January 2004

Personal Data Warehouse Application (Coursework Report)

By

Fahad Habib

ID: 02029881

fah086@londonmet.ac.uk

SB02G - Data Mining and Data Warehousing

Dr. Frank Wang

Semester A

MSc Software Engineering

2003-2004

Department of Computing

London Metropolitan University

TABLE OF CONTENTS

LIST OF FIGURES

1. PROPOSAL

This Personal Data Warehouse Application (PDWA) is based on the Private Practice Management System for doctor/medical personnel. The objective of this application is to provide solution for a Doctor to manage his/her private practice and to administer the patient and practice data effortlessly. The main components of the application are Patient record management, Practice details, Treatment records according to categories, Hospitals or procedure/practice sites and Payment system.

Typical relational database can be used to perform normal day-to-day operations at a private doctor surgery. Every new patient record will be maintained and can be used for future diagnostics and analysis. An invoice will be issued against each treatment performed on specific patient at any specific time. The record of payments received from the patient will be maintained in a separate table.

To implement this application in a data warehouse scenario, following measures can be taken:

- Charges for a patients from different area getting treatments for a certain period of time
- Identify the treatments mostly performed for treating patients of different area and ages

In this PDWA, I am using the sample data for radiology treatments. The information gathered in building this PDWA is provided by a Consultant Radiologist currently working in a hospital and who also provides private health services.

The PDWA will be implemented using Microsoft SQL Server 2000 Database system which also supports OLAP, Data warehousing and Data mining techniques. The sample data for the patients is generated using the actual data due to confidentiality of the personal data. Any resemblance with any person will be accidental and coincidental.

2. SCHEMA FOR PDWA

For creating a data warehouse, first we have to build a relational database. Based on that database, we can create dimensions and fact table for the data warehouse. Those dimensions and fact table will be used in the Star Schema for demonstrating the data warehouse.

2.1 What is a Star Schema?

It is a relational database schema for representing multidimensional data. The data is stored in a central fact table, with one or more tables holding information on each dimension. Dimensions have levels, and all levels are usually shown as columns in each dimension table [NP03].

2.2 Entity Relationship Diagram (ERD)

The scenario selected for the PDWA will contain following entities:

- Patients
- Practice
- Treatments
- Hospitals
- Payments

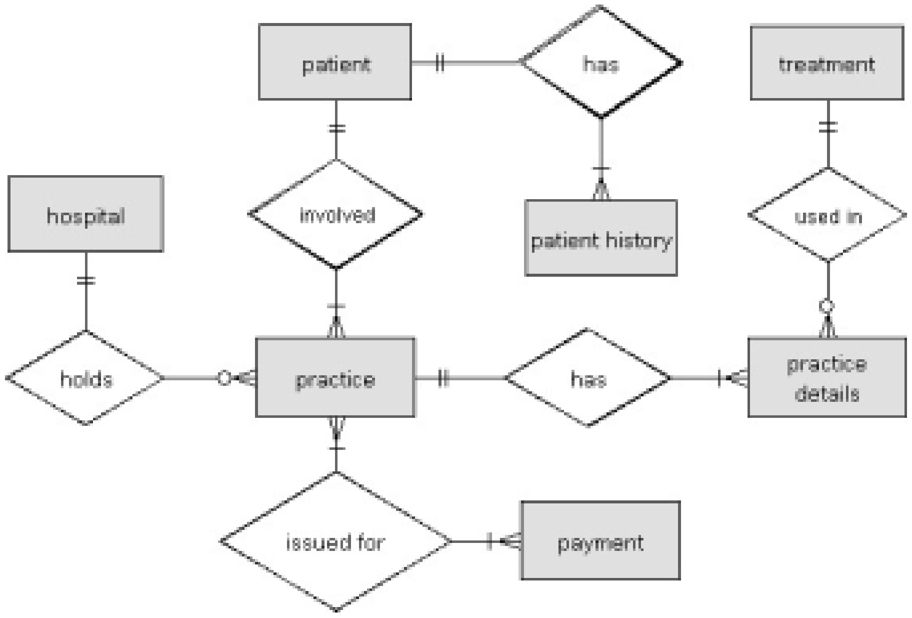


Fig 01: ERD for PDWA with cardinality and relationships

This ERD represents the entities and their relationship with each other. However, when we start designing of the physical database design, it is possible to further expand the entities represented in the above ERD.

For example, we can divide practice into 2 table i.e. `practice` and `practice_details` and `patient` table has `patientHistory`.

2.2.1 Detailed ERD with Primary, Foreign Keys

Following ERD shows entities with their attributes including primary and foreign keys.

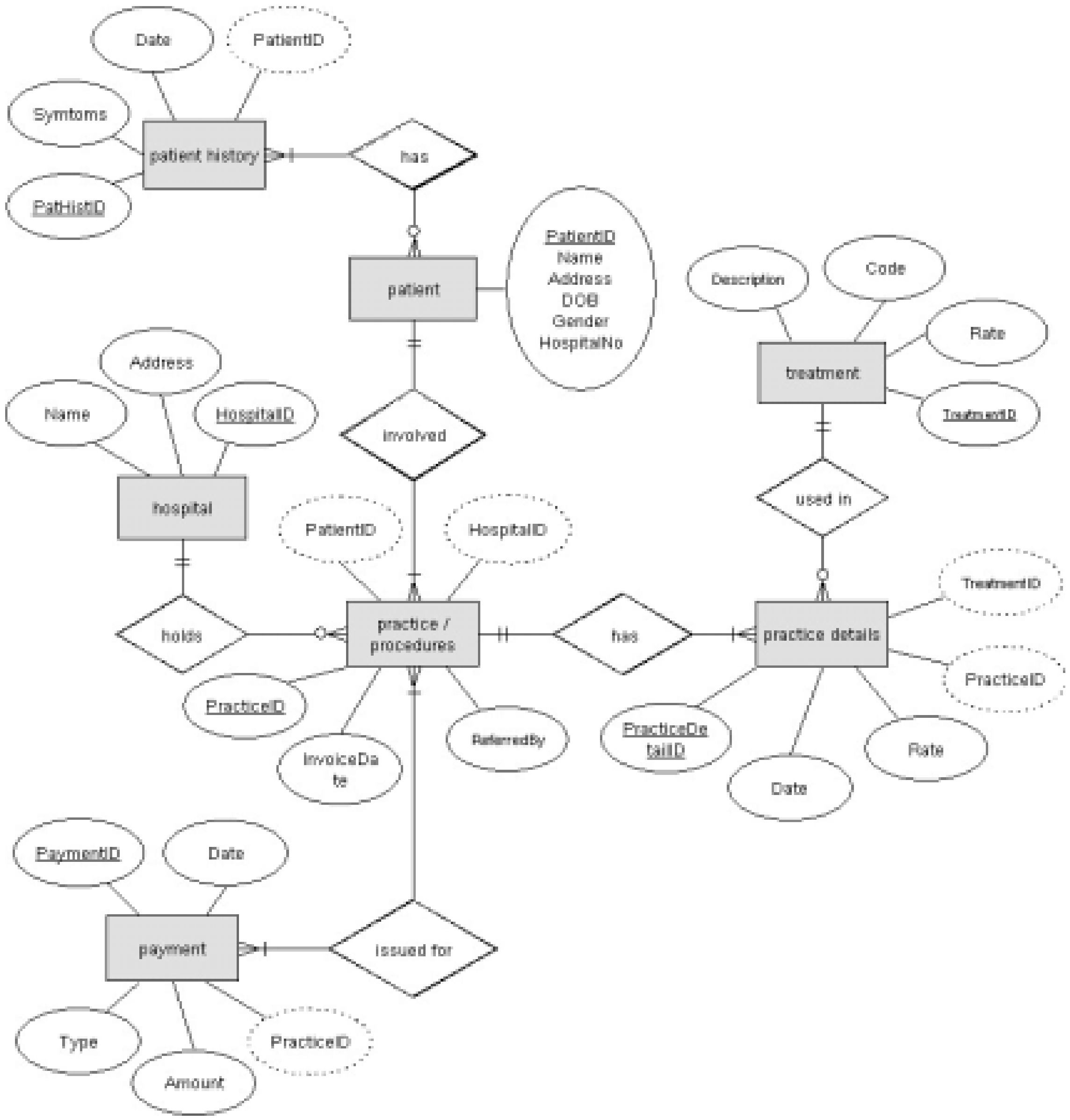


Fig 02: ERD with attributes, primary and foreign keys

2.3 Creating Database

Following command will create a relational database that will be used in the PDWA. The database name given for it is fahad. After that we will add tables to this database.

```
CREATE DATABASE [fahad]

ON (NAME = N'fahad_dat', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL\data\fahad.mdf', SIZE = 2, FILEGROWTH = 10%)

LOG ON (NAME = N'fahad_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL\data\fahad.ldf', SIZE = 2, FILEGROWTH = 10%)
```

2.4 Creating Tables

Following queries can be used in the creation of actual database tables and implementing constraints between them.

2.4.1 Hospitals Table

```
CREATE TABLE [dbo].[Hospitals] (
      [HospitalID] [int] IDENTITY (1, 1) NOT NULL,
      [HospitalName] [nvarchar] (50)  NOT NULL,
      [Address] [nvarchar] (255)  NOT NULL,
      [Town] [nvarchar] (50)  NULL,
      [City] [nvarchar] (25)  NULL,
      [Province] [nvarchar] (20)  NULL,
      [PostCode] [nvarchar] (20)  NULL,
      [PhoneNumber] [nvarchar] (15)  NULL,
      [FaxNumber] [nvarchar] (15)  NULL
) ON [PRIMARY]
```

2.4.2 Patients Table

```
CREATE TABLE [dbo].[Patients] (
      [PatientID] [int] IDENTITY (1, 1) NOT NULL,
      [FirstName] [nvarchar] (25)  NOT NULL,
      [LastName] [nvarchar] (25)  NOT NULL,
      [Address] [nvarchar] (255)  NULL,
      [Town] [nvarchar] (50)  NULL,
      [City] [nvarchar] (50)  NULL,
      [Province] [nvarchar] (20)  NULL,
      [PostCode] [nvarchar] (10)  NULL,
      [Country] [nvarchar] (50)  NULL,
      [Gender] [nvarchar] (8)  NOT NULL,
      [Birthdate] [smalldatetime] NULL,
      [HospitalNo] [nvarchar] (25)  NULL
) ON [PRIMARY]
```

2.4.3 Treatments Table

```
CREATE TABLE [dbo].[Treatments] (
      [TreatmentID] [int] IDENTITY (1, 1) NOT NULL,
      [TreatmentDesc] [nvarchar] (250)  NOT NULL,
      [TreatmentRate] [money] NULL
      [TreatmentType] [nvarchar] (50)  NULL,
) ON [PRIMARY]
```

2.4.4 fact_table Table

```
CREATE TABLE [dbo].[fact_table] (
      [PracticeID] [nvarchar] (50)  NULL,
      [PracticeDetailID] [int] NULL,
      [PatientID] [int] NULL,
      [TreatmentID] [int] NULL,
      [PracticeDate] [datetime] NULL,
      [total_fees] [money] NULL,
      [age] [char] (10)  NULL,
      [symtoms] [nvarchar] (50)  NULL,
      [area] [nvarchar] (50)  NULL
) ON [PRIMARY]
```

2.4.5 PatientHistory Table

```
CREATE TABLE [dbo].[PatientHistory] (
      [Pat_HistID] [numeric](18, 0) IDENTITY (1, 1) NOT NULL,
      [PatientID] [int] NOT NULL,
      [PatSymtoms] [nvarchar] (100)  NULL,
      [PatCheckingDate] [datetime] NULL
) ON [PRIMARY]
```

