

Health Insurance Claims Report

Krishnavamsi Sanisetty



Contents

1.Executive Summary

2.Project Evaluation

3.Database Design

4.Query Writing

5.Performance Tuning

6.Data Visualizations

7.Summary

8.References






1.Executive Summary

The Health Insurance business arena revolves around comprehending member claims and provider policies in order to accomplish and approve claim amounts to the members by scrutinizing the technicalities in the insurance plans. In the United States be it Medicare or Private insurance, overcharging for a particular DRG is always a concern and area worth investing attention. Especially in this unprecedented time of a global pandemic analyzing medical insurance claims is vital for assuring providers, members and insurance companies that there are no disparities among the customer claim payments.

With this motivation we planned to design a real world health insurance claims database which involves information pertaining to the members, providers, claims, address, claim status, claim payment and coverage. We formulated our own dataset by comprehending the business domain knowledge in healthcare business. We generated our own dataset which had 7 tables (described in the further sections). We formulated a relational database using MySQL Workbench and populated all 7 tables with approximately 100 rows each table.

In this project our main goal is to extract reports for the healthcare insurance officials which can enable them to analyse and present insights to the managerial officials. Using our prior database experience and hands on knowledge gained from the course we first designed Database Architecture By formulating an Entity Relationship diagram. After populating our database, we developed multiple queries which involved joins, sorting operations, calculated fields and much more to extract insightful information from the database that could prove crucial for analyzing insurance claims and approved claim payments. In order to optimize our query performance we utilized power of indexing, partitioning, stored procedures and views in our project. This enabled us to perform performance tuning by optimizing our query performance. In addition to generating queries we coupled tableau with our Healthcare Insurance Claims database and developed interactive visualizations.

This project enabled us to understand the entire Database Design process from scratch resulting into a development of a real world database and data visualization system for the health insurance companies.



Project Evaluation

Topic Area	Description	Points
Database Design	We generated our own dataset which contained 7 tables. We worked on designing the ER diagram which depicted the relationship among the tables and finally we populated it to the database using MySQL WorkBench	25
Query Writing	We formulated 4 questions which we thought would be beneficial to gain insights regarding claims, payments, status and coverage plans. We then formulated our queries using joins, where clause and between clause.	30
Performance Tuning	Query optimization is what we really thought about for each query. We used indexing to optimize traversal and lower the overhead. As our queries included multiple inner join operations we created a virtual table view in order to optimize storage and lower the overhead. Moreover, we generated stored procedures to enhance performance, scalability and robustness.	30
Other Topics	To analyse our data, we connected MySQL to Tableau and designed interactive visualizations which enhanced effectiveness of our insightful reports.	15

2.Database Design

2.1 Dataset/Data Tables

As per our research on Healthcare Insurance Business we designed 7 tables described as per below snapshots and populated each table with approximately 100 rows. 1.

Provider

	provider_id	provider_first_name	provider_last_name	degree	network	claim_id	practice_name	address_id	gender
▶	7001	Cadman	Delaney	MS	Out Network	900	ADVENT HEATH	3457	Male
	7002	Brandon	Nieves	MD	In Network	901	SMILE DENTAL	4557	Male

2. Member

	member_id	member_first_name	member_last_name	address_id	gender	member_dob	claim_id	coverage_id
▶	4000	Kadeem	Bush	3457	Not Answered	13-03-1993	900	1000001
	4001	Dana	Guerrero	4557	Male	26-02-1974	901	1000002

3. Claim

	claim_id	status_id	date_of_service	received_date	add_by
▶	900	5001	Oct 3, 2019	Dec 31, 2019	3
	901	5002	Jul 24, 2019	Oct 27, 2019	3

4. Coverage

	coverage_id	member_id	coverage_name	effective_date	term_date
▶	1000001	4000	SILVER	03-Jul-19	12/31/19

5. Claim_Payment

	claim_payment_id	billed_amount	approved_amount	copay_amount	coinsurance_amount	deductible_amount	net_payment	claim_id
▶	20090	4803	2542	3	7	0	2552	900
	20091	4404	2811	5	8	0	2824	901

