ISM 6218: Advanced Database Management

Group 5

Final Project

Kuo Zhao

Sri Satya Arumilli

Shree Vaishnavi Reddy Bhoomi Reddy

Pooja Regadda

Seshi Reddy Syamala

Table of contents

Clinic Database Design

1. Introduction

Our team prepared to design and build a relational database using Oracle SQL Developer to try to meet the complex requirements of a modern clinic system. Our database will be a relatively comprehensive repository that integrates entities critical to healthcare and administration. These entities have been carefully selected to represent fundamental aspects of healthcare delivery and management. Our main goal is to create an efficient and reliable system that acts as the nerve center of the clinic reception desk, responsible for tasks related to appointment management, reception, guidance, post-visit information retention, and the storage of comprehensive patient information.

Our database will optimize management operations and improve resource allocation and scheduling efficiency. In addition, considering the rapid changes in medical technology and regulations, our design approach is modular and flexible. This ensures that the database can be easily updated or modified to accommodate new healthcare practices, regulatory changes, or technological innovations, enabling clinics to achieve long-term operational excellence.

Description of process within clinic:

In the typical patient's journey at the clinic, an individual begins by scheduling an appointment, during which they provide their personal and insurance details. Upon arriving at the clinic, the front desk verifies the appointment and patient information, following which a nurse guides the patient to a consultation room. The nurse then conducts a preliminary assessment, noting down the patient's primary concerns and symptoms. Subsequently, a doctor consults with the patient to understand the ailment in-depth, formulates a diagnosis, and provides treatment recommendations. This could range from prescribing medications, offering in-clinic treatments using specialized equipment. The visit culminates in billing, which considers services provided and insurance coverage.