2.4.6 Practice Table

```
CREATE TABLE [dbo].[Practice] (
      [PracticeID] [nvarchar] (50)  NOT NULL,
      [PatientID] [int] NOT NULL,
      [HospitalID] [int] NULL,
      [InvoiceDate] [datetime] NULL,
      [AdditionalNotes] [nvarchar] (200)  NULL
) ON [PRIMARY]
```

2.4.7 Payments Table

```
CREATE TABLE [dbo].[Payments] (
      [PaymentID] [int] IDENTITY (1, 1) NOT NULL,
      [PracticeID] [nvarchar] (50)  NOT NULL,
      [PaymentAmount] [money] NOT NULL,
      [PaymentDate] [smalldatetime] NULL,
      [PaymentType] [nvarchar] (50)  NULL,
      [ChequeNumber] [nvarchar] (50)  NULL
) ON [PRIMARY]
```

2.4.8 PracticeDetails Table

```
CREATE TABLE [dbo].[PracticeDetails] (
      [PracticeDetailID] [int] IDENTITY (1, 1) NOT NULL,
      [PracticeID] [nvarchar] (50)  NOT NULL,
      [TreatmentID] [int] NULL,
      [Rate] [money] NULL,
      [PDate] [smalldatetime] NULL
) ON [PRIMARY]
```

2.4.9 times Table

```
CREATE TABLE [dbo].[times] (
      [timeID] [numeric](18, 0) IDENTITY (1, 1) NOT NULL,
      [the_date] [datetime] NULL,
      [the_day] [nvarchar] (50) NULL,
      [the_month] [nvarchar] (50) NULL,
      [the_year] [numeric](18, 0) NULL,
      [day_of_month] [numeric](18, 0) NULL,
      [week_of_year] [numeric](18, 0) NULL,
      [month_of_year] [numeric](18, 0) NULL,
      [quarter_of_year] [nvarchar] (50) NULL
) ON [PRIMARY]
```

2.5 Constraints and Relationships between tables

After creating tables, next step is to assign Primary and Foreign keys and creating links between them. This can be done by identifying foreign keys and enforcing integrity constraints between them, so that data reliability remains intact.

2.5.1 Assigning Primary Keys

Following commands can be used in implementing Primary keys.

```
ALTER TABLE [dbo].[Hospitals] WITH NOCHECK ADD
      CONSTRAINT [PK_Hospitals] PRIMARY KEY  CLUSTERED
      (
            [HospitalID]
      )  ON [PRIMARY]

ALTER TABLE [dbo].[Patients] WITH NOCHECK ADD
      CONSTRAINT [PK_Patients] PRIMARY KEY  CLUSTERED
      (
            [PatientID]
      )  ON [PRIMARY]

ALTER TABLE [dbo].[Treatments] WITH NOCHECK ADD
      CONSTRAINT [PK_Treatments] PRIMARY KEY  CLUSTERED
      (
            [TreatmentID]
      )  ON [PRIMARY]



ALTER TABLE [dbo].[PatientHistory] WITH NOCHECK ADD
      CONSTRAINT [PK_PatientHistory] PRIMARY KEY  CLUSTERED
      (
            [Pat_HistID]
      )  ON [PRIMARY]

ALTER TABLE [dbo].[Practice] WITH NOCHECK ADD
      CONSTRAINT [PK_Practice] PRIMARY KEY  CLUSTERED
      (
            [PracticeID]
      )  ON [PRIMARY]

ALTER TABLE [dbo].[Payments] WITH NOCHECK ADD
      CONSTRAINT [PK_Payments] PRIMARY KEY  CLUSTERED
      (
            [PaymentID]
      )  ON [PRIMARY]

ALTER TABLE [dbo].[PracticeDetails] WITH NOCHECK ADD
      CONSTRAINT [PK_PracticeDetails] PRIMARY KEY  CLUSTERED
      (
            [PracticeDetailID]
      )  ON [PRIMARY]
```

2.5.2 Assigning Foreign Keys and Constraints

Following commands can be used in implementing foreign keys, relationships and integrity constraints.

```
ALTER TABLE [dbo].[PatientHistory] ADD
      CONSTRAINT [FK_PatientHistory_Patients] FOREIGN KEY
      (
            [PatientID]
      ) REFERENCES [dbo].[Patients] (
            [PatientID]
      ) ON UPDATE CASCADE

ALTER TABLE [dbo].[Practice] ADD
      CONSTRAINT [FK_Practice_Hospitals] FOREIGN KEY
      (
            [HospitalID]
      ) REFERENCES [dbo].[Hospitals] (
            [HospitalID]
      ) ON UPDATE CASCADE,
      CONSTRAINT [FK_Practice_Patients] FOREIGN KEY
      (
            [PatientID]
      ) REFERENCES [dbo].[Patients] (
            [PatientID]
      ) ON DELETE CASCADE  ON UPDATE CASCADE

ALTER TABLE [dbo].[Payments] ADD
      CONSTRAINT [FK_Payments_Practice] FOREIGN KEY
      (
            [PracticeID]
      ) REFERENCES [dbo].[Practice] (
            [PracticeID]
      ) ON DELETE CASCADE  ON UPDATE CASCADE

ALTER TABLE [dbo].[PracticeDetails] ADD
      CONSTRAINT [FK_PracticeDetails_Practice] FOREIGN KEY
      (
		[PracticeID]
	) REFERENCES [dbo].[Practice] (
		[PracticeID]
	) ON DELETE CASCADE  ON UPDATE CASCADE,
	CONSTRAINT [FK_PracticeDetails_Treatments] FOREIGN KEY
	(
		[TreatmentID]
	) REFERENCES [dbo].[Treatments] (
		[TreatmentID]
	) ON UPDATE CASCADE
```

2.6 Database Schema

After creating my database, I use the *Diagrams* tool of SQL server to see the relationships and constraints between the tables. Following figure shows the database schema created.

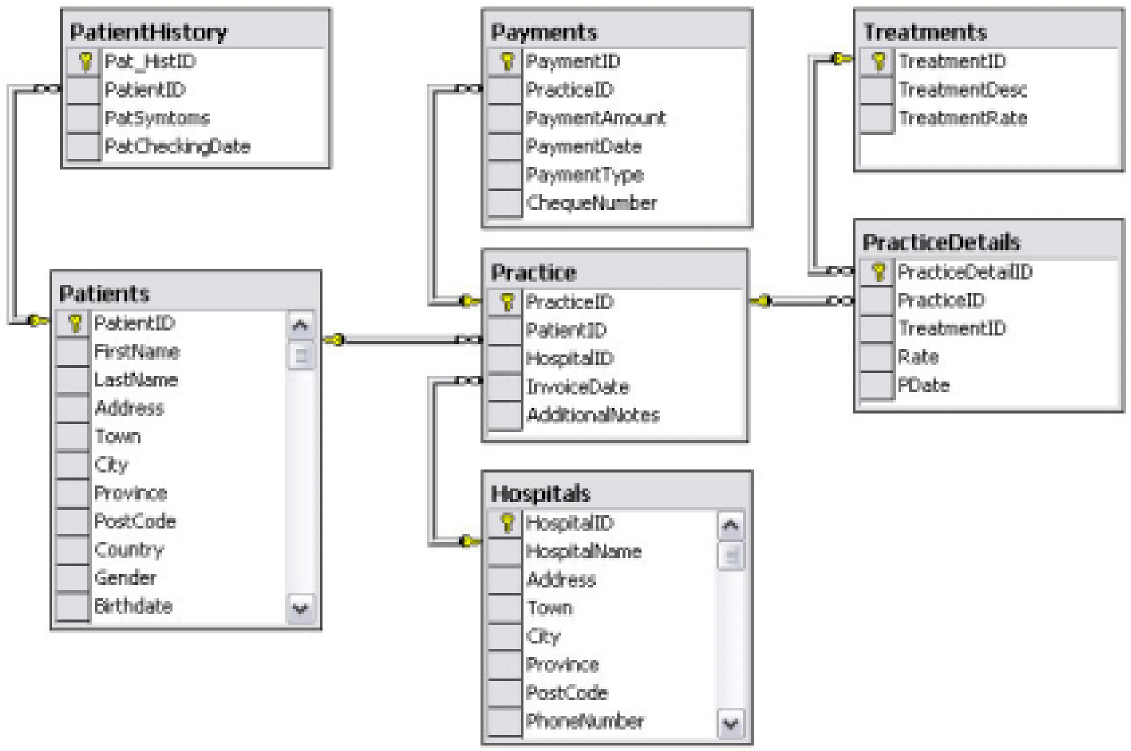
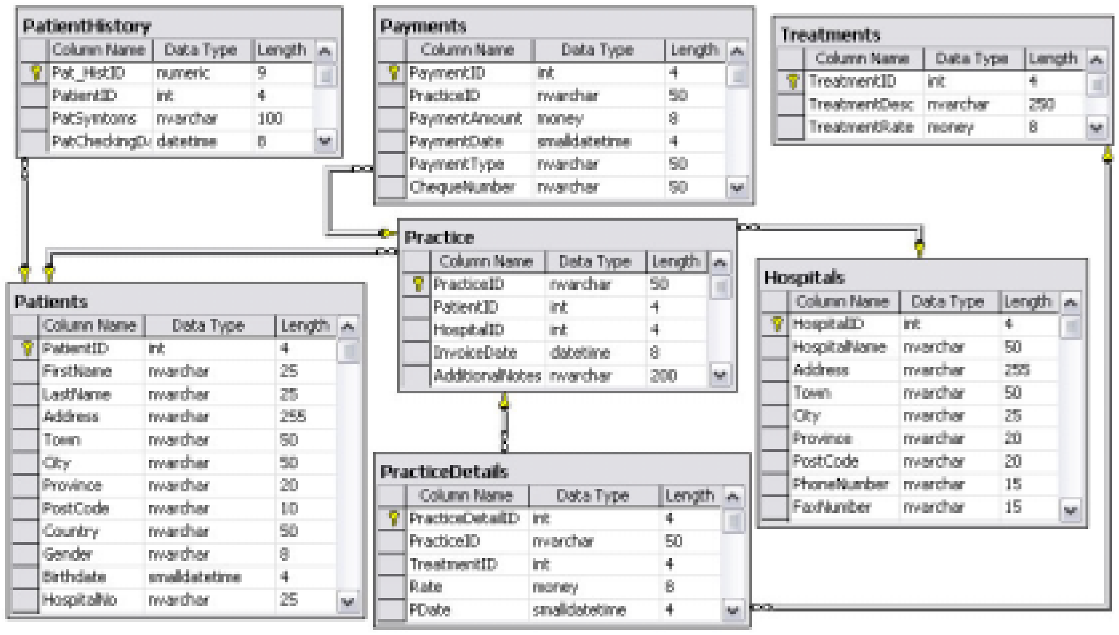


Fig 03: Database Schema for PDWA

2.7 Database Schema with Attribute types and Size

The next figure shows the database schema with Primary keys, foreign keys and other attributes with these Data Types and Sizes as specified during the database creation process.