6. Status

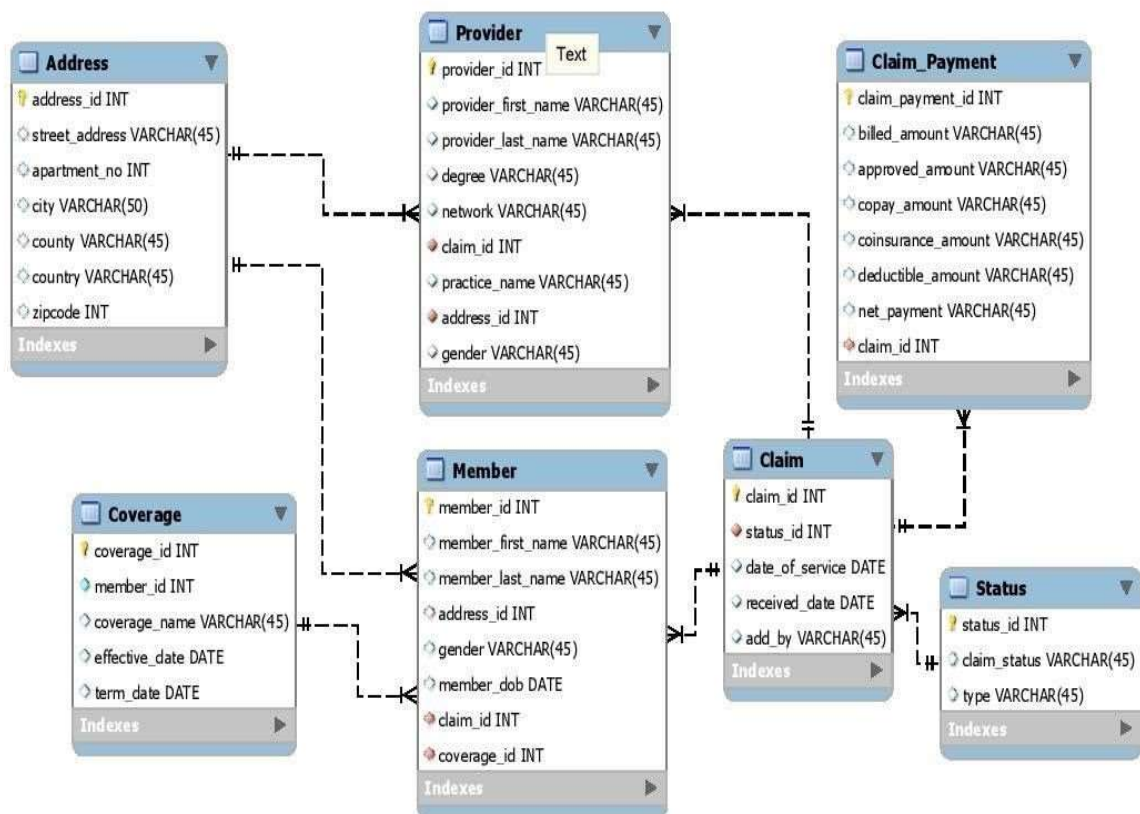
	status_id	claim_status	type
▶	5001	in progress	d
	5002	paid	d

7. Address

	address_id	street_address	apartment_no	city	county	country	zipcode
▶	1052	428-8294 Lobortis, Rd.	105	Seydişehir	Kon	Nauru	06064
	1073	P.O. Box 122, 6026 Ac Av.	102	Sobral	Ceará	Czech Republic	559162

2.2 Logical Database Design

We designed a Logical Database Architecture by defining tables mentioned above. In addition to creating the tables we formulated and defined relationships between the tables. We first developed an Entity Relationship Diagram which defined tables along with unique keys and relationships. While formulating the ER diagram for the project we addressed normality for reducing redundancy and decrease computational complexity.



