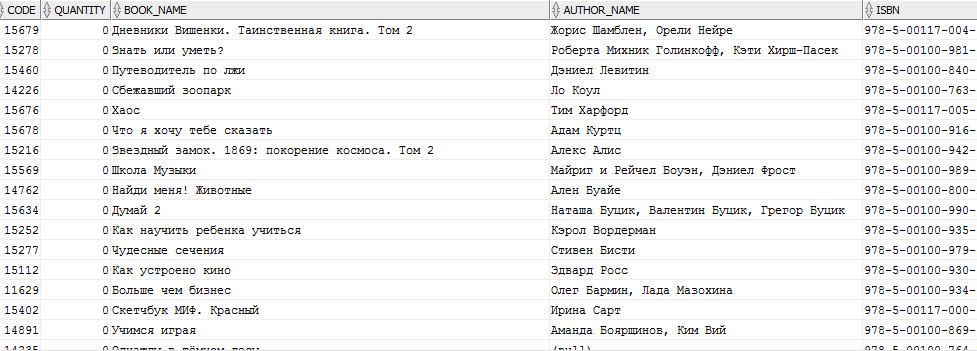


Код я предварительно сгенерировала с помощью sql-запроса и вставила в документ-источник.



## Products

Прайс-лист мне выслал технический директор издательского дома «МИФ» после того, как я обратилась с такой просьбой.



Также был получен json-файл от представителя технической поддержки сайта https://www.samolit.com , но не хватило времени для его загрузки.

## Payments

Файл был сформирован вручную.



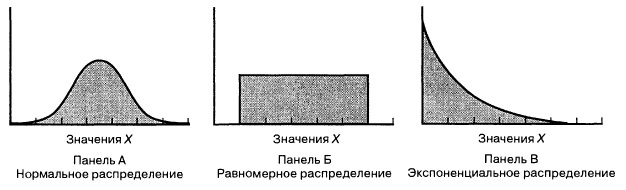
Практически это junk-dimension, так как данные не будут разбиваться на 3нф, а будут храниться вместе (cross join) ввиду их незначительности по отдельности.

## Categories

Файл также формировался вручную

# Fact table

Стояла цель сгенерировать значения, которые бы распределялись нормально, а не равномерно:



Так как функцию dbms\_random.normal() очень тяжело контролировать и нельзя передать промежуток, в котором должны находиться числа, я написала две функции, где сымитировала нормальное распределение.

## Randomize\_max func

Эта функция для тех значений, промежуток которых может изменяться и заранее неизвестен (id магазинов, клиентов, работников итд).

FUNCTION randomize\_max(

table\_name IN VARCHAR2,

col\_name IN VARCHAR2)

RETURN NUMBER;

В функцию в качестве входных параметров передаются имя таблицы и значение поля, для которых генерируются значения:

randomize\_max('bl\_3nf.ce\_customers', 'customer\_id')

randomize\_max('bl\_3nf.ce\_employees', 'employee\_id')

Затем с помощью EXECUTE IMMEDIATE находится максимальное значение в колонке ( то есть значение, до которого надо генерировать).

EXECUTE IMMEDIATE 'SELECT MAX('||col\_name||') FROM '||table\_name INTO v\_max;

Сама генерация:

Сначала генерируется число от 1 до 10, затем оно проверяется, и, в зависимости от значения, генерируется id которое нам нужно.

i:=dbms\_random.value(1,10);

result:=

CASE

WHEN i<=7 THEN

CEIL(dbms\_random.value(1,(v\_max\*0.45))) --45%

WHEN i>7 AND i<=9 THEN

CEIL(dbms\_random.value((v\_max\*0.45),(v\_max\*0.97))) --52%

WHEN i<10 THEN

CEIL(dbms\_random.value((v\_max\*0.97),(v\_max\*1))) --3%

ELSE

'1'

END ;

Первый промежуток взят от 1, так как преполагается, что 3НФ таблица заполняется равномерно (нет дырок в id) и нумерация начинается с 1.

Также использовалась функция CEIL, чтобы избежать значения 0.

## Randomize func

В жту функцию можно самому передавать значения, в пределах которых будут генерироваться промежутки:

FUNCTION randomize(

a IN NUMBER,

b IN NUMBER,

c IN NUMBER,

d IN NUMBER,

e IN NUMBER)

RETURN NUMBER

Вызов функции:

randomize(1,4,10,20,50) as quantity

Принцип такой же, как в предыдущей функции:

Больше всего значений в таблице будет в первом промежутку, чуть меньше во втором, меньше всего в четвертом промежутке.

i:=dbms\_random.value(1,20);

result:=

CASE

WHEN i<=10 THEN

CEIL(dbms\_random.value(a,b))

WHEN i>10 AND i<=16 THEN

CEIL(dbms\_random.value(b,c))

WHEN i>16 AND i<=19 THEN

CEIL(dbms\_random.value(c,d))

WHEN i<20 THEN

CEIL(dbms\_random.value(d,e))

ELSE

'1'

END ;

Также использовалась функция CEIL, чтобы избежать значения 0.

## Virtual columns

Для колонок, которые вычисляются на основе других колонок, я использовала virtual columns, то есть они будут вычислятся автоматически и нет необходимости передавать их значения в таблицу.

Fct\_unit\_price\_disc\_BYN generated always

AS

(Fct\_unit\_price\_BYN\*Fct\_discount), Fct\_sales\_Amount\_BYN generated always

AS

(Fct\_unit\_price\_BYN\*Fct\_discount \*Fct\_quantity ) , Fct\_BYN\_USD NUMBER(8,3) , Fct\_unit\_price\_USD generated always

AS

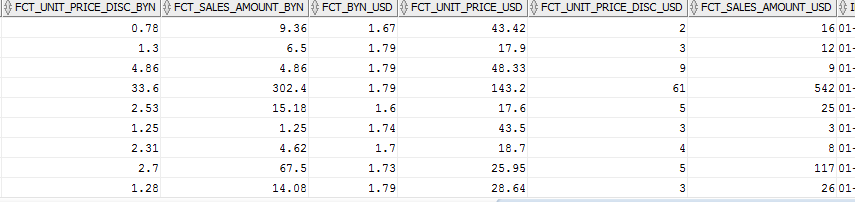
(Fct\_BYN\_USD \* Fct\_unit\_price\_BYN), Fct\_unit\_price\_disc\_USD generated always

AS

(CEIL(Fct\_BYN\_USD \* Fct\_unit\_price\_BYN\*Fct\_discount)) , Fct\_sales\_Amount\_USD generated always

AS

(CEIL(Fct\_BYN\_USD \* Fct\_unit\_price\_BYN\*Fct\_discount \*Fct\_quantity ))



# CLS-tables

В данном разделе будут описаны основные пробелы, с которыми я столкнулась в процессе переноса данных, их решения, а также фичи, которые были внедрены.