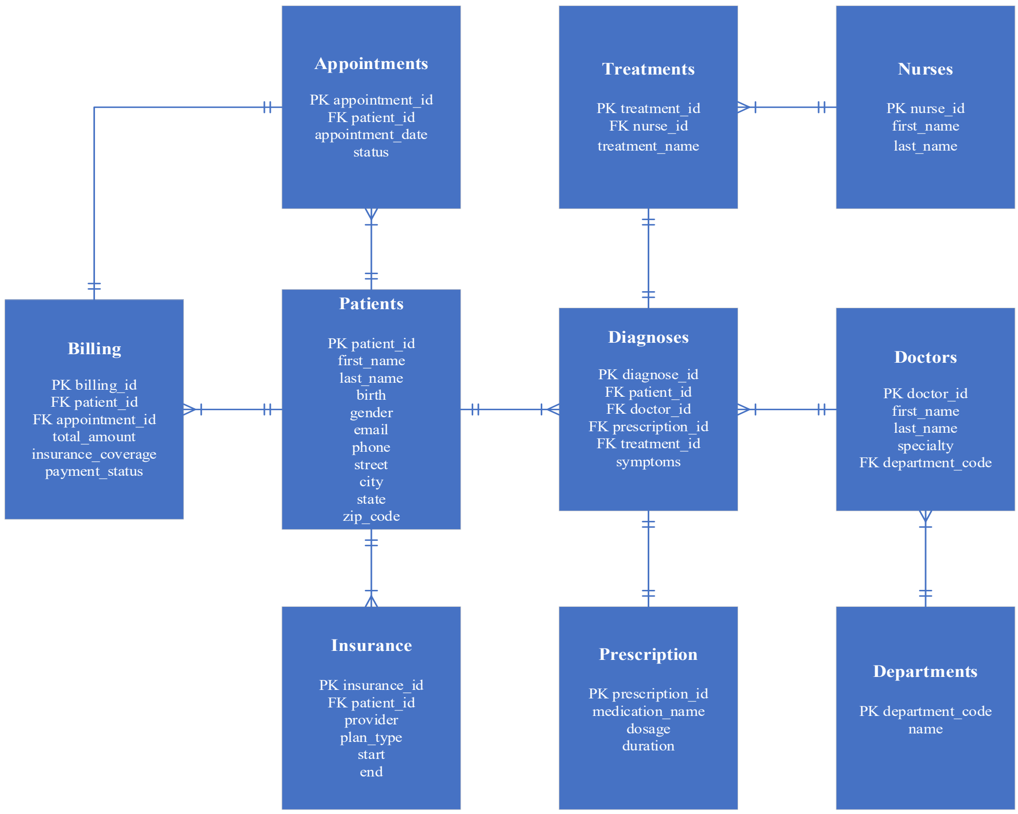
2. Entity Sets

1). appointments: **PK** appointment_id; **FK** patient_id; date; status
2). billing: **PK** billing_id; **FK** patient_id; **FK** appointment_id; total_amount; insurance_coverage; payment_status
3). departments: **PK** department_code; name
4). diagnoses: **PK** diagnose_id; **FK** patient_id; **FK** doctor_id; **FK** prescription_id; **FK** treatment_id; symptoms
5). doctors: **PK** doctor_id; first_name; last_name; specialty; **FK** department_code
6). insurance: **PK** insurance_id; **FK** patient_id; provider; plan_type; start; end
7). nurses: **PK** nurse_id; first_name; last_name
8). patients: **PK** patient_id; first_name; last_name; birth; gender; email; phone; street; city; state; zip_code
9). prescription: **PK** prescription_id; medication_name; dosage; duration
10). treatments: **PK** treatment_id; **FK** nurse_id; treatment_name

3. Relationships of Entity Sets

1). Patients and Appointments: One-to-Many (Each patient can have multiple appointments, but each appointment is with one patient.)
2). Patients and Billing: One-to-Many (Each patient can have multiple billing records, but each billing record is for one patient.)
3). Appointments and Billing: One-to-One (Each appointment can have one billing record, and each billing record is linked to a single appointment.)
4). Doctors and Diagnoses: One-to-Many (A doctor can make multiple diagnoses, but each diagnosis is made by one doctor.)
5). Patients and Diagnoses: One-to-Many (A patient can have multiple diagnoses, but each diagnosis is for one patient.)
6). Prescription and Diagnoses: One-to-One (Each diagnosis can have one prescription, and each prescription can be linked to a single diagnosis.)
7). Treatments and Diagnoses: One-to-One (Each diagnosis can have one treatment, and each treatment can be linked to a single diagnosis.)
8). Departments and Doctors: One-to-Many (A department can have multiple doctors, but each doctor belongs to one department.)
9). Patients and Insurance: One-to-Many (A patient can have multiple insurance records, but each insurance record is for one patient.)
10). Nurses and Treatments: One-to-Many (A nurse can administer multiple treatments, but each treatment is administered by one nurse.)

4. ER Diagram



5. Physical Design

The SQL for creating tables, constraints, indexes and PK and FK
(The Engine we chose is Oracle DBMS - Oracle SQL Developer)

5.1 Table Generation (Including Data Types, PK and FK)

```
-- Create the 'departments' table
CREATE TABLE departments (
    department_code VARCHAR2(20) PRIMARY KEY,
    name VARCHAR2(50)
);
-- Create the 'doctors' table
CREATE TABLE doctors (
    doctor_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    specialty VARCHAR2(50),
    department_code VARCHAR2(20),
    FOREIGN KEY (department_code) REFERENCES departments(department_code)
);
-- Create the 'patients' table
CREATE TABLE patients (
    patient_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    birth DATE,
    gender VARCHAR2(10),
    email VARCHAR2(100),
    phone VARCHAR2(15),
    street VARCHAR2(255),
    city VARCHAR2(50),
    state VARCHAR2(20),
    zip_code VARCHAR2(10)
);
-- Create the 'nurses' table
CREATE TABLE nurses (
    nurse_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50)
);
-- Create the 'appointments' table
CREATE TABLE appointments (
    appointment_id NUMBER PRIMARY KEY,
    patient_id NUMBER,
    appointment_date DATE,
```

```sql
    status VARCHAR2(20),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);
-- Create the 'insurance' table
CREATE TABLE insurance (
    insurance_id NUMBER PRIMARY KEY,
    patient_id NUMBER,
    provider VARCHAR2(50),
    plan_type VARCHAR2(20),
    start_date DATE,
    end_date DATE,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);
-- Create the 'billing' table
CREATE TABLE billing (
    billing_id NUMBER PRIMARY KEY,
    patient_id NUMBER,
    appointment_id NUMBER,
    total_amount DECIMAL(10,2),
    insurance_coverage DECIMAL(10,2),
    payment_status VARCHAR2(20),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (appointment_id) REFERENCES appointments(appointment_id)
);
-- Create the 'prescriptions' table
CREATE TABLE prescriptions (
    prescription_id NUMBER PRIMARY KEY,
    medication_name VARCHAR2(50),
    dosage VARCHAR2(255),
    duration VARCHAR2(255)
);
-- Create the 'diagnoses' table
CREATE TABLE diagnoses (
    diagnose_id NUMBER PRIMARY KEY,
    patient_id NUMBER,
    doctor_id NUMBER,
    prescription_id NUMBER,
    treatment_id NUMBER,
    symptoms VARCHAR2(255),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id),
    FOREIGN KEY (prescription_id) REFERENCES prescriptions(prescription_id)
);
-- Create the 'treatments' table
```

```
CREATE TABLE treatments (
    treatment_id NUMBER PRIMARY KEY,
    nurse_id NUMBER,
    treatment_name VARCHAR2(100),
    FOREIGN KEY (nurse_id) REFERENCES nurses(nurse_id)
);
```