2.3 Data Definition Language (DDL)

1. Provider

```
CREATE TABLE `provider` (
  `provider_id` int NOT NULL,
  `provider_first_name` varchar(50) DEFAULT NULL,
  `provider_last_name` varchar(50) DEFAULT NULL,
  `degree` varchar(50) DEFAULT NULL,
  `network` varchar(50) DEFAULT NULL,
  `claim_id` int DEFAULT NULL,
  `practice_name` varchar(50) DEFAULT NULL,
  `address_id` int DEFAULT NULL,
  `gender` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`provider_id`),
  KEY `claim_id_idx` (`claim_id`),
  KEY `address_id_idx` (`address_id`),
  CONSTRAINT `address_id_p` FOREIGN KEY (`address_id`) REFERENCES `address` (`address_id`),
  CONSTRAINT `claim_id_p` FOREIGN KEY (`claim_id`) REFERENCES `claim` (`claim_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

2. Member

```
CREATE TABLE `member` (
  `member_id` int NOT NULL,
  `member_first_name` varchar(50) DEFAULT NULL,
  `member_last_name` varchar(50) DEFAULT NULL,
  `address_id` int NOT NULL,
  `gender` varchar(50) DEFAULT NULL,
  `member_dob` varchar(50) DEFAULT NULL,
  `claim_id` int NOT NULL,
  `coverage_id` int NOT NULL,
  PRIMARY KEY (`member_id`),
  KEY `address_id_idx` (`address_id`),
  KEY `claim_id_idx` (`claim_id`),
  KEY `coverage_id_idx` (`coverage_id`),
  CONSTRAINT `address_id` FOREIGN KEY (`address_id`) REFERENCES `address` (`address_id`),
  CONSTRAINT `claim_id_m` FOREIGN KEY (`claim_id`) REFERENCES `claim` (`claim_id`),
  CONSTRAINT `coverage_id` FOREIGN KEY (`coverage_id`) REFERENCES `coverage` (`coverage_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

3. Status

```
CREATE TABLE `status` (
  `status_id` int NOT NULL,
  `claim_status` varchar(50) DEFAULT NULL,
  `type` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`status_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```


4. Claim

```
CREATE TABLE `claim` (  
  `claim_id` int NOT NULL,  
  `status_id` int DEFAULT NULL,  
  `date_of_service` varchar(50) DEFAULT NULL,  
  `received_date` varchar(50) DEFAULT NULL,  
  `add_by` int DEFAULT NULL,  
  PRIMARY KEY (`claim_id`),  
  UNIQUE KEY `claim_id_UNIQUE` (`claim_id`),  
  KEY `status_id_idx` (`status_id`),  
  CONSTRAINT `status_id` FOREIGN KEY (`status_id`) REFERENCES `status` (`status_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

5. Coverage

```
CREATE TABLE `coverage` (  
  `coverage_id` int NOT NULL,  
  `member_id` int NOT NULL,  
  `coverage_name` varchar(50) DEFAULT NULL,  
  `effective_date` varchar(50) DEFAULT NULL,  
  `term_date` varchar(50) DEFAULT NULL,  
  PRIMARY KEY (`coverage_id`),  
  KEY `member_id_idx` (`member_id`),  
  CONSTRAINT `member_id` FOREIGN KEY (`member_id`) REFERENCES `member` (`member_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

6. Claim_Payment

```
CREATE TABLE `claim_payment` (  
  `claim_payment_id` int NOT NULL,  
  `billed_amount` float DEFAULT NULL,  
  `approved_amount` float DEFAULT NULL,  
  `copay_amount` float DEFAULT NULL,  
  `coinsurance_amount` float DEFAULT NULL,  
  `deductible_amount` float DEFAULT NULL,  
  `net_payment` float DEFAULT NULL,  
  `claim_id` int DEFAULT NULL,  
  PRIMARY KEY (`claim_payment_id`),  
  KEY `claim_id_idx` (`claim_id`),  
  CONSTRAINT `claim_id` FOREIGN KEY (`claim_id`) REFERENCES `claim` (`claim_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

7. Address


```

CREATE TABLE `address` (
  `address_id` int NOT NULL,
  `street_address` varchar(50) DEFAULT NULL,
  `apartment_no` int DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `county` varchar(50) DEFAULT NULL,
  `country` varchar(50) DEFAULT NULL,
  `zipcode` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`address_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

2.3 Primary Key and Foreign Keys

Table Name	Primary Key	Foreign Key
Provider	provider_id	claim_id(claim) address_id(address)
Member	member_id	address_id(address) claim_id(claim) coverage_id(coverage)
Claim	claim_id	status_id(status)
Claim_Payment	claim_payment_id	claim_id(claim)
Coverage	coverage_id	member_id(member)
Status	status_id	
Address	address_id	

3. Query Writing

- 1) Reporting top 20 members along with provider details, billed amount and approved amount associated and having approved_amount greater than \$2200.

Query