PatientHistory

Column Name	Data Type	Length
Pat_HistID	numeric	9
PatientID	int	4
PatSyntoms	nvarchar	100
PatCheckingD.	datetime	8

Payments

Column Name	Data Type	Length
PaymentID	int	4
PracticeID	nvarchar	50
PaymentAmount	money	8
PaymentDate	smalldatetime	4
PaymentType	nvarchar	50
ChequeNumber	nvarchar	50

Treatments

Column Name	Data Type	Length
TreatmentID	int	4
TreatmentDesc	nvarchar	250
TreatmentRate	money	8

Practice

Column Name	Data Type	Length
PracticeID	nvarchar	50
PatientID	int	4
HospitalID	int	4
InvoiceDate	datetime	8
AdditionalNotes	nvarchar	200

Hospitals

Column Name	Data Type	Length
HospitalID	int	4
HospitalName	nvarchar	50
Address	nvarchar	255
Town	nvarchar	50
City	nvarchar	25
Province	nvarchar	20
PostCode	nvarchar	20
PhoneNumber	nvarchar	15
FaxNumber	nvarchar	15

Patients

Column Name	Data Type	Length
PatientID	int	4
FirstName	nvarchar	25
LastName	nvarchar	25
Address	nvarchar	255
Town	nvarchar	50
City	nvarchar	50
Province	nvarchar	20
PostCode	nvarchar	10
Country	nvarchar	50
Gender	nvarchar	8
Birthdate	smalldatetime	4
HospitalNo	nvarchar	25

PracticeDetails

Column Name	Data Type	Length
PracticeDetailID	int	4
PracticeID	nvarchar	50
TreatmentID	int	4
Rate	money	8
PDate	smalldatetime	4

Fig 04: Database Schema with Types and Size

2.8 Star Schema

After establishing relational database, we are now able to produce dimensions and a fact table for those dimensions to start building our data warehouse. The following diagram illustrates Star Schema for our personal data warehouse.

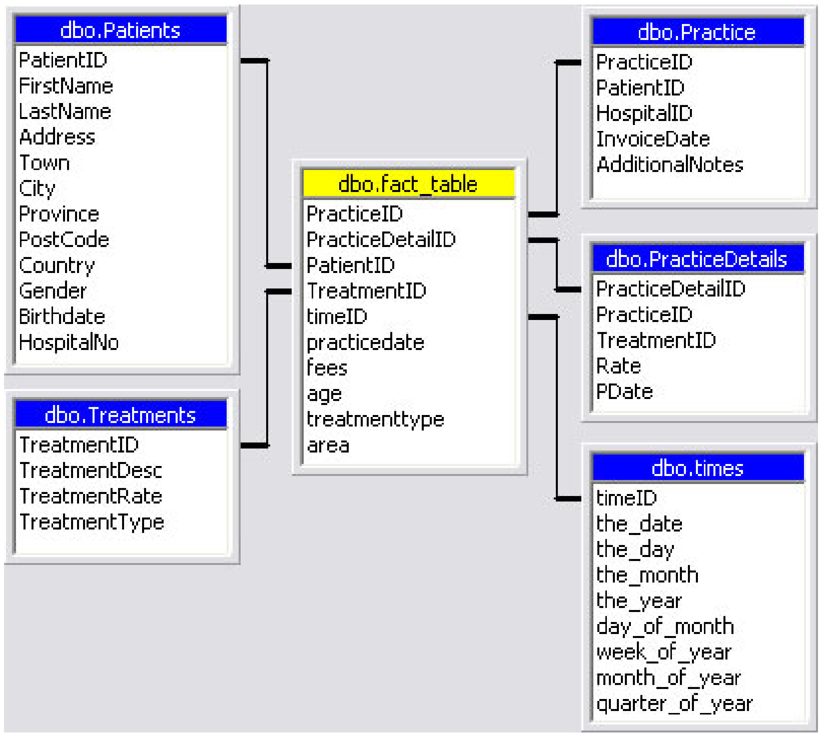


Fig 05: Star Schema for PDWA

2.9 SQL Queries

2.9.1 Insert command for Patients table

Following command can be used to populate patients table with data.

Input:

```
INSERT INTO patients (firstname, lastname, address, town, city,
            province, postcode, country, gender, birthdate,
            hospitalno)
VALUES ('Fahad', 'Habib', '123 London Road,', 'Ilford', 'London',
            'Essex', 'IG2 6RY', 'UK', 'Male',
            '1-Aug-1980', 'K123456')
```

Output:

```
(1 row(s) affected)
```

2.9.2 Insert command for Fact Table

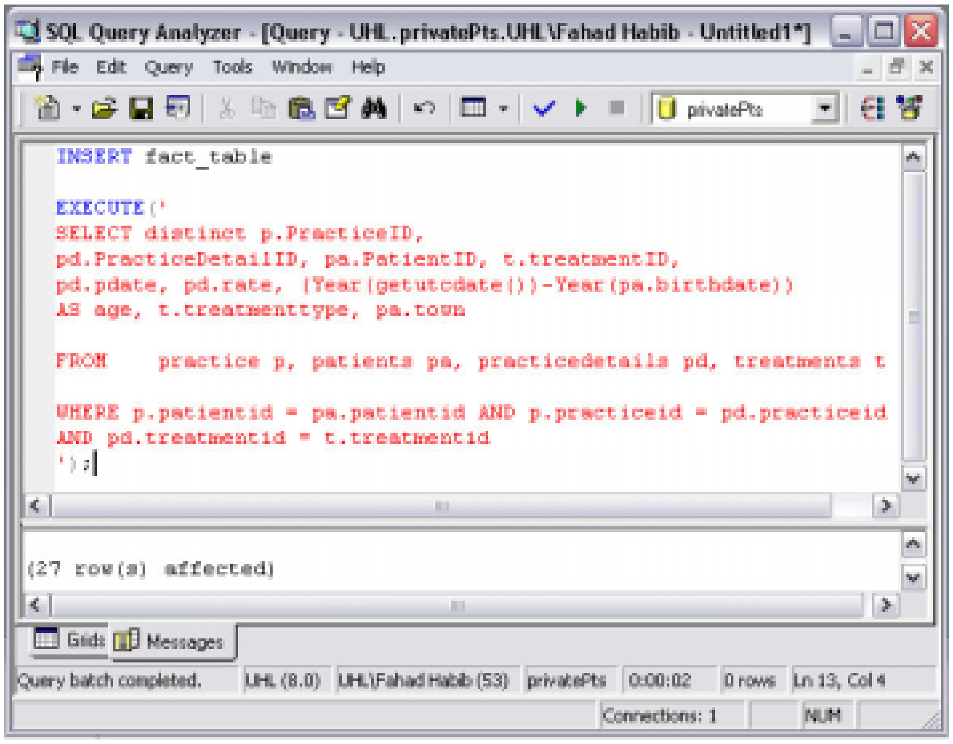
Following command can be used to populate our fact_table with data from various tables.

Input:

```
INSERT fact_table

EXECUTE('
      SELECT distinct p.PracticeID,
      pd.PracticeDetailID, pa.PatientID, t.treatmentID, ti.timeid,
      pd.pdate, pd.rate, (Year(getutcdate())-Year(pa.birthdate))
      AS age, t.treatmenttype, pa.town


      FROM  practice p, patients pa, practicedetails pd, treatments t

      WHERE p.patientid = pa.patientid AND p.practiceid = pd.practiceid
      AND pd.treatmentid = t.treatmentid AND pd.pdate = ti.the_date
      ');
```

Output:

```
(27 row(s) affected)
```


SQL Query Analyzer - [Query - UHL.privatePts.UHL\Fahad Habib - Untitled1*]
File Edit Query Tools Window Help
privatePts
Grids Messages

```
INSERT fact_table

EXECUTE('
SELECT distinct p.PracticeID,
pd.PracticeDetailID, pa.PatientID, t.treatmentID,
pd.pdate, pd.rate, (Year(getutcdate())-Year(pa.birthdate))
AS age, t.treatmenttype, pa.town

FROM    practice p, patients pa, practicedetails pd, treatments t

WHERE p.patientid = pa.patientid AND p.practiceid = pd.practiceid
AND pd.treatmentid = t.treatmentid
');
```

```
(27 row(s) affected)
```

Query batch completed. | UHL (8.0) | UHL\Fahad Habib (53) | privatePts | 0:00:02 | 0 rows | Ln 13, Col 4
Connections: 1 | NUM

Fig 06: Insert command for Fact Table

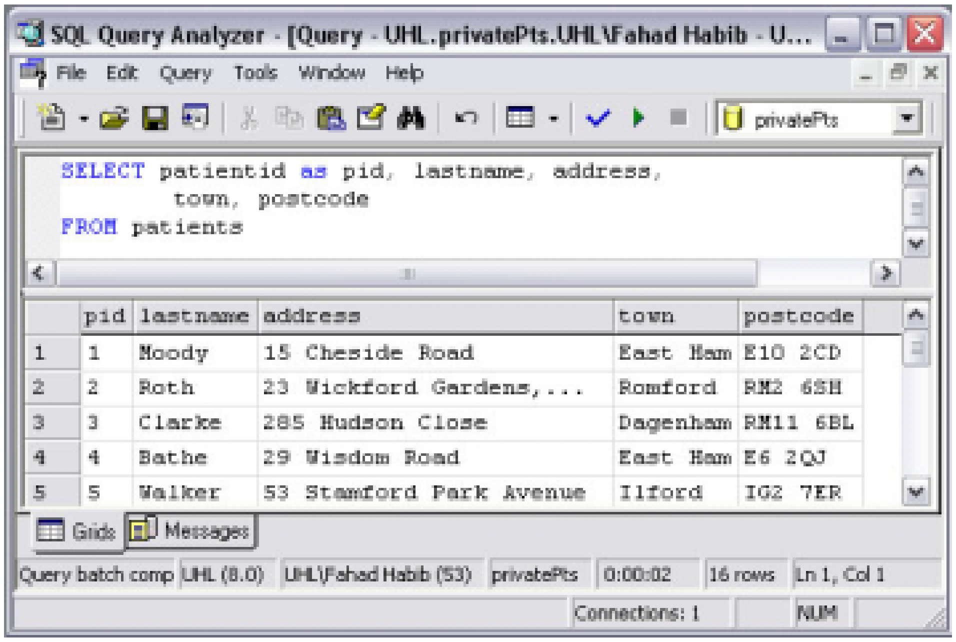
2.9.3 Select command for Patients table

Input:

```
SELECT patientid as pid, lastname, address, town, postcode
FROM patients
```

Output:

```
pid   lastname    address                             town       postcode
---   --------    -----------------                   --------   --------
1     Moody       15 Cheside Road                     East Ham   E10 2CD
2     Roth        23 Wickford Gardens, Reebok Road    Romford    RM2 6SH
3     Clarke      285 Hudson Close                    Dagenham   RM11 6BL
4     Bathe       29 Wisdom Road                      East Ham   E6 2QJ
5     Walker      53 Stamford Park Avenue             Ilford     IG2 7ER
6     Bang        34 Millford Avenue, Gants Hill      Ilford     IG3 8JX
7     Brown       Trickfield Mansion, High Street     Barking    IG1 4BC
8     Margreat    71 Mystery Road                     Dagenham   RM7 6DB
9     Beckham     2 Fuller Gardens                    Ilford     IG3 5AK
10    Butcher     Terriffic Court, Major Road         Barking    IG11 2RQ
11    Campbell    10 Fieldway Avenue                  Dagenham   RM7 7TK
12    Blake       96 Hardrock Street                  Ilford     IG9 4WD
13    Gordon      13 Sydney Road                      Romford    RM4 8KR
14    William     42 Speedy Drive, Gants Hill         Ilford     IG5 8WB
15    Ferguson    172 Bricken Road                    Barking    IG12 7TG
16    Habib       123 London Road                     Ilford     IG2 6RY
```


SQL Query Analyzer - [Query - UHL.privatePts.UHL\Fahad Habib - U...
File Edit Query Tools Window Help
privatePts
Grids Messages

```
SELECT patientid as pid, lastname, address,
       town, postcode
FROM patients
```

	pid	lastname	address	town	postcode
1	1	Moody	15 Cheside Road	East Ham	E10 2CD
2	2	Roth	23 Wickford Gardens,...	Romford	RM2 6SH
3	3	Clarke	285 Hudson Close	Dagenham	RM11 6BL
4	4	Bathe	29 Wisdom Road	East Ham	E6 2QJ
5	5	Walker	53 Stamford Park Avenue	Ilford	IG2 7ER