6. Data Integrity

In addressing the requirements for the Data Integrity section of our database design project, we aim to explore and implement a variety of integrity constraints. These include primary keys, foreign keys, check constraints, unique constraints, and not null constraints. Here, we only showing that we are selecting a few representative tables from our database to apply these constraints, ensuring data quality and integrity. Although this is only the part on Data Integrity, we could understand the practical application and significance of these constraints in maintaining a robust and reliable database, while also providing a clear understanding of their role in upholding data integrity in a real-world context.

1) Constraint: Primary Key
Table: appointments
Description: Ensures each appointment is uniquely identifiable.
DDL: ALTER TABLE appointments ADD CONSTRAINT pk_appointments PRIMARY KEY (appointment_id);

2) Constraint: Foreign Key
Table: diagnoses
Description: Links each diagnosis to a specific patient, ensuring referential integrity.
DDL: ALTER TABLE diagnoses ADD CONSTRAINT fk_diagnoses_patient_id FOREIGN KEY (patient_id) REFERENCES patients(patient_id);

3) Constraint: Check Constraint
Table: billing
Description: Ensures the payment status is valid (e.g., 'Paid', 'Pending', 'Cancelled').
DDL: ALTER TABLE billing ADD CONSTRAINT chk_billing_payment_status
CHECK (payment_status IN ('Paid', 'Pending', 'Unpaid', 'Declined', 'Claim Rejected'));

4) Constraint: Unique Constraint
Table: billing
Description: Guarantees that each billing record is unique to an appointment.
DDL: ALTER TABLE billing ADD CONSTRAINT unq_billing_appointment_id UNIQUE (appointment_id);

5) Constraint: Not Null Constraint
Table: billing
Description: Ensures critical billing information, like the total amount, is always provided.
DDL: ALTER TABLE billing MODIFY total_amount NOT NULL;