```
select m.member_first_name, m.member_last_name, s.claim_status,  
       p.provider_first_name, p.provider_last_name, p.network,  
       p.practice_name, cp.billed_amount, cp.approved_amount, cp.net_payment  
from  
      (((member m inner join claim c on m.claim_id = c.claim_id)  
        inner join status s on s.status_id = c.status_id)  
        inner join provider p on p.claim_id = m.claim_id)  
        inner join claim_payment cp on cp.claim_id = c.claim_id)  
where cp.approved_amount > 2200  
order by cp.approved_amount desc limit 20;
```

Output:

	member_first_name	member_last_name	claim_status	provider_first_name	provider_last_name	network	practice_name	billed_amount	approved_amount	net_payme
▶	Marcia	Farrell	history	Harper	Ochoa	In Network	WELLCARE HOSPITAL	4785	2991	3003
	Plato	Stuart	history	Cade	Conrad	In Network	ADVENT HEATH	4726	2983	2989
	Eve	Spence	paid	Asher	Rodgers	In Network	VIERRA HOSPITAL	4921	2982	2995
	Piper	Weber	deny	Norman	Floyd	Out Network	MEMORIAL HOSPITAL	4812	2979	2982
	Jordan	Hill	paid	Nissim	Ellis	Out Network	ABC HOSPITAL	4330	2977	2988
	Lane	Weiss	in progress	Preston	Dejesus	In Network	ADVENT HEATH	4991	2972	2989
	Irene	Rutledge	in progress	Owen	Montgomery	Out Network	MEMORIAL HOSPITAL	4423	2972	2981
	Jolene	Bridges	pend	Jerome	Henry	In Network	ADVENT HEATH	4108	2942	2952
	Leslie	Mason	in progress	Cain	Buddey	Out Network	ADVENT HEATH	4134	2941	2947
	Josephine	Rhodes	in progress	Jordan	Bernard	In Network	ADVENT HEATH	4458	2934	2948
	Jeremy	Powell	deny	Rahim	Peters	Out Network	ADVENT HEATH	4605	2928	2935
	Teegan	Wolfe	history	Dalton	Bauer	In Network	CONCORDIA HEALTH	4077	2920	2930
	Tashya	Bowen	paid	Kirk	Gross	In Network	SMILE DENTAL	4234	2919	2926
	Mariam	Whitaker	history	Holmes	Hanson	Out Network	SMILE DENTAL	4087	2919	2930
	Kylee	Greer	history	Ross	Small	Out Network	SMILE DENTAL	4183	2894	2907
	Evan	Simon	history	Samson	Stevens	In Network	VIERRA HOSPITAL	4207	2890	2903
	Dana	Langley	pend	Price	Daniel	Out Network	SMILE DENTAL	4000	2880	2899
	Lyle	William	deny	Judah	Hendricks	In Network	MEMORIAL HOSPITAL	4931	2862	2874
	Timon	Richard	in progress	Amal	Richard	Out Network	SMILE DENTAL	4234	2844	2864
	Mia	Mcgee	paid	Allstair	Britt	Out Network	ADVENT HEATH	4816	2843	2853

2) Reporting distinct members (member demographic information) who have their claims in progress.

Query