Все сls-таблицы перед заполнением очищались с помощью операции truncate.

Именно здесь осуществлялся join с родительскими таблицами из 3НФ.

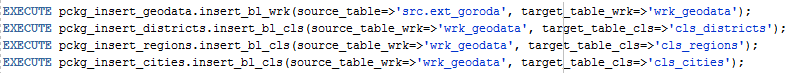
## Geodata

Трудности:

1. Так как данные пришли в денормализованном виде, то необходимо было сначала разбить их на города->регионы->округа.
2. Отсутствовали бизнес-ключи

Решения:

1. Для нормализации данных сначала внешняя таблица полностью выгружалась в wrk, а затем в cls-таблицы раскидывалась по нужной иерархии:



1. Так как в источнике данных не было никаких уникальных кодов(бизнес-ключей), чтобы однозначно идентифицировать одну запись, я их генерировала в sql с помощью case:

* Округа:

CASE

WHEN (INSTR(district\_name,'-', 1, 1)>0 )

THEN SUBSTR(district\_name,1,3)

||SUBSTR(district\_name,(INSTR(district\_name,'-', 1, 1)+1),3)

WHEN LENGTH(district\_name)>13

THEN SUBSTR(district\_name,1,3)

||SUBSTR(district\_name,7,3)

WHEN district\_name='Сибирский'

THEN 'Сиб'

WHEN district\_name='Приволжский'

THEN 'Привлж'

WHEN district\_name='Уральский'

THEN 'Урал'

ELSE SUBSTR(district\_name,1,3)

END AS code

* Регионы

CASE

WHEN (INSTR(region\_name,' ', 1, 1)>0 )

THEN SUBSTR(region\_name,1,4)||SUBSTR(region\_name,7,1)

||SUBSTR(region\_name,(INSTR(region\_name,' ', 1, 1)+1),2)

WHEN (INSTR(region\_name,'-', 1, 1)>0 )

THEN SUBSTR(region\_name,1,3)

||SUBSTR(region\_name,(INSTR(region\_name,'-', 1, 1)+1),3)

ELSE SUBSTR(region\_name,1,3)

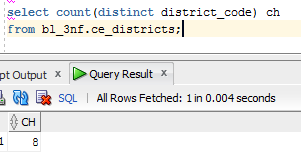
END AS region

* Города

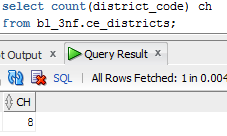
SELECT geo\_id||substr(city\_name,1,2)||'CT' as geo\_id,

Также были сделаны проверки на уникальность:

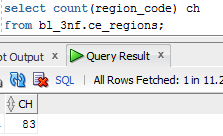
* Округа:



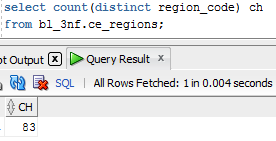
-------------------------------------------------------------------



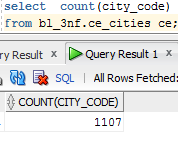
* Регионы:



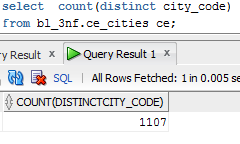
------------------------------------------------



* Города:



-----------------------------------------------------

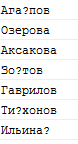


Везде значения совпали, значит дубликатов нет. Цель достигнута.

## Customers

Трудности:

1. Не было уникального кода для каждого клиента (id-значения повторяются).
2. Некоторые фамилия содержали в себе посторонние символы (знак вопроса):



1. Есть вероятность, что не все клиенты оставляют свое e-mail/номер телефона.

Решения:

1. Уникальный код я генерировала как id + первые три цифры номера кредитной карты.

SELECT distinct code||SUBSTR(card\_num,1,3)

1. С помощью функции REPLACE я убрала все вопростельные знаки.

INITCAP(REPLACE(sur,'?',''))

1. Использовала функцию NVL, чтобы все null-значения заменить на ‘N/D’

NVL(LOWER(email),'N/D') email,

NVL(phone,'N/D') phone

## Employees

Трудности:

1. Разные регистры при написании имен/фамилий.
2. У очень многих работников не было адреса электронной почты.
3. Генерация значений

Решения:

1. Функция INITCAP привела все к одному регистру:

INITCAP(surname) sur,

INITCAP(name) n

1. Функция NVL+LOWER:

NVL(phone,'N/D') phone,

NVL(LOWER(email),'N/D') email

1. Генерация осуществлялась с помощью dbms\_random.normal и специально написанной функции.

3.1) Id отдела сотрудника:

ABS(CEIL(dbms\_random.normal()\*4))+1

3.2) Id магазина, где работает сотрудник:

ABS(CEIL(dbms\_random.normal()\*65))+1

3.3) Id менеджера(руководителя)

Функция randomize\_man\_id

Я предусмотрела случай, что не у всех сотрудников будет менеджер (т.е.его id-номер - -99).

i:=dbms\_random.value(1,10);

a:=

CASE

WHEN i<=8 THEN

ROUND(dbms\_random.value(140,663))

WHEN i>8 THEN

'-99;

## Payments

Для таблицы Payments я не создавала cls-таблицу, а использовала табличну функцию, которую вызывала, когда переносила данные из wrk в 3НФ (подробнее о технической части можно почитать в отчете 8).