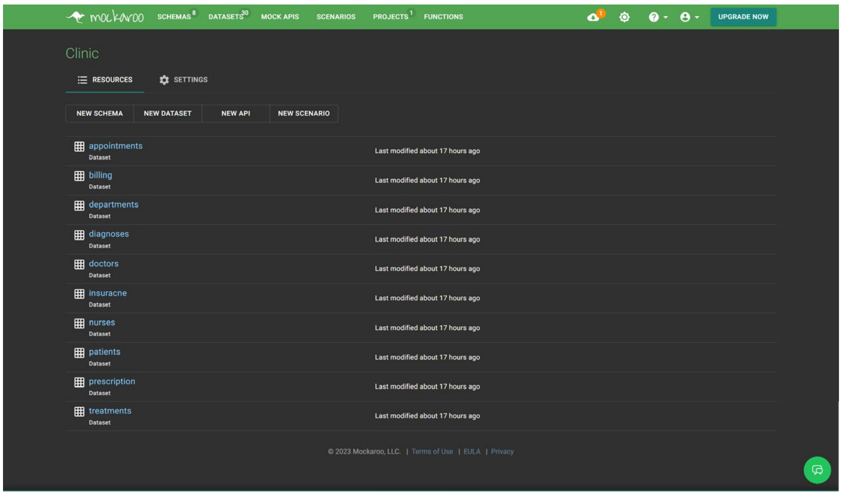
7. Indexes From the performance Tunning

1) CREATE INDEX idx_doctors_specialty_lastname ON doctors(specialty, last_name);
2) CREATE INDEX idx_patients_phone ON patients(phone);
3) CREATE INDEX idx_patients_city_birth ON patients(city, birth);
4) CREATE INDEX idx_diagnoses_symptoms ON diagnoses(symptoms);
5) CREATE INDEX idx_doctors_covering ON doctors(doctor_id, first_name, last_name);
   CREATE INDEX idx_diagnoses_doctor_id ON diagnoses(doctor_id);
6) CREATE INDEX idx_patients_covering ON patients(patient_id, first_name, last_name);
   CREATE INDEX idx_diagnoses_covering ON diagnoses(patient_id, symptoms, treatment_id, prescription_id);
   CREATE INDEX idx_treatments_covering ON treatments(treatment_id, treatment_name);
   CREATE INDEX idx_prescriptions_covering ON prescriptions(prescription_id, medication_name);

8. Data Generation and Loading

8.1 Data Generation:

In this case we used the MocKaroo to generate all the data, to make sure each data has been related on different tables, and the data has been random generalized. Here is a screenshot with our project in MocKaroo.



Screenshot of the MocKaroo

8.2 The description of the data in each table:

There are 1,000 rows in the **diagnoses**, **prescriptions**, and **treatments** table, and they are all related.

There are 1,000 rows in the **billing** and **insurance** table and related to all the patients with the diagnoses, prescriptions, and treatments records.

There are 32 rows in **doctors** table and 10 rows in **departments** table means that we have thirty-two doctors in ten different departments in this clinic.

There are 51 rows in the **nurses** table means that we have fifty-one nurses.

There are 1,500 rows in the **patients** and **appointments** table, 1,000 records of them are related to all other tables and other 500 records are only related to the appointments table.

8.3 The glance of part of data for each table:

billing_id	patient_id	appointment_id	total_amount	insurance_coverage	payment_status
80001	3272128298	60001	762.46	609.97	Pending
80002	4581087376	60002	722.16	649.94	Paid
80003	6794509113	60003	483.25	483.25	Paid
80004	6298669175	60004	594.72	475.78	Claim Rejected
80005	5841485075	60005	450.12	405.11	Paid
80006	0206411847	60006	378.83	340.95	Paid
80007	9518821704	60007	994.92	895.43	Paid
80008	2060518350	60008	993.26	893.93	Paid
80009	0501515445	60009	975.16	975.16	Paid
80010	7625775992	60010	225.35	157.75	Claim Rejected
80011	3421889996	60011	482.71	386.17	Declined
80012	1230967621	60012	707.03	636.33	Paid
80013	2368954716	60013	794.26	635.41	Paid
80014	3338568681	60014	346.14	276.91	Pending
80015	9474907129	60015	889.04	800.14	Paid
80016	4150714827	60016	756.57	605.26	Pending
80017	7496934755	60017	275.26	220.21	Paid
80018	9912743326	60018	972.2	874.98	Paid
80019	6007838935	60019	963.87	771.1	Unpaid

doctor_id	first_name	last_name	specialty	department_code
1011	Alexio	Skinner	Orthopedist	ORTHO
1012	Wilma	Klouz	Pediatrician	PEDS
1013	Kane	Carbry	Gastroenterologist	GI
1014	Oralee	Cattrell	Orthopedist	ORTHO
1015	Mickie	Kearey	Dermatologist	DERM
1016	Ivar	Basnett	Pediatrician	PEDS
1017	Dorelle	Dignam	Ophthalmologist	OPHTH
1018	Gaile	Pischel	Dermatologist	DERM
1019	Garreth	Spenton	Urologist	URO
1020	Bertie	de Keep	Endocrinologist	ENDO
1021	Isadora	Petriello	Internist	IM
1022	Hendrick	McKoy	Audiologist	AUD
1023	Clywd	Dagworthy	General	ORTHO
1024	Matty	Eliet	Urologist	URO
1025	Marlie	Christall	Internist	IM
1026	Dillon	Jaulmes	Ophthalmologist	OPHTH
1027	Ladonna	Hammerberg	Dermatologist	DERM
1028	Jackie	Smedmoor	Internist	IM
1029	Sonnie	Thelwll	General	PEDS