```

select distinct m.member_first_name, m.member_last_name, m.member_dob, m.gender, m.claim_id,
               s.claim_status, s.type
from
  ((member m inner join claim c on m.claim_id = c.claim_id)
   inner join status s on s.status_id = c.status_id)
where s.claim_status = "in progress"
order by m.member_first_name;

```

Output

	member_first_name	member_last_name	member_dob	gender	claim_id	claim_status	type
▶	Adria	Macias	03-07-2002	Male	940	in progress	d
	Amal	Whitehead	16-02-1992	Female	905	in progress	d
	Chase	Mueller	14-02-1993	Male	935	in progress	d
	Emily	Vincent	29-08-1999	Male	970	in progress	d
	India	Francis	24-10-1970	Not Answered	980	in progress	d
	Irene	Rutledge	04-03-1987	Not Answered	995	in progress	d
	Jaquelyn	Flynn	23-04-1995	Male	915	in progress	d
	Josephine	Rhodes	15-12-1998	Not Answered	950	in progress	d
	Kadeem	Bush	13-03-1993	Not Answered	900	in progress	d
	Keely	Pena	01-03-1980	Female	920	in progress	d
	Lane	Weiss	12-11-1970	Male	955	in progress	d
	Leslie	Mason	06-01-1977	Not Answered	985	in progress	d
	Maxwell	Rocha	16-02-1994	Female	990	in progress	d
	Meghan	Schneider	14-02-1981	Male	960	in progress	d
	Neville	Whitney	08-11-1981	Not Answered	925	in progress	d
	Rajah	Winters	24-06-1991	Not Answered	965	in progress	d
	Rooney	Hoffman	12-12-1985	Not Answered	930	in progress	d
	Shana	Ferguson	02-08-1970	Male	945	in progress	d
	Timon	Richard	13-09-2007	Female	975	in progress	d

3) Finding member subscribed to a coverage plan having address ID between 2000 to 5000

Query:

```
select distinct m.member_first_name, m.member_last_name, cp.coverage_name, m.address_id
from (coverage cp inner join member m on m.member_id = cp.member_id)
where m.address_id between 2000 and 5000;
```

Output:

	member_first_name	member_last_name	coverage_name	address_id
►	Kadeem	Bush	SILVER	3457
	Dana	Guerrero	SILVER	4557
	Gretchen	Browning	PLATINUM	2235
	Calista	Vance	PLATINUM	4366
	Evan	Simon	SILVER	2265
	Amal	Whitehead	BRONZE	2041
	Rhoda	Mckay	GOLD	3286
	Porter	Battle	BRONZE	3960
	Leah	Carrillo	SILVER	4542
	Maris	Santiago	BRONZE	3519
	Yolanda	Bryan	GOLD	2157
	Colton	Wiggins	BRONZE	2061
	Brynne	Nash	SILVER	4806
	Zeus	Matthews	PLATINUM	2583
	Shelby	Chandler	PLATINUM	4067
	Latifah	Goodman	SILVER	3381
	Chester	Powers	SILVER	2107
	Keely	Pena	SILVER	2224
	Tristan

4. Performance Tuning

4.1 Creating Views for Queries with Multiple Join Operations

- Creating View top_20_claim_approved_customers

Views: Views are a virtual table which stores results of the SQL statement. Views optimize storage and decrease computational overhead. Views are apt for those who want to work on the subset of data. Views enhance performance while performing data analysis and also adds significant data security.

The below snapshot shows an example which we tried in our project. We created a virtual table for extracting top 20 customers with approved payment greater than \$2200


```
Use health_insurance_claims_database;
```

```
create view top_20_claim_approved_customers as (select m.member_first_name, m.member_last_name, s.claim_status,
p.provider_first_name, p.provider_last_name, p.network,
p.practice_name, cp.billed_amount, cp.approved_amount, cp.net_payment
from
((((member m inner join claim c on m.claim_id = c.claim_id)
inner join status s on s.status_id = c.status_id)
inner join provider p on p.claim_id = m.claim_id)
inner join claim_payment cp on cp.claim_id = c.claim_id)
where cp.approved_amount > 2200
order by cp.approved_amount desc limit 20);
```