Вызов табличной функции:

SELECT code, type, descr, prepayment FROM TABLE(tab\_func())

## Catalog

В качестве бизнес-ключа используется **ISBN-код** (International Standard Book Number - уникальный номер книжного издания, необходимый для распространения книги в торговых сетях и автоматизации работы с изданием.

Также именно здесь осуществлялся маппинг на id-значения автора и жанра, которые вытягивались с предварительно заполненного уровня 3nf (таблицы ce\_authors и ce\_genres соответственно).

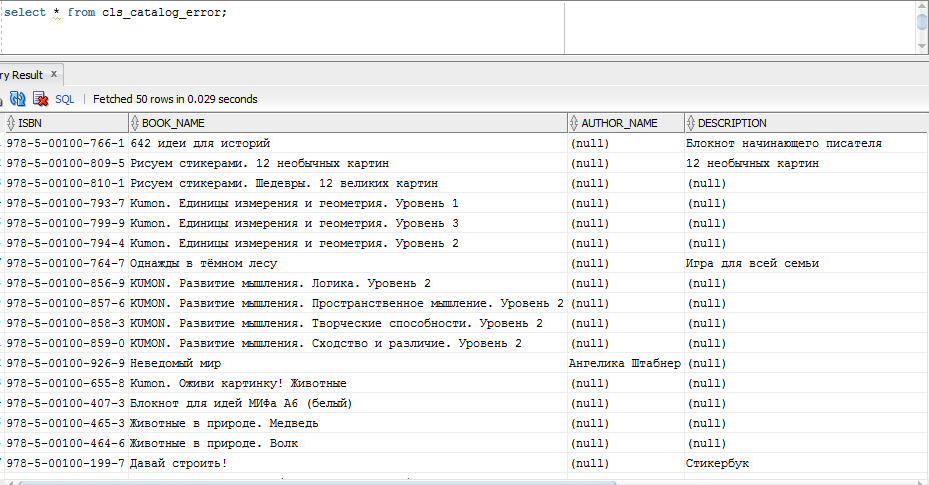
Трудности:

1. В каталоге кроме книг содержались данные о настольных играх, которые также имеются в продаже.
2. Не у всех книг было описание (description).

Решения:

1. Для того, чтобы отслеживать все некорректные данные, была создана **error-таблица.**

Как она выглядит:

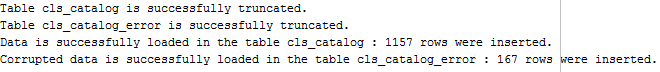


Как формируется:

В таблицу попадают те записи, в которых отсутствует автор,жанр или ISBN номер.

Для операции был использован явный курсор и bulk collect (для улучшения перфоманса).

Также в итоге можно посмотреть сколько записей попало в каждую из таблиц (был создан специальный счетчик, так как обычный %rowcount не подходит из-за цикла) :



1. Была использована функция NVL, чтобы заменить все null-значения на ‘N/D’

SELECT DISTINCT

NVL(t\_data(i).description, 'N/D')

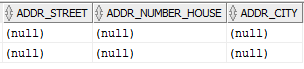
## Adresses

Трудности:

1. Были данные, где не было номера дома и улицы.

Решения:

1. Также была создана error-таблица.



## Stores\_SCD

Трудности:

1. Возникло много проблем, так как на сорсе не было никаких дат.
2. В cls-таблице нужны только новые зачения (которых еще нет в 3НФ) и значения за текущий период (isActive=’Y’), где произошли какие-то изменения.
3. Генерация id менеджера магазина.

Решения:

1. За start\_dt бралась текущая дата (sysdate)
2. Чтобы найти записи за текущий период, где произошли какие-то изменения, был сделан left join с таблицей магазинов из 3НФ, откуда бы брали данные, где что-то необходимо поменять:

left join bl\_3nf.ce\_stores ce

ON (ce.start\_dt<= TRUNC(sysdate)

and ce.end\_dt > TRUNC(sysdate)

and ce.store\_code = wrk.code)

WHERE DECODE(wrk.name,ce.store\_name,0,1)+DECODE(wrk.phone,ce.store\_phone,0,1)

+DECODE(addr\_id,ce.store\_address\_id,0,1)>0

1. Использовала функцию dbms\_random.value

round(dbms\_random.value(250,450))

# 3NF-tables

1. Перенос в 3НФ таблицы осуществлялся по одному и тому же прицнипу для всех (кроме stores\_scd):

Использовалась операция MERGE, где

* в условии USING c помощью оператора MINUS я выбирала записи, которых еще нет в 3НФ таблице
* в условии ON я испольовала бизнес-ключ
* Когда записи совпадали (WHEN MATCHED) по бизнес-ключу, я обновляла все колонки (кроме бизнес-ключа)
* Когда записи не совпадали (WHEN NOT MATCHED), я добавляла эту запись в 3НФ таблицу + добавляла id с помощью последовательности.

Все 3НФ таблицы типа SCD\_1, то есть имеют дату вставки и дату изменеий (sysdate по дефолту).

1. Логика немного другая была только для таблицы типа SCD\_2:

* в условии USING я не использовала minus, а выбирала сразу записи из всей cls-таблицы (SELECT \* FROM), потому что фильтрация произошла на уровне cls.
* Также в эту таблицу я не передаю значения ‘Y’/’N’ для колонки isActive, так как я использовала виртуальную колонку:

is\_active AS (

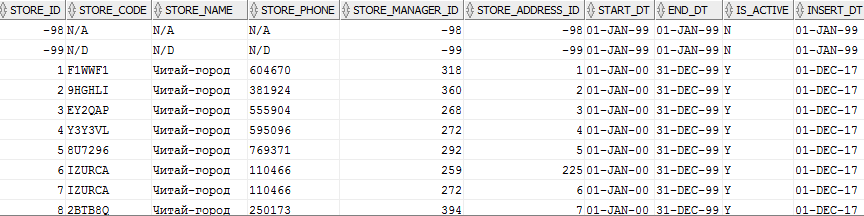
CASE

WHEN end\_dt=to\_date('31-DEC-9999')

THEN 'Y'

ELSE 'N'

END) ,



UPDATE wrk\_stores SET phone='000010' WHERE code = 'MS0BSU';

COMMIT;



Также во все таблицы были добавлены поля с id = -98 (N/A) и id = -99 (N/D).