nurse_id	first_name	last_name
2101	Chanda	Lowater
2102	Chelsey	Tuffrey
2103	Cherise	Frenzl
2104	Emlynn	Brownbill
2105	Atlante	Halbeard
2106	Eulalie	Zold
2107	Shani	Hinge
2108	Aggy	Challicombe
2109	Romona	Caulier
2110	Amalia	Prover
2111	Inessa	Belleny
2112	Nike	Steptow
2113	Caresse	Sanday
2114	Etheline	Bhar
2115	Kirstin	Preddy
2116	Latisha	Stelfox
2117	Alyss	Ogger
2118	Joyous	L'Episcopio
2119	Farah	Moir

diagnose_id	patient_id	doctor_id	prescription_id	treatment	symptoms
50101	3272128298	1011	12101	13101	Joint pain
50102	4581087376	1011	12102	13102	Limited range of motion
50103	6794509113	1014	12103	13103	Swelling around a joint
50104	6298669175	1023	12104	13104	Bone fractures
50105	5841485075	1014	12105	13105	Muscle pain
50106	0206411847	1011	12106	13106	Back pain
50107	9518821704	1011	12107	13107	Tingling
50108	2060518350	1023	12108	13108	Tenderness
50109	0501515445	1014	12109	13109	Joint pain
50110	7625775992	1011	12110	13110	Limited range of motion
50111	3421889996	1023	12111	13111	Swelling around a joint
50112	1230967621	1011	12112	13112	Bone fractures
50113	2368954716	1011	12113	13113	Muscle pain
50114	3338568681	1011	12114	13114	Back pain
50115	9474907129	1011	12115	13115	Tingling
50116	4150714827	1014	12116	13116	Tenderness
50117	7496934755	1023	12117	13117	Joint pain
50118	9912743326	1023	12118	13118	Limited range of motion
50119	6007838935	1014	12119	13119	Swelling around a joint

prescriptior	medication_name	dosage	duration
12101	Ibuprofen	200-800 mg every 6-8 hours	as needed
12102	Naproxen	250-500 mg twice daily	3 days
12103	Prednisone	5-60 mg daily in single or div	3 days
12104	Acetaminophen	325-1000 mg every 4-6 hour	3 days
12105	Methocarbamol	500-1500 mg every 6 hours	3 days
12106	Acetaminophen	325-1000 mg every 4-6 hour	3 days
12107	Gabapentin	Starting at 300 mg at bedtim	5 days
12108	Ibuprofen	200-800 mg every 6-8 hours	as needed
12109	Acetaminophen	325-1000 mg every 4-6 hour	1-2 days
12110	Ibuprofen	200-800 mg every 6-8 hours	as needed
12111	Ibuprofen	200-800 mg every 6-8 hours	as needed
12112	Ibuprofen	200-800 mg every 6-8 hours	as needed
12113	Acetaminophen	325-1000 mg every 4-6 hour	3 days
12114	Ibuprofen	200-800 mg every 6-8 hours	as needed
12115	Pregabalin	50-150 mg two or three time	5 days
12116	Naproxen	250-500 mg twice daily	3 days
12117	Naproxen	250-500 mg twice daily	3 days
12118	Diclofenac	Apply to affected area 3-4 ti	1-2 weeks
12119	Acetaminophen	325-1000 mg every 4-6 hour	3 days

department_code	name
ORTHO	Orthopedics
PEDS	Pediatrics
GI	Gastroenterology
DERM	Dermatology
OPHTH	Ophthalmology
URO	Urology
ENDO	Endocrinology
IM	Internal Medicine
AUD	Audiology
RESP	Respiratory