```
select * from top_20_claim_approved_customers;
```

Output

	member_first_name	member_last_name	claim_status	provider_first_name	provider_last_name	network	practice_name	billed_amount	approved_amount	net_payment
▶	Marcia	Farrell	history	Harper	Ochoa	In Network	WELLCARE HOSPITAL	4785	2991	3003
	Plato	Stuart	history	Cade	Conrad	In Network	ADVENT HEATH	4726	2983	2989
	Eve	Spence	paid	Asher	Rodgers	In Network	VIERRA HOSPITAL	4921	2982	2995
	Piper	Weber	deny	Norman	Floyd	Out Network	MEMORIAL HOSPITAL	4812	2979	2982
	Jordan	Hill	paid	Nissim	Ellis	Out Network	ABC HOSPITAL	4330	2977	2988
	Lane	Weiss	in progress	Preston	Dejesus	In Network	ADVENT HEATH	4991	2972	2989
	Irene	Rutledge	in progress	Owen	Montgomery	Out Network	MEMORIAL HOSPITAL	4423	2972	2981
	Jolene	Bridges	pend	Jerome	Henry	In Network	ADVENT HEATH	4108	2942	2952
	Leslie	Mason	in progress	Cain	Buckley	Out Network	ADVENT HEATH	4134	2941	2947
	Josephine	Rhodes	in progress	Jordan	Bernard	In Network	ADVENT HEATH	4458	2934	2948
	Jeremy	Powell	deny	Rahim	Peters	Out Network	ADVENT HEATH	4605	2928	2935
	Teegan	Wolfe	history	Dalton	Bauer	In Network	CONCORDIA HEALTH	4077	2920	2930
	Tashya	Bowen	paid	Kirk	Gross	In Network	SMILE DENTAL	4234	2919	2926
	Mariam	Whitaker	history	Holmes	Hanson	Out Network	SMILE DENTAL	4087	2919	2930
	Kylee	Greer	history	Ross	Small	Out Network	SMILE DENTAL	4183	2894	2907
	Evan	Simon	history	Samson	Stevens	In Network	VIERRA HOSPITAL	4207	2890	2903
	Dana	Langley	pend	Price	Daniel	Out Network	SMILE DENTAL	4000	2880	2899
	Lyle	William	deny	Judah	Hendricks	In Network	MEMORIAL HOSPITAL	4931	2862	2874
	Timon	Richard	in progress	Amal	Richard	Out Network	SMILE DENTAL	4234	2844	2864
	Mia	Mcgee	paid	Allstair	Britt	Out Network	ADVENT HEATH	4816	2843	2853

4.2 Stored procedures for adding a centralized business logic.

- **Stored procedures:** Stored procedures reduce network traffic and can be used to avoid executing several complex queries each time. Utilizing stored procedures we can pass parameters for the query in the form of arguments.

With the goal of tuning our second query we defined a stored procedure which extracted member demographic data based on claim_status. While defining the procedure we defined claim_status as a parameter which could be passed while calling the procedure. Stored procedure also adds an extra effectiveness in terms of data security.

- **Stored Procedure** for extracting member demographic data by using parameter claim_status

```
use health_insurance_claims_database;
DELIMITER //
> CREATE PROCEDURE get_member_claim_status_data(
    IN claim_status_in VARCHAR(50)
~ )
> BEGIN
    select m.member_first_name, m.member_last_name, m.member_dob, m.gender, m.claim_id,
        s.claim_status, s.type
    from
    >      ((member m inner join claim c on m.claim_id = c.claim_id)
    ~      inner join status s on s.status_id = c.status_id)
    where s.claim_status = claim_status_in
    order by m.member_first_name;
~ END //
DELIMITER ;
CALL get_member_claim_status_data('paid');
```

- **Output:**

member_first_name	member_last_name	member_dob	gender	claim_id	claim_status	type
Abigail	Mosley	30-08-1987	Female	951	paid	d
Britanni	Sloan	25-10-1980	Male	961	paid	d
Candice	Skinner	28-06-1981	Male	931	paid	d
Dacey	Finch	03-05-1976	Female	946	paid	d
Dana	Guerrero	26-02-1974	Male	901	paid	d
Davis	Perry	02-11-2003	Male	981	paid	d
Debra	Stokes	30-08-1982	Male	936	paid	d
Eve	Spence	30-10-1999	Not Answered	991	paid	d
Hunter	Mcdowell	25-07-1980	Female	911	paid	d
India	Harding	22-01-1995	Not Answered	971	paid	d
Jordan	Hill	28-08-1994	Female	921	paid	d
Kristen	Cantrell	03-04-2005	Male	926	paid	d
Mia	Mcgee	24-09-1989	Male	966	paid	d
Paula	Adkins	12-04-1995	Female	986	paid	d
Rhoda	Mckay	20-05-1996	Male	906	paid	d
Ria	Wells	06-10-2002	Male	976	paid	d
Ruby	Travis	08-06-2004	Female	956	paid	d
Tashya	Bowen	25-07-2010	Not Answered	941	paid	d
Zeus	Matthews	26-11-1994	Male	916	paid	d

4.3 Use of Indexes for optimizing table traversals

Indexing is vital for enhancing performance of the search queries. Indexes can be used to identify and retrieve records uniquely. In our project, we tried Indexing for multiple tables to achieve faster results and less overhead. We used B-Tree indexes in our database.

Indexes example:

The below snapshot shows an instance of the query which we created before and after creating index on address_id.