1. Для фактовой таблицы использовала просто INSERT, так как в ней ничего не может изменяться.

# CL\_DM-TABLES

На этом уровне осуществлялась подготовка таблицы для переноса из 3НФ в DM.

Все сls-таблицы перед заполнением очищались с помощью операции truncate.

Именно здесь осуществлялся join с родительскими таблицами из DM.

Во всех INSERT-операциях я использовала условие

where ce.employee\_id>0;

чтобы не переносить значения -98 и -99.

# DM-tables

Использовался тот же принцип, что и в 3НФ (MERGE).

Отдельно опишу особенности процедуры для заполнения Dim\_Time\_Day:

## Dim\_Time\_day

Я написала процедуру для генерации всех необходимых полей для измерения дат.

В качестве входных параметров передаются значения годов, в промежутке которых надо сгенерировать измерение:

EXECUTE SP\_DATE\_DIMENSION(2005,2045);

Все индикаторы:

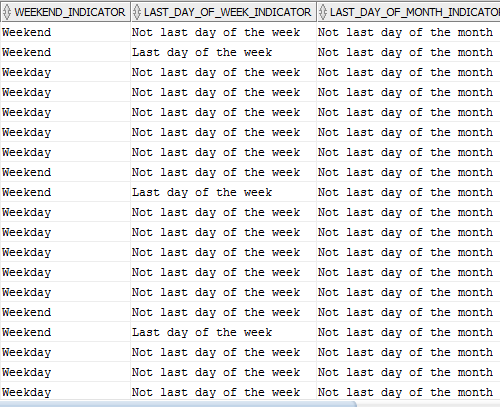
* + Weekend Indicator
  + Last Day Of Week Indicator
  + Last Day Of Month Indicator

заполнялись «говорящими значениями», так как:

Cryptic abbreviations, true/false flags, and operational indicators should be supplemented

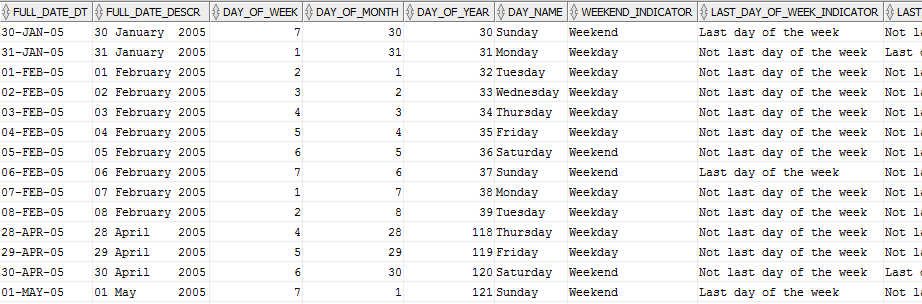
in dimension tables with full text words that have meaning when independently viewed.