Query batch comp UHL (8.0) UHL\Fahad Habib (53) privatePts 0:00:02 16 rows Ln 1, Col 1
Connections: 1 NUM

Fig 07: Select command for Patient Table

2.9.4 Select command for Fact table

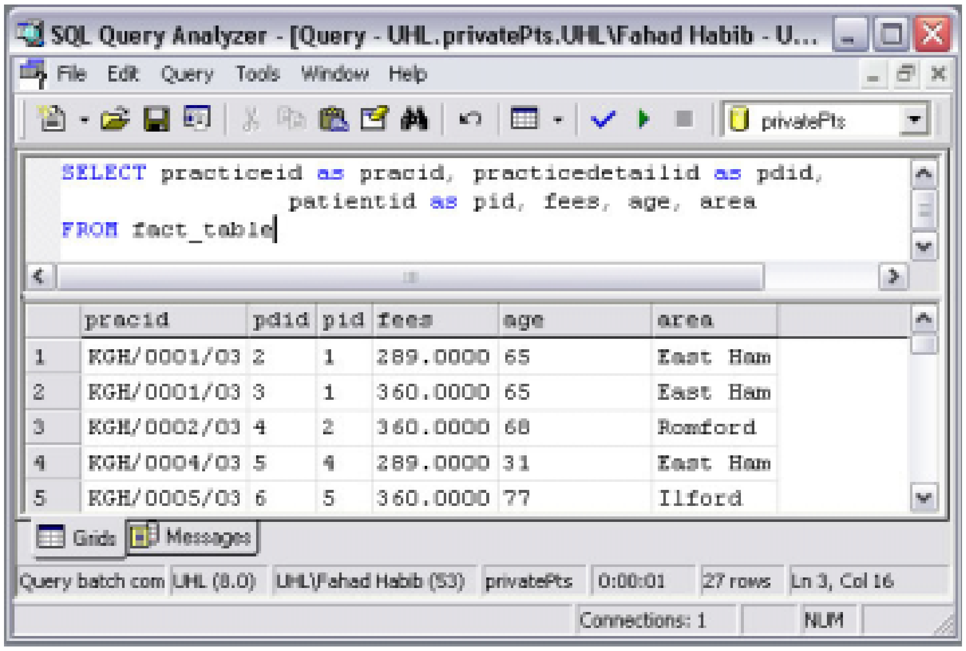
Input:

```
SELECT practiceid as pracid, practicedetailid as pdid,
      patientid as pid, fees, age, area
FROM fact_table
```

Output:

```
pracid             pdid  pid   total_fees  age         area
-----------        ----- ----  ----------- ----        ---------
KGH/0001/03        2     1     289.00      65          East Ham
KGH/0001/03        3     1     360.00      65          East Ham
KGH/0002/03        4     2     360.00      68          Romford
KGH/0004/03        5     4     289.00      31          East Ham
KGH/0005/03        6     5     360.00      77          Ilford
KGH/0005/03        7     5     508.00      77          Ilford
KGH/0006/03        8     2     508.00      68          Romford
KGH/0007/03        9     6     289.00      66          Ilford
KGH/0008/03        10    7     249.00      38          Barking
KGH/0003/03        11    3     782.00      55          Dagenham
KGH/0009/03        12    8     782.00      79          Dagenham
KGH/0009/03        13    8     386.00      79          Dagenham
KGH/0010/03        14    9     386.00      61          Ilford
KGH/0011/03        15    10    289.00      48          Barking
KGH/0012/03        16    9     548.00      61          Ilford
KGH/0012/03        17    9     386.00      61          Ilford
KGH/0012/03        18    9     400.00      61          Ilford
KGH/0013/03        19    11    782.00      27          Dagenham
KGH/0013/03        20    11    400.00      27          Dagenham
KGH/0013/03        21    11    548.00      27          Dagenham
KGH/0014/03        22    12    360.00      78          Ilford
KGH/0014/03        23    12    508.00      78          Ilford
KGH/0015/03        24    13    360.00      67          Romford
```

```
KGH/0016/03        25     13     508.00        67           Romford
KGH/0017/03        26     14     360.00        52           Ilford
KGH/0017/03        27     14     508.00        52           Ilford
KGH/0018/03        28     15     452.00        84           Barking
```


SQL Query Analyzer - [Query - UHL.privatePts.UHL\Fahad Habib - U...
File Edit Query Tools Window Help
privatePts
Grids
Messages

```
SELECT practiceid as pracid, practicedetailid as pdid,
            patientid as pid, fees, age, area
FROM fact_table
```

	pracid	pdid	pid	fees	age	area
1	KGH/0001/03	2	1	289.0000	65	East Ham
2	KGH/0001/03	3	1	360.0000	65	East Ham
3	KGH/0002/03	4	2	360.0000	68	Romford
4	KGH/0004/03	5	4	289.0000	31	East Ham
5	KGH/0005/03	6	5	360.0000	77	Ilford

Query batch com | UHL (8.0) | UHL\Fahad Habib (53) | privatePts | 0:00:01 | 27 rows | Ln 3, Col 16

Connections: 1 | NUM

Fig 08: Select command for Fact Table

3. OLAP OPERATIONS

After creation of the database with fact table and inserting of sample records, now I am able to perform certain On-line Analytical Processing (OLAP) operations on my database.

OLAP provides a user-friendly environment for interactive data analysis [HK01]. Some of the typical OLAP operations are [HK01]:

1. Roll-up
2. Drill-down
3. Slice and Dice
4. Pivot
5. Others like
    a. Drill across
    b. Drill through

From these available OLAP operations, I will try to demonstrate **drill-down** and **slice** operations on my logical data cube according to my scenario and information available. First of all let me formulate my base cuboid, depending on that base cuboid I will perform OLAP operations.

3.1 Base Cuboid

To make a base cuboid, following measure can be assumed according to my scenario.

"We want to calculate Average Fees (fees) earned from different Areas (towns) of the city and of different Treatments (types) over specific amount of Time (quarters)."

The aim of this measure is to analyze the earnings and possible treatment from different geographical location and to predict the future practice of patients from dissimilar areas. After calculating this measure, by just asking area of the patient, we can assess that what kind of treatment we may provide and how much practice we can expect from that patient.

Following query based on my fact table will show the data required for this cuboid.

Input:

```
SELECT t.quarter_of_year AS Quarters, Avg(ft.fees) AS Avg_Fees,
      ft.Area, ft.TreatmentType
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid
GROUP by ft.area,  ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

Output:

```
Quarters    Avg_Fees    Area        TreatmentType
--------    --------    -------     --------------
Q1          269.00      Barking     Biopsy
Q1          400.00      Dagenham    Nephrostogram
Q1          548.00      Dagenham    Other
Q1          683.00      Dagenham    Urinary
Q1          360.00      East Ham    Angiogram
Q1          289.00      East Ham    Biopsy
Q1          360.00      Ilford      Angiogram
Q1          508.00      Ilford      Angioplasty
Q1          289.00      Ilford      Biopsy
Q1          393.00      Ilford      Nephrostogram
Q1          548.00      Ilford      Other
Q1          386.00      Ilford      Urinary
Q1          360.00      Romford     Angiogram
Q1          508.00      Romford     Angioplasty
Q2          452.00      Barking     Drainage
Q2          360.00      Ilford      Angiogram
Q2          508.00      Ilford      Angioplasty
Q2          508.00      Romford     Angioplasty
```

The above output shows the result according to the scenario as described before. Now I will formulate the cube that shows this data in form of a cube. The cube may look like this:

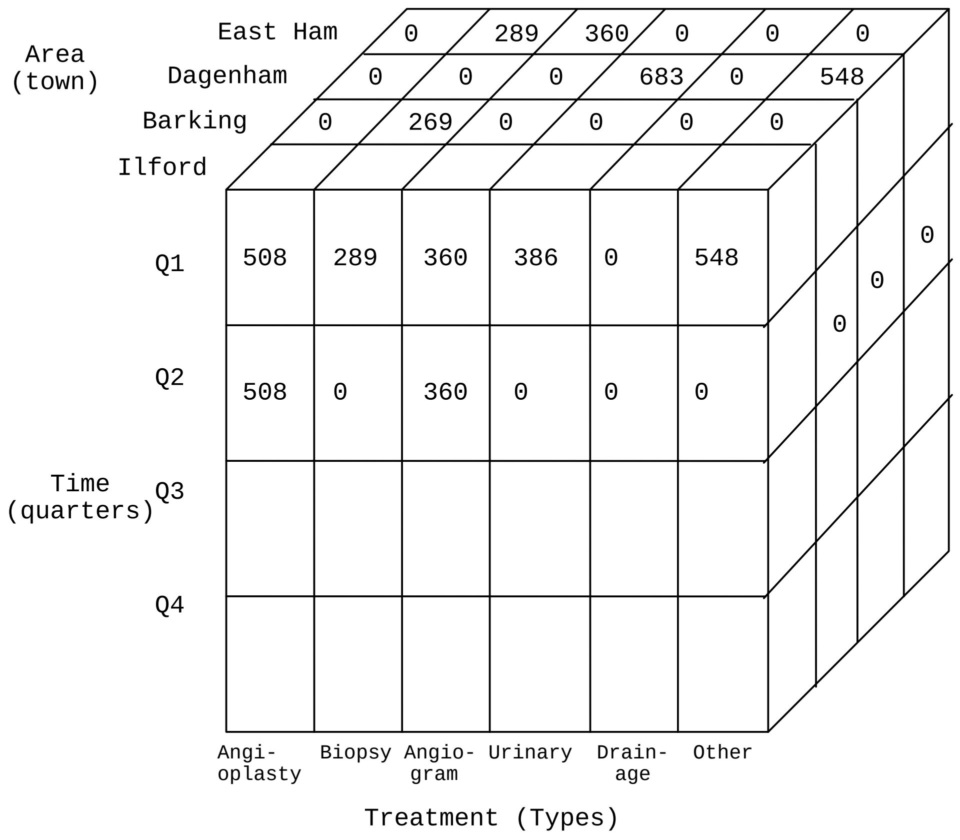
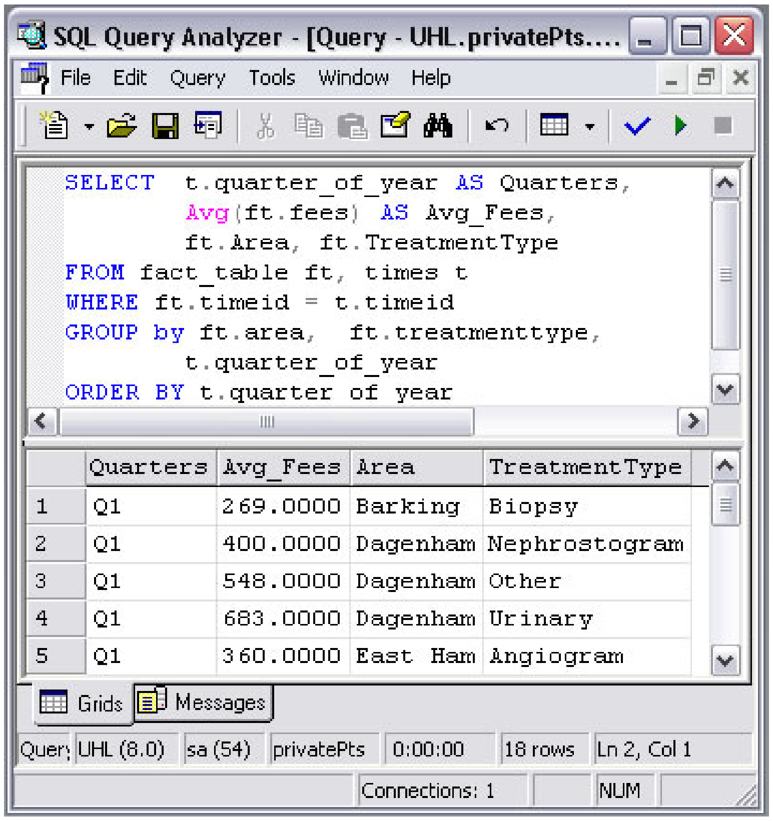


Fig 09: Base Cube showing Average Fees


SQL Query Analyzer - [Query - UHL.privatePts....
File Edit Query Tools Window Help
Grids
Messages
Query UHL (8.0) sa (54) privatePts 0:00:00 18 rows Ln 2, Col 1
Connections: 1
NUM