```
use health_insurance_claims_database;

select distinct m.member_first_name, m.member_last_name, cp.coverage_name, m.address_id
from (coverage cp inner join member m on m.member_id = cp.member_id)
where m.address_id between 2000 and 5000;

create index member_index on member(address_id);

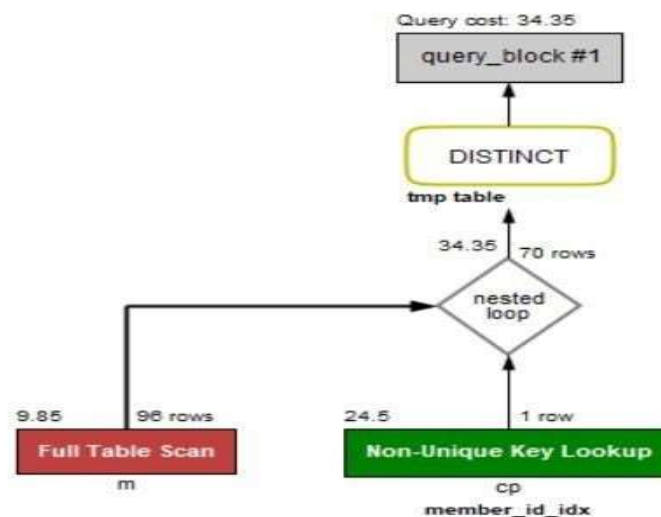
select distinct m.member_first_name, m.member_last_name, cp.coverage_name, m.address_id
from (coverage cp inner join member m on m.member_id = cp.member_id)
where m.address_id between 2000 and 5000;
```

Output:

	member_first_name	member_last_name	coverage_name	address_id
▶	Kadeem	Bush	SILVER	3457
	Dana	Guerrero	SILVER	4557
	Gretchen	Browning	PLATINUM	2235
	Calista	Vance	PLATINUM	4366
	Evan	Simon	SILVER	2265
	Amal	Whitehead	BRONZE	2041
	Rhoda	Mckay	GOLD	3286
	Porter	Battle	BRONZE	3960
	Leah	Carrillo	SILVER	4542
	Maris	Santiago	BRONZE	3519
	Yolanda	Bryan	GOLD	2157
	Colton	Wiggins	BRONZE	2061
	Brynne	Nash	SILVER	4806
	Zeus	Matthews	PLATINUM	2583
	Shelby	Chandler	PLATINUM	4067
	Latifah	Goodman	SILVER	3381
	Chester	Powers	SILVER	2107
	Keely	Pena	SILVER	2224

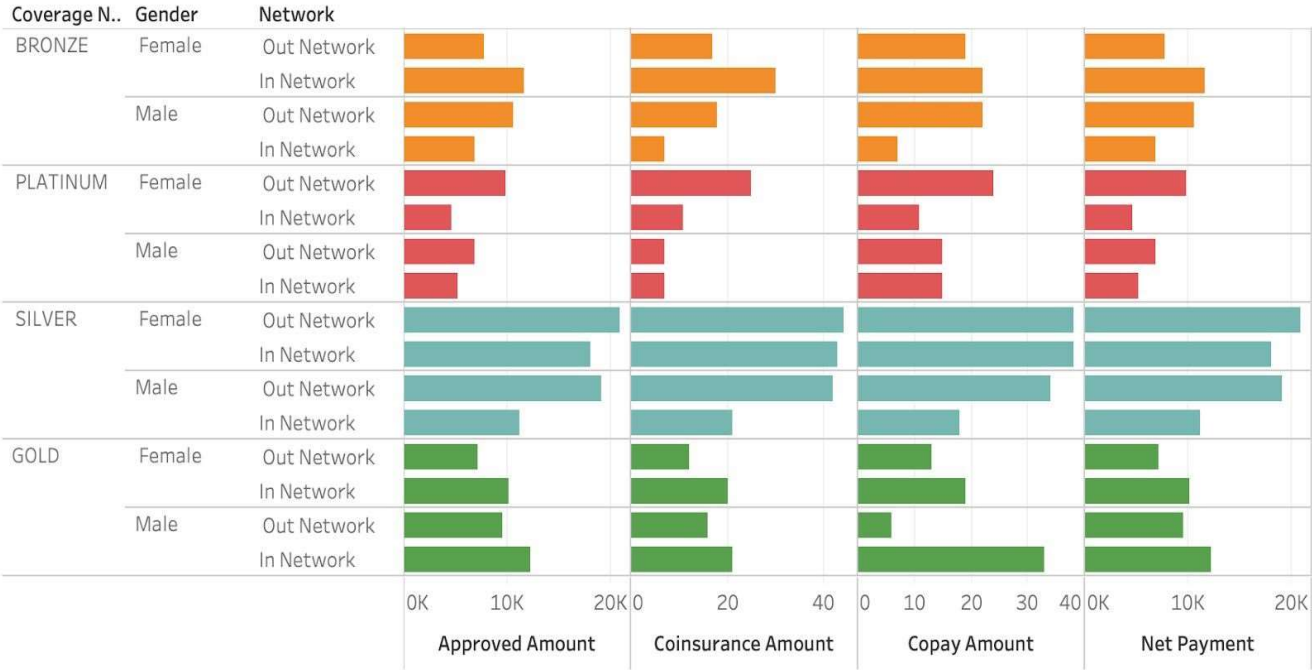
Explain Plan

The below snapshot shows the query plan generated by executing a query using indexed columns. In the figure we can see that the executing query is 34.35 which is less than for query which was executed without indexing.

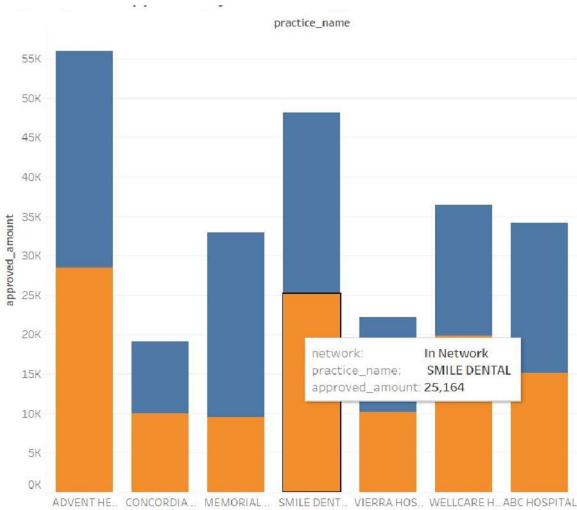


Data Visualizations

Net Payment Covered by Different Plans



Practice Names with Maximum Approved Amount



Claim Amount Approved by Practice Name



Summary

This project can be utilized as a reporting model by a variety of target audiences such as health insurance companies, facilities and physicians. The queries developed can be used and tweaked for a variety of insightful data extractions. We developed queries by formulating questions which are used in diurnal business operations. In addition to query writing we focused on performance tuning which is crucial in terms of big data applications.

This project helped us to dive deep into the business domain of medical insurance. Apart from the insurance industry we also focussed on medical facilities datasets which helped us to extract insights from the medical field. Considering the future scope of the project, we would like to extend this system by formulation transactions and triggers. Apart from real world industry knowledge the project enabled us to gain hands on experience of developing an entire database administration system from scratch.

Moreover, integrating MySQL with Tableau enabled us to analyse real time data on the go and develop interactive visualizations which could be crucial in identifying Key Performance Indicators.



References

1. <https://money.usnews.com/money/personal-finance/family-finance/articles/2018-10-18/7factors-to-consider-when-shopping-for-health-insurance>
2. <https://www.sisense.com/blog/8-ways-fine-tune-sql-queries-production-databases/>
3. <https://www.sqlshack.com/query-optimization-techniques-in-sql-server-the-basics/> 4.
<https://dataschool.com/sql-optimization/how-indexing-works/>