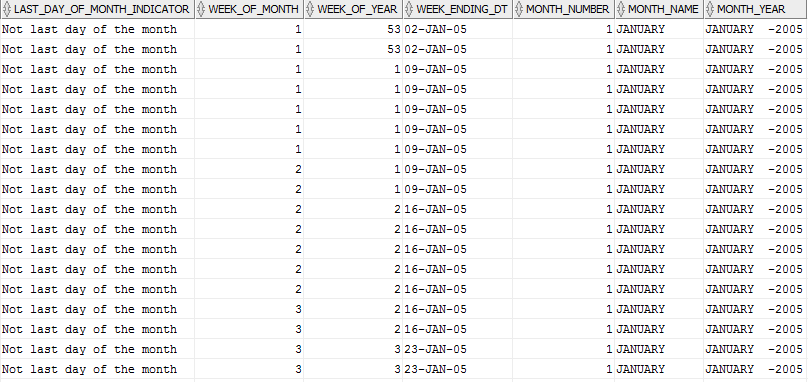
R.Kimball

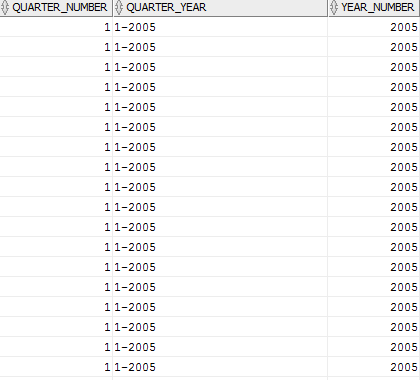


## Final results

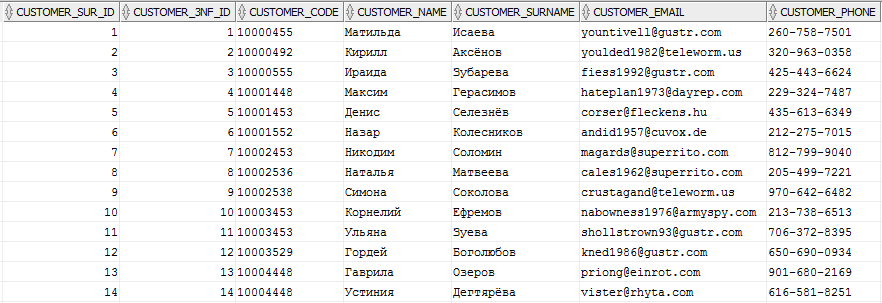
### Dim\_Time\_Day

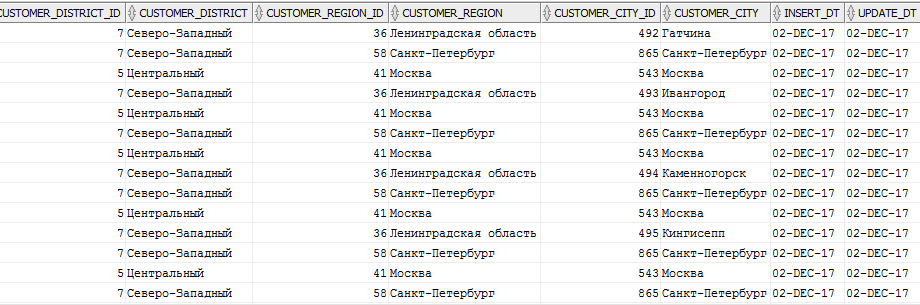




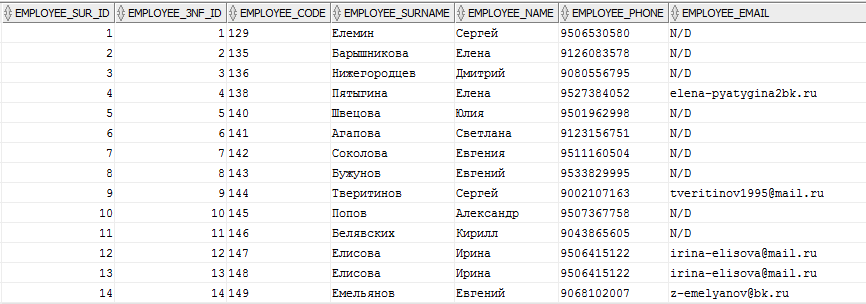


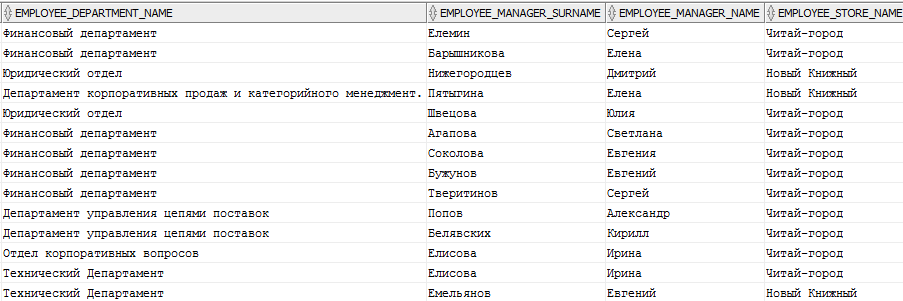
### Dim\_Customers

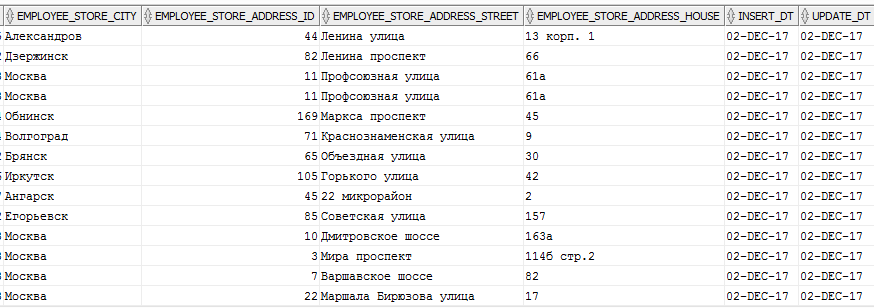




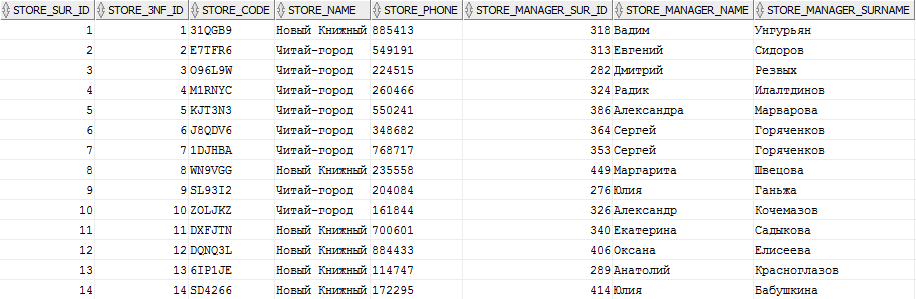
### Dim\_Employees

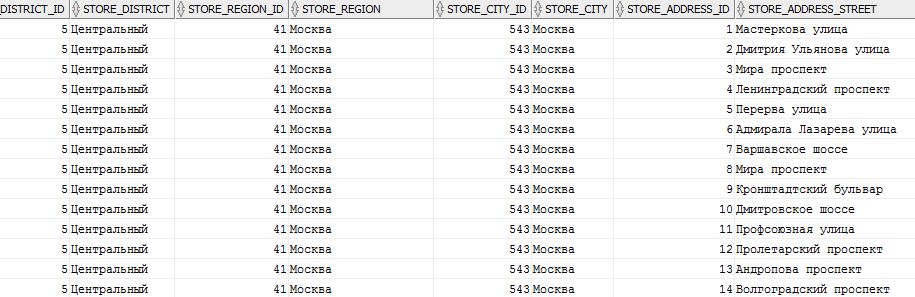


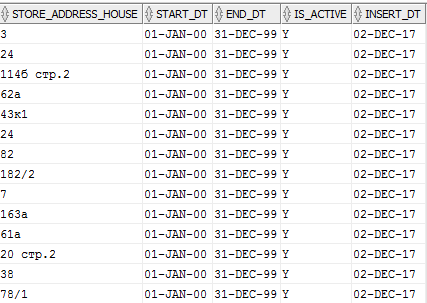




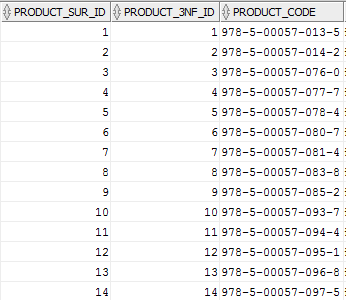
### Dim\_Stroes\_SCD

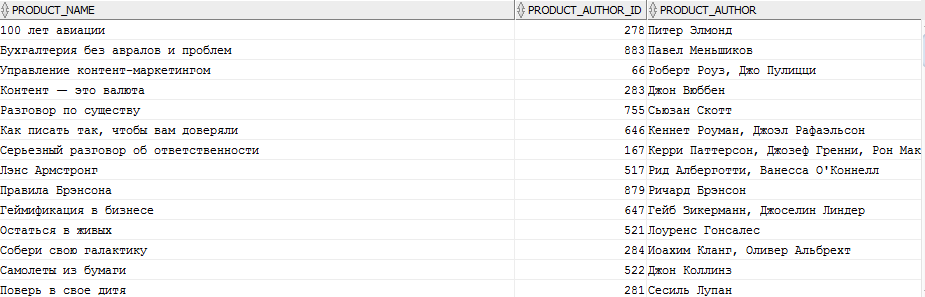


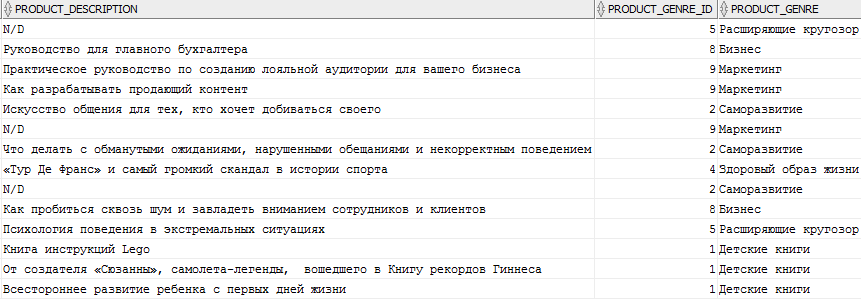


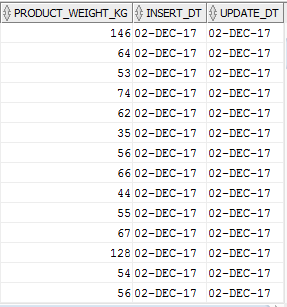