```
SELECT  t.quarter_of_year AS Quarters,
        Avg(ft.fees)  AS Avg_Fees,
        ft.Area,  ft.TreatmentType
FROM fact_table ft,  times t
WHERE ft.timeid = t.timeid
GROUP by ft.area,   ft.treatmenttype,
        t.quarter_of_year
ORDER BY t.quarter of year
```

	Quarters	Avg_Fees	Area	TreatmentType
1	Q1	269.0000	Barking	Biopsy
2	Q1	400.0000	Dagenham	Nephrostogram
3	Q1	548.0000	Dagenham	Other
4	Q1	683.0000	Dagenham	Urinary
5	Q1	360.0000	East Ham	Angiogram

Fig 10: Screenshot for Base Cube query

3.2 ROLAP Operations

ROLAP stands for Relational On-Line Analytical Processing. It is a product that provides multidimensional analysis of data, aggregates and metadata stored in an RDBMS. The multidimensional processing may be done within the RDBMS, a mid-tier server or the client. A 'merchant' ROLAP is one from an independent vendor which can work with any standard RDBMS [NP03].

Based on the above cube and query, following operations can be performed.

3.2.1 Drill-Down Operation

In drill-down, we navigate from less detailed data to more detailed data [HK01]. In my scenario I have hierarchy on time as day < month < quarters < year, so on base cube, I can move from quarters to months to each day records for more detail analysis.

Input:

```
SELECT t.month_of_year AS Months,
      Avg(ft.fees) AS Avg_Fees,
      ft.Area, ft.TreatmentType
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid
GROUP by ft.area,  ft.treatmenttype, t.month_of_year
ORDER BY t.month_of_year
```

Output:

```
Months       Avg_Fees    Area        TreatmentType
------       --------    -------     --------------
1            360.00      East Ham    Angiogram
1            289.00      East Ham    Biopsy
1            360.00      Romford     Angiogram
2            782.00      Dagenham    Urinary
2            289.00      East Ham    Biopsy
2            360.00      Ilford      Angiogram
2            508.00      Ilford      Angioplasty
2            508.00      Romford     Angioplasty
3            269.00      Barking     Biopsy
3            400.00      Dagenham    Nephrostogram
3            548.00      Dagenham    Other
3            650.00      Dagenham    Urinary
3            360.00      Ilford      Angiogram
3            508.00      Ilford      Angioplasty
3            289.00      Ilford      Biopsy
3            393.00      Ilford      Nephrostogram
3            548.00      Ilford      Other
3            386.00      Ilford      Urinary
3            360.00      Romford     Angiogram
4            360.00      Ilford      Angiogram
4            508.00      Ilford      Angioplasty
4            508.00      Romford     Angioplasty
5            452.00      Barking     Drainage
```

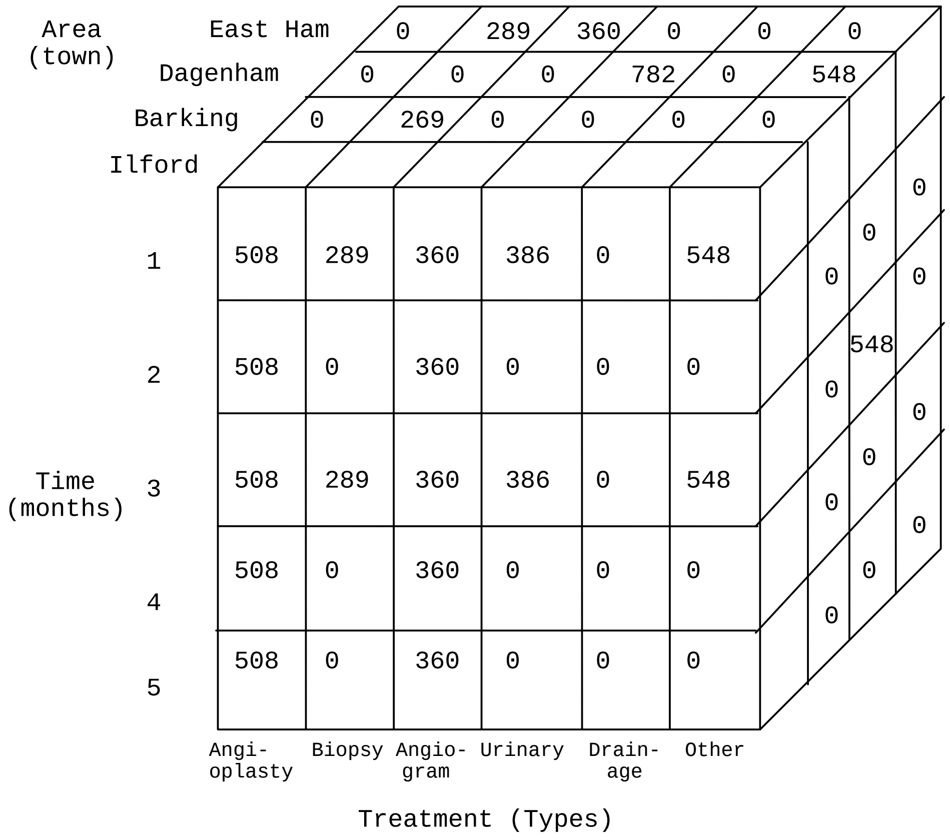
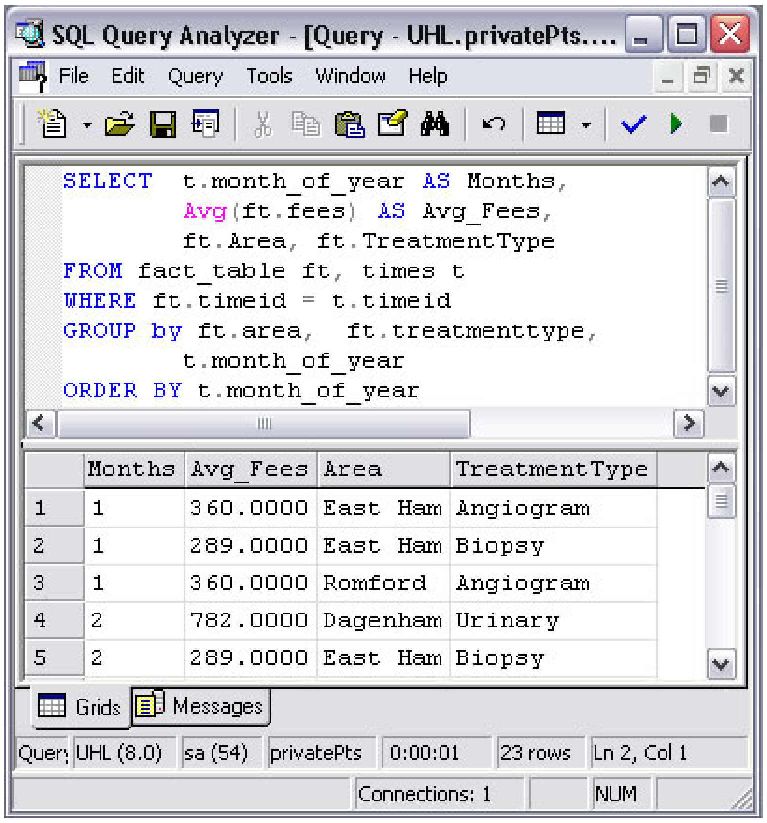


Fig 11: Drill-down operation data cube


SQL Query Analyzer - [Query - UHL.privatePts....
File Edit Query Tools Window Help
Grids
Messages
Quer UHL (8.0) sa (54) privatePts 0:00:01 23 rows Ln 2, Col 1
Connections: 1
NUM

```
SELECT  t.month_of_year AS Months,
        Avg(ft.fees) AS Avg_Fees,
        ft.Area, ft.TreatmentType
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid
GROUP by ft.area,  ft.treatmenttype,
        t.month_of_year
ORDER BY t.month_of_year
```

	Months	Avg_Fees	Area	TreatmentType
1	1	360.0000	East Ham	Angiogram
2	1	289.0000	East Ham	Biopsy
3	1	360.0000	Romford	Angiogram
4	2	782.0000	Dagenham	Urinary
5	2	289.0000	East Ham	Biopsy

Fig 12: Screenshot for Drill-down operation

3.2.2 Slice Operation

This operation performs a selection on one dimension of the given cube, resulting in a sub cube [HK01].

Input:

```
SELECT t.quarter_of_year AS Quarters,
       Avg(ft.fees) AS Avg_Fees,
       ft.Area, ft.TreatmentType
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND t.quarter_of_year = 'Q2'
GROUP BY ft.area, ft.treatmenttype, t.quarter_of_year
```

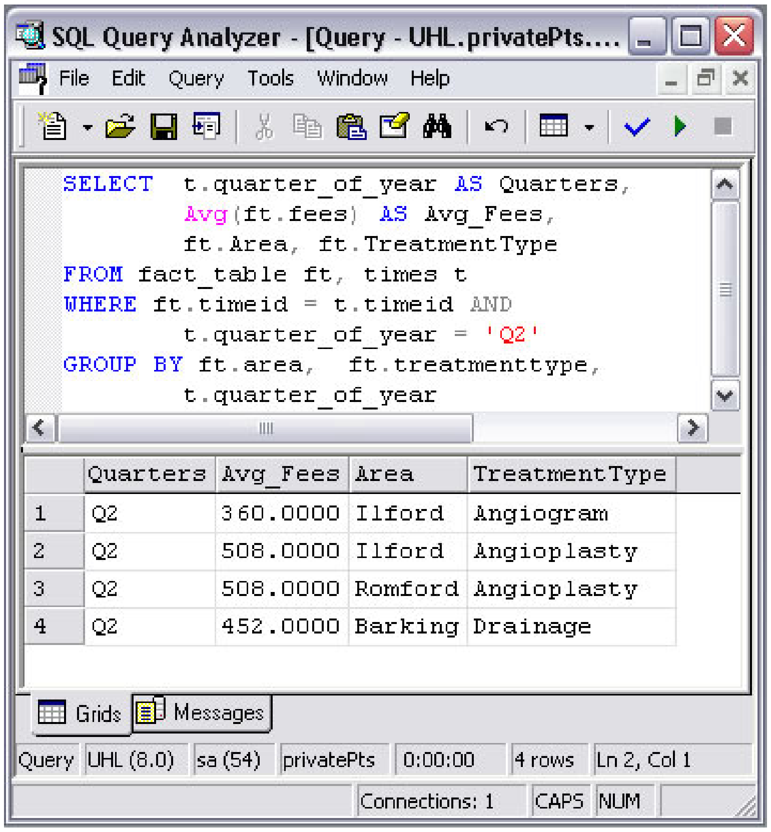
Output:

```
Quarter     Avg_Fees    Area        TreatmentType
-------     --------    -------     -------------
Q2          452.00      Barking     Drainage
Q2          360.00      Ilford      Angiogram
Q2          508.00      Ilford      Angioplasty
Q2          508.00      Romford     Angioplasty
```

Area (town)		Angi-oplasty	Biopsy	Angio-gram	Urinary	Drain-age	Other
	Ilford	508	0	360	0	0	0
	Barking	508	0	360	0	452	0
	Dagenham	0	0	0	0	0	0
	East Ham	0	0	0	0	0	0

Treatment (Types)

Fig 13: Slice Operation Data for Quarter = 'Q2'


SQL Query Analyzer - [Query - UHL.privatePts....
File Edit Query Tools Window Help
Grids Messages
Query UHL (8.0) sa (54) privatePts 0:00:00 4 rows Ln 2, Col 1
Connections: 1 CAPS NUM

```
SELECT  t.quarter_of_year AS Quarters,
        Avg(ft.fees) AS Avg_Fees,
        ft.Area, ft.TreatmentType
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND
        t.quarter_of_year = 'Q2'
GROUP BY ft.area,  ft.treatmenttype,
        t.quarter_of_year
```

	Quarters	Avg_Fees	Area	TreatmentType
1	Q2	360.0000	Ilford	Angiogram
2	Q2	508.0000	Ilford	Angioplasty
3	Q2	508.0000	Romford	Angioplasty
4	Q2	452.0000	Barking	Drainage

Fig 14: Screenshot of Slice Operation

4. BITMAP INDEXING

4.1 What is Bitmap Indexing?

It is a method in OLAP products that allows quick searching in data cubes. In bitmap index for a given attribute, there is a distinct bit vector, *Bv*, for each value *v* in the domain of the attribute. If the domain of the given attribute consists of *n* values, then *n* bits are needed for each entry in the bitmap index. If the attribute has the value *v* for a given row in the data table, then the bit representing that value is set to 1 in the corresponding row of the bitmap index. All other bits for that row are set to 0 [HK01].

4.2 Bitmap Indexing and PDWA

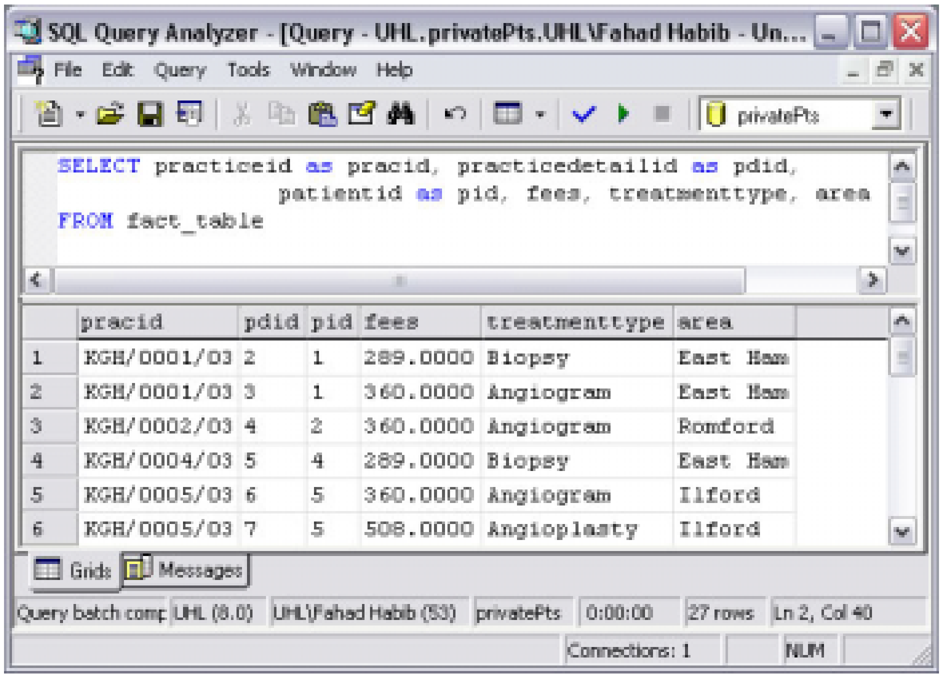
For our database, in `PracticeDetails` we have `TreatmentType` which represent the treatments performed on patients during past times. If we use this attribute for bitmap indexing then we will come across with the following table:

Input:

```
SELECT practiceid as pracid, practicedetailid as pdid,
            patientid as pid, fees, treatmenttype, area
FROM fact_table
```

Output:

```
pracid            pdid  pid   fees        treatmenttype     area
---------         ----- ----  --------    -------------     -----
KGH/0001/03       2     1     289.00      Biopsy            East Ham
KGH/0001/03       3     1     360.00      Angiogram         East Ham
KGH/0002/03       4     2     360.00      Angiogram         Romford
KGH/0003/03       11    3     782.00      Urinary           Dagenham
KGH/0004/03       5     4     289.00      Biopsy            East Ham
KGH/0005/03       6     5     360.00      Angiogram         Ilford
KGH/0005/03       7     5     508.00      Angioplasty       Ilford
KGH/0006/03       8     2     508.00      Angioplasty       Romford
KGH/0007/03       9     6     289.00      Biopsy            Ilford
KGH/0008/03       10    7     249.00      Biopsy            Barking
KGH/0009/03       12    8     782.00      Urinary           Dagenham
KGH/0009/03       13    8     386.00      Urinary           Dagenham
KGH/0010/03       14    9     386.00      Urinary           Ilford
KGH/0011/03       15    10    289.00      Biopsy            Barking
KGH/0012/03       16    9     548.00      Other             Ilford
KGH/0012/03       17    9     386.00      Nephrostogram     Ilford
KGH/0012/03       18    9     400.00      Nephrostogram     Ilford
KGH/0013/03       19    11    782.00      Urinary           Dagenham
KGH/0013/03       20    11    400.00      Nephrostogram     Dagenham
KGH/0013/03       21    11    548.00      Other             Dagenham
KGH/0014/03       22    12    360.00      Angiogram         Ilford
KGH/0014/03       23    12    508.00      Angioplasty       Ilford
KGH/0015/03       24    13    360.00      Angiogram         Romford
KGH/0016/03       25    13    508.00      Angioplasty       Romford
KGH/0017/03       26    14    360.00      Angiogram         Ilford
KGH/0017/03       27    14    508.00      Angioplasty       Ilford
KGH/0018/03       28    15    452.00      Drainage          Barking
```


SQL Query Analyzer - [Query - UHL.privatePts.UHL\Fahad Habib - Un...
File Edit Query Tools Window Help
privatePts
Grids Messages
Query batch comp UHL (8.0) UHL\Fahad Habib (53) privatePts 0:00:00 27 rows Ln 2, Col 40
Connections: 1 NUM

```
SELECT practiceid as pracid, practicedetailid as pdid,
            patientid as pid, fees, treatmenttype, area
FROM fact_table
```

```
   pracid        pdid pid fees     treatmenttype area
1  KGH/0001/03   2    1   289.0000 Biopsy        East Ham
2  KGH/0001/03   3    1   360.0000 Angiogram     East Ham
3  KGH/0002/03   4    2   360.0000 Angiogram     Romford
4  KGH/0004/03   5    4   289.0000 Biopsy        East Ham
5  KGH/0005/03   6    5   360.0000 Angiogram     Ilford
6  KGH/0005/03   7    5   508.0000 Angioplasty   Ilford
```

Fig 15: Fact table for Bitmap Indexing

4.2.1 Bitmap Indexing Table

For each practice record e.g. KGH/0001/03, we have one or more type of treatment type. There are total 18 rows in the Practice table, which have 27 corresponding rows in practicedetails table. Therefore we have 27 rows in fact_table, consisting of all records. Following table indicates treatment type appearing in each Practice.

	Angiogram	Angioplasty	Biopsy	Drainage	Nephrostogram	Urinary	Other
1	1	0	1	0	0	0	0
2	1	0	0	0	0	0	0
3	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0
5	1	1	0	0	0	0	0
6	0	1	0	0	0	0	0
7	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
9	0	0	1	0	0	1	0
10	0	0	0	0	0	1	0
11	0	0	1	0	0	0	0
12	0	0	0	0	1	0	1
13	0	0	0	0	1	1	0
14	1	1	0	0	0	0	0
15	1	0	0	0	0	0	1
16	0	1	0	0	1	0	0
17	1	1	0	0	1	0	0
18	0	0	0	0	0	1	0

Fig 16: Bitmap Index Table

4.3 Advantages

Some of the advantages of the bitmap index are as follows [VG99]:

- It is simple to represent and consume less disk space as compared to other techniques.
- It is quick and CPU-efficient as it uses bitwise operators.
- It is easy to update bitmap index table after addition of new data.
- It is suitable for data warehouses where we have low-cardinality columns.
- It can be used as an alternate method of the row ids representation.

4.4 Disadvantages

Some of the disadvantages of the bitmap index are [VG99]:

- It creates large number of spare/unused data that's never needed.
- It is not suitable for warehouses with high cardinality columns.
- More query processing time and space needed to build and answer queries involving high cardinality columns.

5. HIGH LEVEL LANGUAGE IMPLEMENTATION

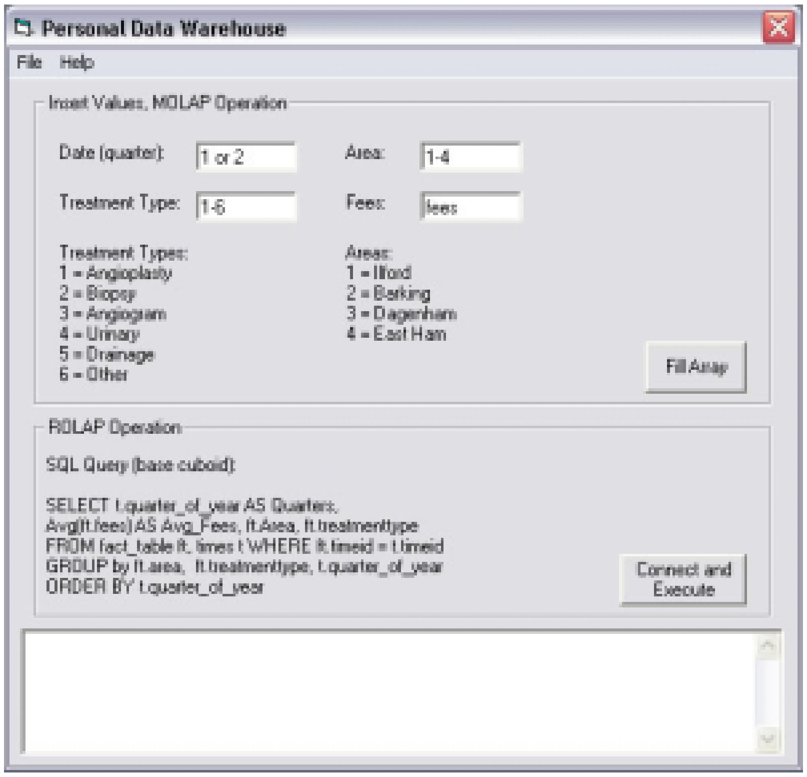
5.1 What is MOLAP?

MOLAP (multidimensional online analytical processing) is online analytical processing (OLAP) that indexes directly into a multidimensional database. In general, an OLAP application treats data multidimensionally; the user is able to view different aspects or facets of data aggregates such as sales by time, geography, and product model. If the data is stored in a relational data base, it can be viewed multidimensionally, but only by successively accessing and processing a table for each dimension or aspect of a data aggregate [TT03].

5.2 Front End Application

The application to demonstrate multidimensional array for MOLAP operation is designed using Visual Basic 6. The simple interface of the application requires user input for the array to be filled and shows data in the form that resembles with the ROLAP operations performed using SQL queries.

It also shows the results we get with our base cuboid. The application establishes a link with our SQL Server database, fetch data according to our base cube query and display results in a textbox. The interface of the application is as follows:



Date (quarter): 1 or 2
Area: 1-4
Treatment Type: 1-6
Fees: fees

Treatment Types:
1 = Angioplasty
2 = Biopsy
3 = Angiogram
4 = Urinary
5 = Drainage
6 = Other

Areas:
1 = Ilford
2 = Barking
3 = Dagenham
4 = East Ham

SQL Query (base cuboid):