### Dim\_Products

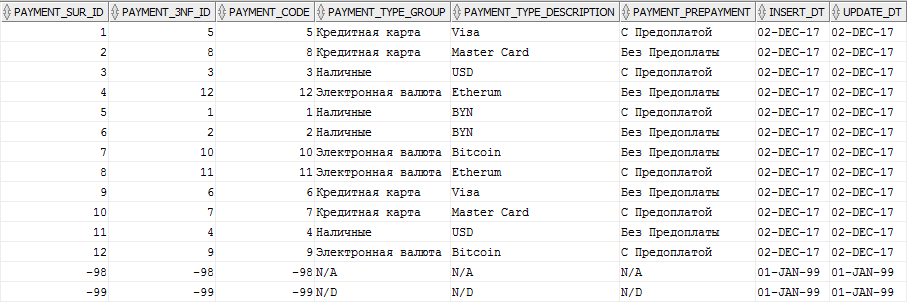




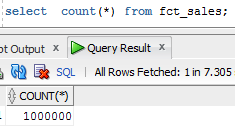


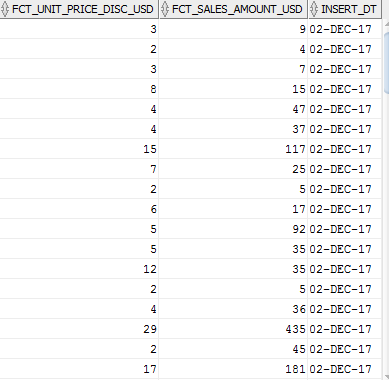
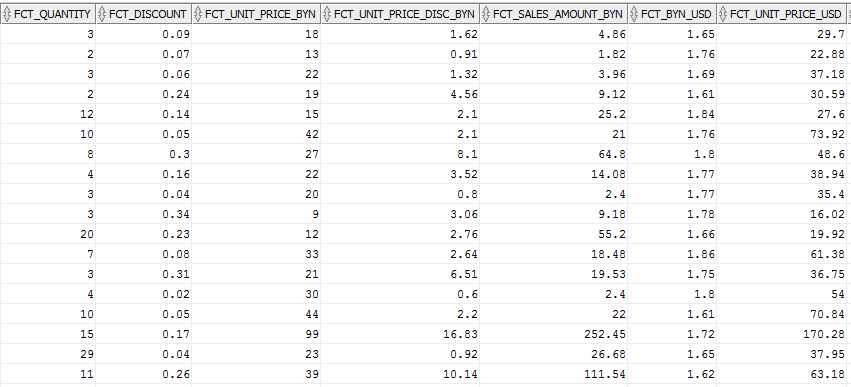
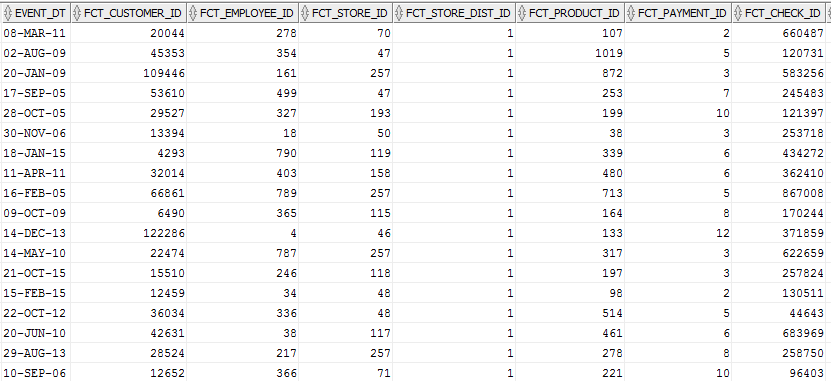


### Dim\_Payments



### Fct\_sales



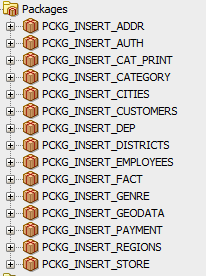


# Package structure and execution

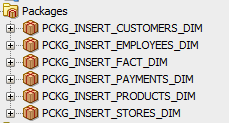
## Package structure

Все пакеты находятся на cls-уровнях:

* BL\_CL



* BL\_CL\_DM

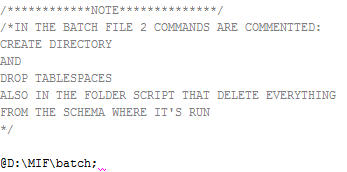


## Execution

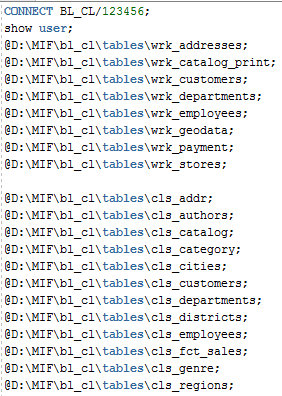
Также был создан batch-файл, который запускает другой файл, где все скрипты вызызваются в нужном порядке.

Код можно запускать **несколько раз.**

* Файл 1 (total\_execution.sql)



* Файл 2 (batch.sql)



## Repeatable execution

### SCD1 dimension

Обновила email на wrk-таблице (как будто новая инфа с источника пришла).

После этого повторно выполнила необходимые пакеты:

EXECUTE pckg\_insert\_employees.insert\_bl\_cls(source\_table\_wrk=>'wrk\_employees', target\_table\_cls=>'cls\_employees');

EXECUTE pckg\_insert\_employees.insert\_bl\_3nf;

EXECUTE pckg\_insert\_employees\_dim.insert\_bl\_cls;

EXECUTE pckg\_insert\_employees\_dim.insert\_bl\_dim;

Результат выполнения в ce\_таблице:

\* insert\_dt – 04-DEC-2017

\* update\_dt – 05-DEC-2017



Результат выполнения в dim\_таблице:

\* insert\_dt – 04-DEC-2017

\* update\_dt – 05-DEC-2017



### SCD2 dimension

Изменила колонку в wrk таблицы, как будто данные изменились на сорсе.

UPDATE wrk\_stores SET phone='000010' WHERE code = 'MS0BSU';

COMMIT;

После потоврного выполнения пакетов:

EXECUTE pckg\_insert\_store.insert\_bl\_cls(source\_table\_wrk=>'wrk\_stores', target\_table\_cls=>'cls\_stores');

EXECUTE pckg\_insert\_store.insert\_bl\_3nf;

EXECUTE pckg\_insert\_stores\_dim.insert\_bl\_cls;

EXECUTE pckg\_insert\_stores\_dim.insert\_bl\_dim;

SELECT \* FRM Dim\_Stores\_SCD

WHERE store\_code = 'MS0BSU';



Таким образом:

* SCD-измерение работает корректно.
* Пакеты reusable.

# Fact Table Partitioning Strategy

Партицирование будет осуществляться по полю Full\_Date\_DT по годам:

* Партиция с данными с 2005 по 2010 года.
* Отдельные партиции с данными за каждый год с 2010 по 2020.
* Партиция со всеми остальными годами (с 2020 по 2045).

Внутри также будет партицирование (composite partitioning) по округам, в которых находятся магазины.

Будет использоваться Range-List partition.

Я использовала PARTITION TEMPLATE.

PARTITION BY RANGE

(

event\_dt

)

SUBPARTITION BY LIST (Fct\_store\_dist\_id)

SUBPARTITION TEMPLATE

(SUBPARTITION south VALUES ('1') TABLESPACE tbs1,

SUBPARTITION fareast VALUES ('2') TABLESPACE tbs2,

SUBPARTITION volga VALUES ('3') TABLESPACE tbs3,

SUBPARTITION siberia VALUES ('4') TABLESPACE tbs4,

SUBPARTITION centr VALUES ('5') TABLESPACE tbs5,

SUBPARTITION northcauc VALUES ('6') TABLESPACE tbs6,

SUBPARTITION northwest VALUES ('7') TABLESPACE tbs7,

SUBPARTITION ural VALUES ('8') TABLESPACE tbs8

)

PARTITION sales\_2005\_2010 VALUES LESS THAN (TO\_DATE('01-JAN-2010','dd-Mon-yyyy')),

PARTITION sales\_2010\_2011 VALUES LESS THAN (TO\_DATE('01-JAN-2011','dd-Mon-yyyy')),

. . .

PARTITION sales\_2021\_more VALUES LESS THAN (maxvalue)

# Strategy of Parallel Load

Пареллельная загрузка использовалась при перемещении данных фактовой таблицы в DM-слой.

INSERT /\*+ PARALLEL(BL\_3NF.CE\_FCT\_SALES) \*/

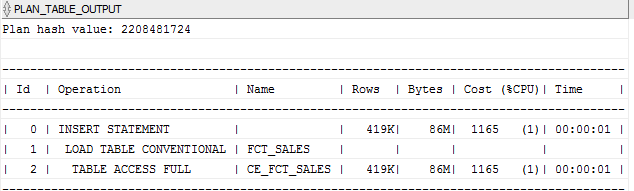
INTO bl\_dm.fct\_sales

(

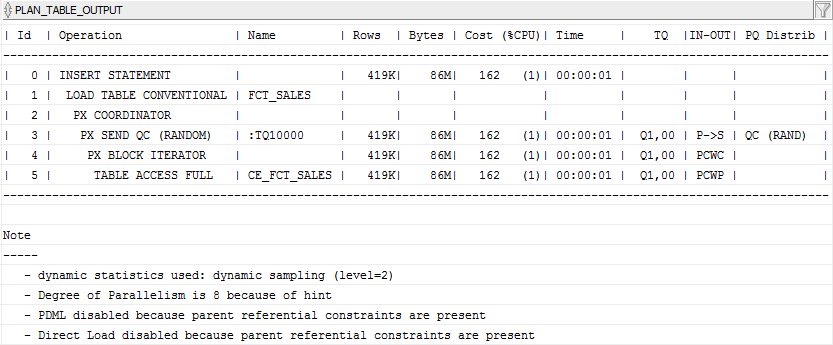
...