```
SELECT t.quarter_of_year AS Quarters,
Avg(ft.fees) AS Avg_Fees, ft.Area, ft.treatmenttype
FROM fact_table ft, times t WHERE ft.timeid = t.timeid
GROUP by ft.area,  ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

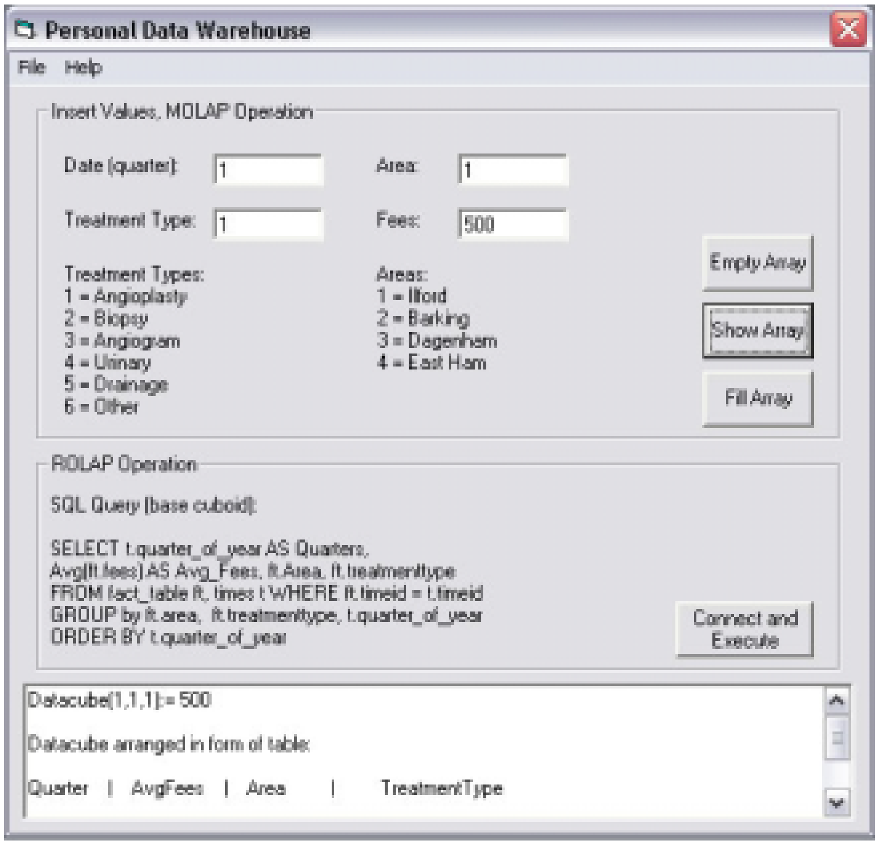
Fig 17: Front-end Application screenshot

5.3 MOLAP Operation

The application requires user interaction by asking for the values for multidimensional array. The array accepts the values within certain limits so that I can demonstrate according to my scenario with some values.

After getting desired input values, the data will be stored in a 3-dimensional array of fixed length. The length of the array is set according to the scenario and base cube requirements. We can add as many values as we want. All relevant data will be updated according to the values provided.

After inserting values, we can view the contents of the multidimensional array in the form of an array data table. The array data table resembles with the actual base cube results as in ROLAP operation.



Insert Values, MOLAP Operation

Date (quarter): 1
Area: 1
Treatment Type: 1
Fees: 500

Treatment Types:
1 = Angioplasty
2 = Biopsy
3 = Angiogram
4 = Urinary
5 = Drainage
6 = Other

Areas:
1 = Ilford
2 = Barking
3 = Dagenham
4 = East Ham

ROLAP Operation

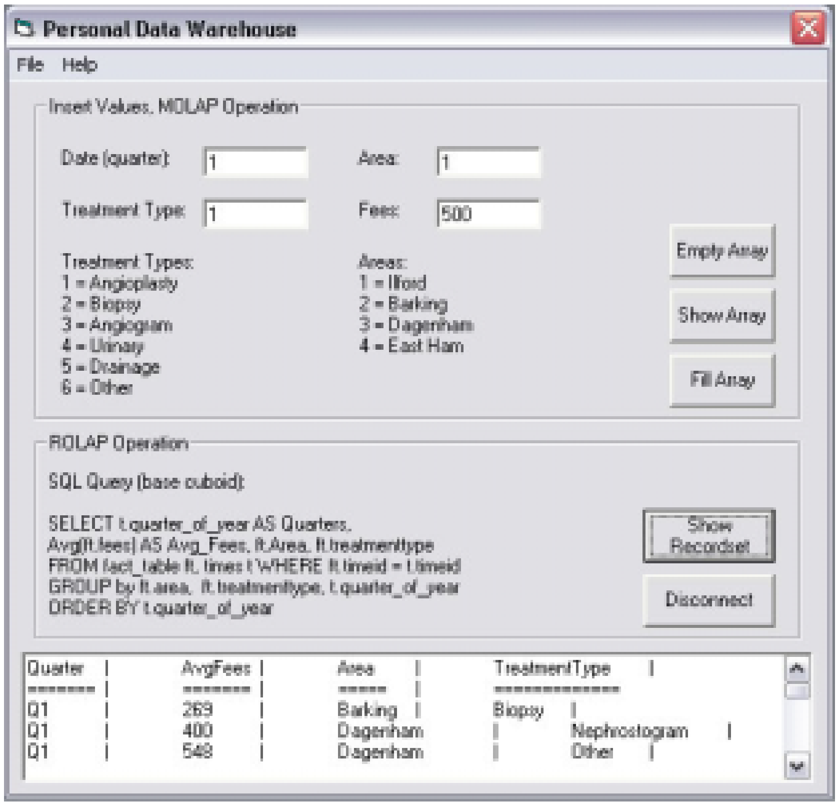
SQL Query (base cuboid):

```
SELECT t.quarter_of_year AS Quarters,
Avg(ft.fees) AS Avg_Fees, ft.Area, ft.treatmenttype
FROM fact_table ft, times t WHERE ft.timeid = t.timeid
GROUP by ft.area, ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

```
Datacube[1,1,1]:= 500

Datacube arranged in form of table:

Quarter  |  AvgFees  |  Area      |      TreatmentType
```

Fig 18: MOLAP Operation, inserting and retrieving values

To demonstrate the resemblance with actual base cube, we can establish link with the actual database and fetch the base cuboid data. After fetching data, the application will show values in the textbox in the form of a table. The information retrieved resembles with the multidimensional array values we just created.



Insert Values, MOLAP Operation

Date (quarter): 1
Area: 1
Treatment Type: 1
Fees: 500

Treatment Types:
1 = Angioplasty
2 = Biopsy
3 = Angiogram
4 = Urinary
5 = Drainage
6 = Other

Areas:
1 = Ilford
2 = Barking
3 = Dagenham
4 = East Ham

ROLAP Operation

SQL Query (base cuboid):

```
SELECT t.quarter_of_year AS Quarters,
Avg(ft.fees) AS Avg_Fees, ft.Area, ft.treatmenttype
FROM fact_table ft, times t WHERE ft.timeid = t.timeid
GROUP by ft.area,  ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

```
Quarter  |   AvgFees  |   Area     |   TreatmentType   |
======== |   ======== |   =====    |   ==============
Q1       |   269      |   Barking  |   Biopsy   |
Q1       |   400      |   Dagenham |        Nephrostogram   |
Q1       |   548      |   Dagenham |        Other   |
```

Fig 19: MOLAP vs ROLAP, Getting data from database

After performing required operations on our multidimensional array, we can empty and fill it again with new data to test again. Similarly, we can disconnect with our SQL server database and free the system resources.

5.4 Comparison between ROLAP and MOLAP

Following points can be noticed between two operations [TT03].

- MOLAP processes data that is already stored in a multidimensional array whereas in case of ROLAP we have to fetch data from database [TT03].
- MOLAP reflects all possible combinations of data, each in a cell that can be accessed directly whereas in case of ROLAP we have to formulate our data according to our required to create potential arrangement [TT03].

For these reasons, MOLAP is, for most uses, faster and more user-responsive than relational online analytical processing (ROLAP) [TT03].

There is also hybrid OLAP (HOLAP), which combines some features from both ROLAP and MOLAP [TT03].

5.5 Conclusion

In my opinion, the MOLAP operation is more difficult than ROLAP operation. The reason behind this the uncertain size of our multidimensional array. For small database, we can perform MOLAP operation easily, but for large and complex databases, MOLAP operation looks difficult to implement.

On the other hand ROLAP operation looks much simpler and easier to understand and implemental on databases of all sizes. But according to experts [TT03], as mentioned earlier, the MOLAP is faster than ROLAP.

6. SPARSE MATRIX PROBLEM

6.1 What is Sparse Matrix?

It is a matrix with small number of non-zero values [JK02]. Thus, only a small proportion of potential data cells actually occupied in a multidimensional structure [NP03].

6.2 Example

For my PDWA, following cube shows a sparse data cube:

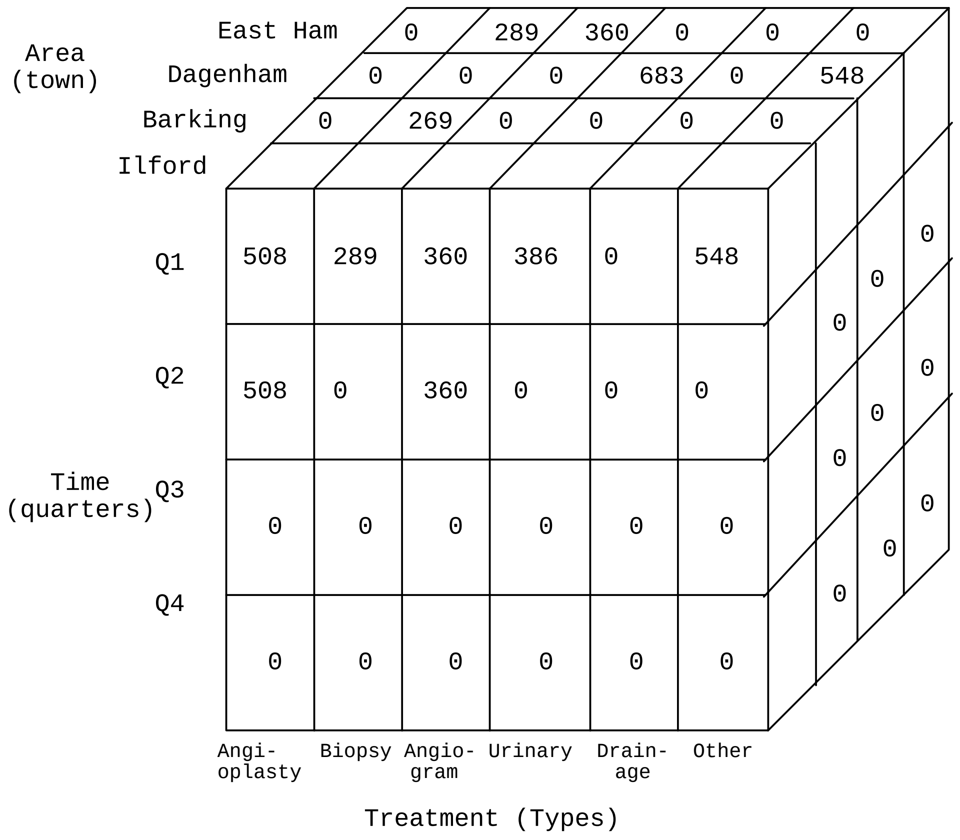


Fig 20: Sparse Data Cube

It is obvious from the above example, that we have large amount of useless data in the cube. There are more zero values than non-zero ones, hence making our cube inefficient and sparse.