SELECT /\*+ PARALLEL(4) \*/

Без параллельной загрузки:



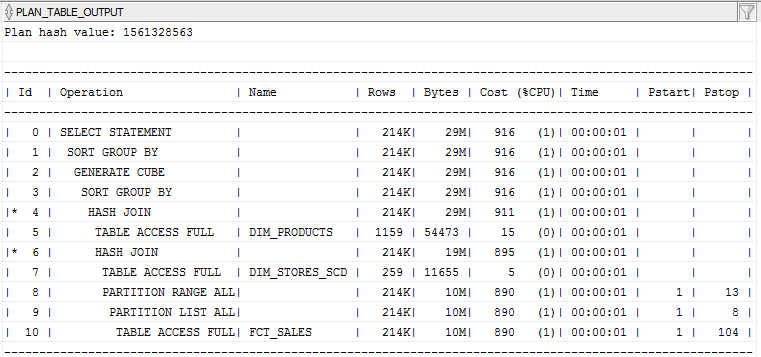
C параллельной загрузкой

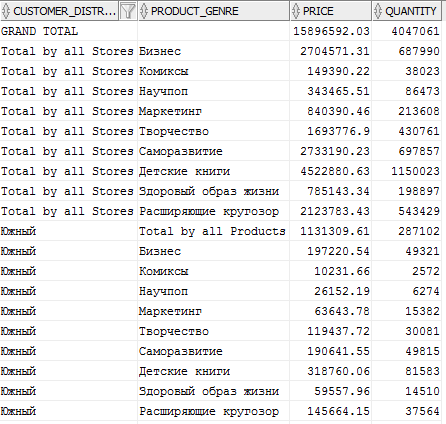


Разница в производительности практически в 10 раз.

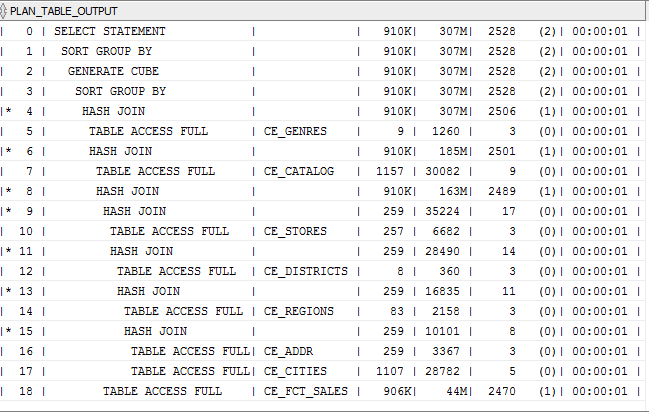
# Perfomanse 3NF VS STAR MODEL

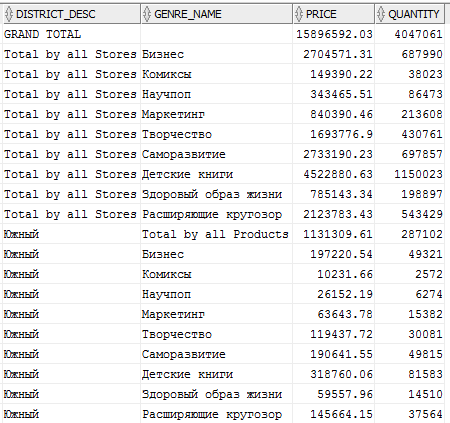
### Dim-layer





### 3NF layer





### Conclusion

Один и тот же запрос показывает разные результаты в зависимости от того, где он запущен:

* BL\_DM: cost=916
* BL\_3NF: cost=2528

Разница очевидна: на BL\_DM запрос выполняется практически в 3 раза эффективнее, чем на уровне BL\_3NF.

Это обусловлено тем, что в схеме Звезда данные уже сагрегированы, потому количество join-ов значительно меньше.

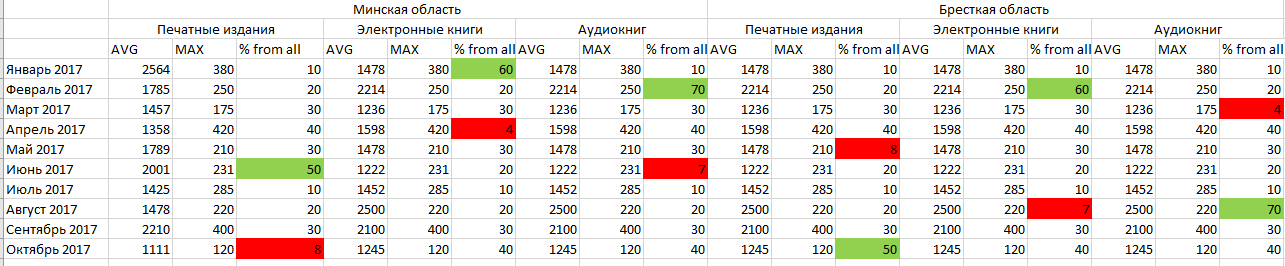
# Business analysis

Цель: Анализ продаж существующих категорий книг (печатные, электронные, аудио) за каждый месяц 2017 года (до октбря включительно, тк сейчас ноябрь) в Минской и Бресткой областях РБ.

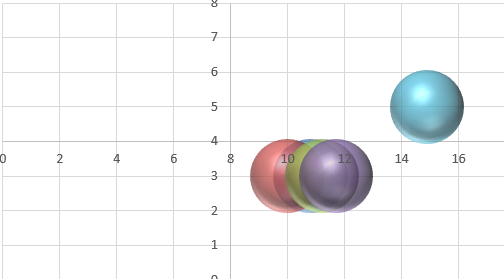
* AVG – среднее значени Sales\_amount
* MAX – максимальное значени Sales\_amount
* % of all – процент выручки от общегочисла продаж

Красный цвет: самый низкий уровень продаж.

Зеленый цвет: самый высокий уровень продаж.



Соотношение цена/количество проданныхх товаров по регионам.



| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| n.n |  |  | dd-Mmm-yyyy |  | dd-Mmm-yyyy |
|  |  |  |  |  |  |
|  |  |  |  |  |  |