Since this PDWA is not big as compared to the actual, real-time scenario, but still it creates problem of useless values for us. So for real life situations, this problem will increase many times as we have large amount of data. Hence it is clear that we will get huge sparse data cubes for actual data warehouse applications.

6.3 Solution

To overcome this problem, we may implement linked lists. In linked lists we allocate values dynamically, hence more space efficient than multidimensional arrays. With arrays, we have to mention size before assigning values, therefore it consume huge amount of disk space and also reduce the efficiency of data cube by adding useless data.

On other hand with linked lists, we can add only those values which are non-zero and useful for us, hence eliminating the problem of sparse data cube.

6.4 Alternative Solution

According to Jong Kim [JK02], the pointers can be used to overcome the sparse matrix problem in multidimensional OLAP. Following example shows Kim's view about sparse matrix implementation.

6.4.1 Example

Suppose we have [JK02]:

- 1000 X 1000 matrix with 2000 entries
- 2-dimensional array implementation
    - 1000 * 1000 * 8 byte = 8M byte
    - Matrix inversion requires temporary matrices
- Pointer implementation
    - 2000 * (4(row pointer) + 4(column pointer) + 2(row number) + 2(column number) + 8(value)) = 40Kbyte

This example illustrates how Kim overcome the problem of sparse matrix using pointers and reduces the large useless space to small useful data.

7. THE 4D CUBE

For showing 4D cube according to my scenario, I will add another measure age in my base cuboid measures. By adding this new measure I can generate different cubes according to my base cube criteria, and will show values for different ages or age ranges.

It will help me in creating a series of 3D cubes and enable me to demonstrate 4D cube visualization for my PDWA.

7.1 Series of 3D Cubes

For generating series of 3D cubes to represent a 4D cube, following queries are used:

Input:

Query1, Age <= 35:

```
SELECT t.quarter_of_year AS Quarters,
      Avg(ft.fees) AS Avg_Fees, '<=35' AS Age_Range,
      ft.area, ft.treatmenttype
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND ft.age <=35
GROUP BY ft.area, ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

Query2, Age >35 and Age <= 65:

```
SELECT  t.quarter_of_year AS Quarters,
      Avg(ft.fees) AS Avg_Fees, '>35 and <=65' AS Age_Range,
      ft.area, ft.treatmenttype
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND (ft.age > 35 AND ft.age <= 65)
GROUP BY ft.area, ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

Query3, Age > 65:

```
SELECT  t.quarter_of_year AS Quarters,
      Avg(ft.fees) AS Avg_Fees, '>65' AS Age_Range,
      ft.area, ft.treatmenttype
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND ft.age >65
GROUP BY ft.area, ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year
```

Output:

Query1, Age <= 35:

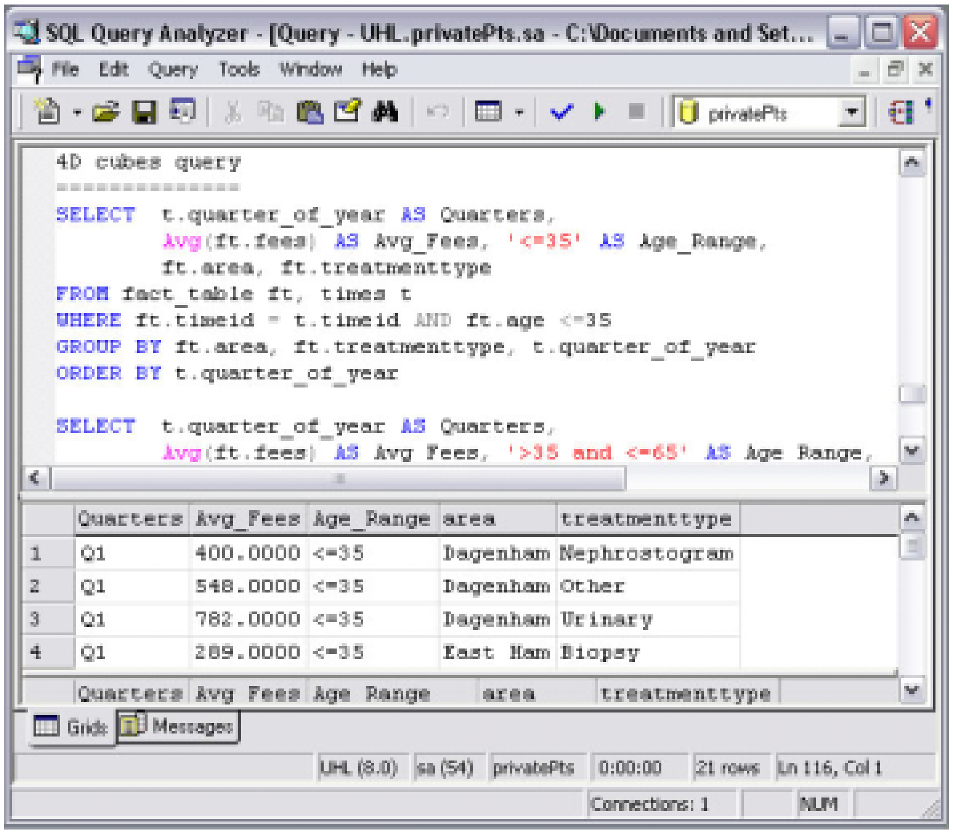
```
Quarters    Avg_Fees    Age   Town        TreatmentType
--------    ---------   ----  ------      -------------
Q1          400.00      <=35  Dagenham    Nephrostogram
Q1          548.00      <=35  Dagenham    Other
Q1          782.00      <=35  Dagenham    Urinary
Q1          289.00      <=35  East Ham    Biopsy
```

Query2, Age >35 and Age <= 65:

```
Quarters     Avg_Fees     Age                Town         TreatmentType
--------     ---------    ------------       ------       -------------
Q1           269.00       >35 and <=65       Barking      Biopsy
Q1           782.00       >35 and <=65       Dagenham     Urinary
Q1           360.00       >35 and <=65       East Ham     Angiogram
Q1           289.00       >35 and <=65       East Ham     Biopsy
Q1           393.00       >35 and <=65       Ilford       Nephrostogram
Q1           548.00       >35 and <=65       Ilford       Other
Q1           386.00       >35 and <=65       Ilford       Urinary
Q2           360.00       >35 and <=65       Ilford       Angiogram
Q2           508.00       >35 and <=65       Ilford       Angioplasty
```

Query3, Age > 65:

```
Quarters     Avg_Fees     Age    Town         TreatmentType
--------     ---------    ----   ------       -------------
Q1           584.00       >65    Dagenham     Urinary
Q1           360.00       >65    Ilford       Angiogram
Q1           508.00       >65    Ilford       Angioplasty
Q1           289.00       >65    Ilford       Biopsy
Q1           360.00       >65    Romford      Angiogram
Q1           508.00       >65    Romford      Angioplasty
Q2           452.00       >65    Barking      Drainage
Q2           508.00       >65    Romford      Angioplasty
```


SQL Query Analyzer - [Query - UHL.privatePts.sa - C:\Documents and Set...
File Edit Query Tools Window Help
privatePts
Grids
Messages
UHL (8.0)
sa (54)
privatePts
0:00:00
21 rows
Ln 116, Col 1
Connections: 1
NUM

```
4D cubes query
===============
SELECT  t.quarter_of_year AS Quarters,
        Avg(ft.fees) AS Avg_Fees, '<=35' AS Age_Range,
        ft.area, ft.treatmenttype
FROM fact_table ft, times t
WHERE ft.timeid = t.timeid AND ft.age <=35
GROUP BY ft.area, ft.treatmenttype, t.quarter_of_year
ORDER BY t.quarter_of_year

SELECT  t.quarter_of_year AS Quarters,
        Avg(ft.fees) AS Avg Fees, '>35 and <=65' AS Age Range,
```

	Quarters	Avg_Fees	Age_Range	area	treatmenttype
1	Q1	400.0000	<=35	Dagenham	Nephrostogram
2	Q1	548.0000	<=35	Dagenham	Other
3	Q1	782.0000	<=35	Dagenham	Urinary
4	Q1	289.0000	<=35	East Ham	Biopsy

Quarters	Avg Fees	Age Range	area	treatmenttype

Fig 21: 4D Cube query screenshot

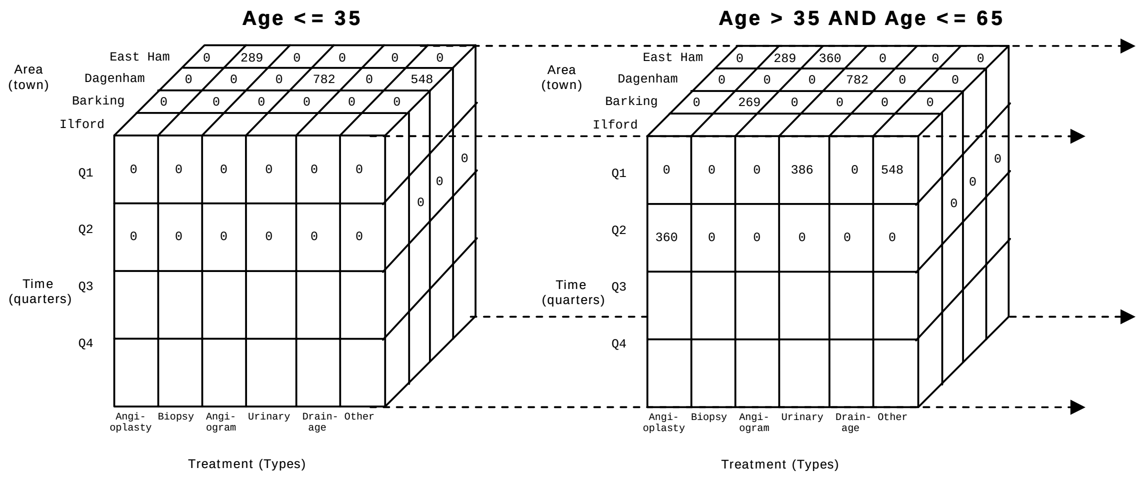


Fig 22: Series of 3D cubes

7.2 Conclusion

Thus, it is possible to show the 4D cube using a series of 3D cubes to overcome the tricky problem of showing data in the form of 4D cube.

8. REFERENCES AND BIBLIOGRAPHY

In alphabetical ordering:

[HK01] Jiawei Han and Micheline Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001

[JK02] Jong Kim, *Pointer Usage Example, Dept. of Computer Science & Engineering, Pohang University of Science & Technology* (URL: http://www.postech.ac.kr/class/cs103-2/CS103-09.ppt, Visited on: 15 December 2003 14:23)

[NP03] Nigel Pendse, *The OLAP Report Glossary* (URL: http://www.olapreport.com/glossary.htm, Visited on: 15 December 2003 14:27)

[TT03] *MOLAP Definition and features, SearchDatabase site on TechTarget.com* (URL: http://searchdatabase.techtarget.com/sDefinition/0,290660,sid13_gci882493,00.html, Visited on: 15 December, 2003 at 14:35)

[VG99] *Sirirut Vanichayobon & Le Gruenwald, Indexing Techniques for Data Warehouses' Queries, School of Computer Science, The University of Oklahoma* (URL: http://www.cs.ou.edu/~database/documents/vg99.pdf, Visited on: 13 Dec, 2003 at 17:25)