patient_id	first_name	last_name	birth	gender	email	phone	street	city	state	zip_code
3272128298	Hendrika	Battlestone	4/6/1981	Female	hbattlestone0@	941-813-1858	363 Mitchell P	Punta Gorda	Florida	33982
4581087376	Aleece	Zanardii	1/31/1966	Female	azanardii1@ust	727-575-6836	370 Rieder Hil	Largo	Florida	34643
6794509113	Ronnie	Nairns	8/16/1985	Male	rnairns2@good	561-251-3407	799 Chinook W	West Palm Be	Florida	33416
6298669175	Spencer	Strevens	8/5/1992	Male	sstrevens3@go	772-658-2307	74 Stone Corr	Vero Beach	Florida	32969
5841485075	Rod	Dingsdale	5/30/2001	Male	rdingsdale4@n	941-247-9044	6 Blaine Cross	North Port	Florida	34290
0206411847	Urbanus	Krojn	10/23/1991	Male	ukrojn5@360.cr	850-987-3031	584 Myrtle Pla	Tallahassee	Florida	32314
9518821704	Ofilia	Laydel	1/21/1990	Female	olaydel6@globe	813-670-0587	548 Nova Poir	Tampa	Florida	33694
2060518350	Aloisia	Leadbeater	5/3/1968	Female	aleadbeater7@	941-608-3212	0762 Dottie C	Bradenton	Florida	34205
0501515445	Thacher	Wild	1/7/1961	Male	twild8@networ	561-466-8948	281 Service Pl	Delray Beach	Florida	33448
7625775992	Blakelee	Beltzner	10/10/2000	Female	bbeltzner9@ml	904-517-6716	03 Meadow V	Jacksonville	Florida	32255
3421889996	Heriberto	MacPeake	6/23/1996	Male	hmacpeakea@l	954-492-9479	29627 Dorton	Fort Lauderd	Florida	33330
1230967621	Terence	Gaddas	6/3/1994	Male	tgaddasb@fotk	772-471-5262	7 Arrowood P	Fort Pierce	Florida	34981
2368954716	Audrey	Fortnon	7/8/1988	Female	afortnonc@tiny	904-809-2257	49 Kingsford A	Jacksonville	Florida	32277
3338568681	Roderich	Wheowall	2/8/1971	Male	rwheowalld@di	305-339-3820	1 Bartelt Lane	Hialeah	Florida	33018
9474907129	Randene	Lording	8/29/2000	Female	rlordinge@seat	305-851-4256	073 Forest Da	Naples	Florida	33961
4150714827	Noach	Dendle	3/1/1974	Male	ndendlef@goog	772-614-2952	54 Morrow W	Vero Beach	Florida	32969
7496934755	Britta	Garside	6/22/1964	Female	bgarsideg@dig	305-924-2374	1002 Utah Crc	Hialeah	Florida	33018
9912743326	Ferdy	Geraghty	2/17/1980	Male	fgeraghtyh@hu	850-175-5903	62 Washingto	Pensacola	Florida	32595
6007838935	Vida	Pouck	7/6/1996	Female	vpoucki@thegu	239-249-6447	435 Bonner Pl	Fort Myers	Florida	33913

insurance_id	patient_id	provider	plan_type	start	end
30001	3272128298	BCBS	HMO	8/10/2021	10/1/2025
30002	4581087376	CNC	PPO	4/3/2021	10/30/2025
30003	6794509113	CI	PPO	4/12/2021	10/29/2025
30004	6298669175	BCBS	HDHP	10/14/2021	5/3/2024
30005	5841485075	HUM	EPO	9/23/2021	4/4/2024
30006	0206411847	UHC	HMO	11/4/2021	7/29/2024
30007	9518821704	CI	PPO	5/23/2022	12/18/2025
30008	2060518350	USAA	HMO	5/27/2021	5/10/2024
30009	0501515445	AET	POS	4/25/2021	10/30/2024
30010	7625775992	AET	HDHP	3/29/2023	10/8/2025
30011	3421889996	CI	HDHP	12/24/2021	9/19/2025
30012	1230967621	CNC	HDHP	1/20/2023	1/24/2025
30013	2368954716	HUM	HMO	1/10/2023	12/2/2024
30014	3338568681	BCBS	EPO	4/20/2023	3/8/2025
30015	9474907129	CNC	PPO	7/8/2022	9/17/2024
30016	4150714827	AET	POS	8/13/2023	10/23/2024
30017	7496934755	CNC	PPO	2/20/2022	8/26/2025
30018	9912743326	CNC	HDHP	9/29/2021	8/7/2024
30019	6007838935	BCBS	EPO	9/18/2023	10/9/2025

appointme	patient_id	appointmrnt_date	status
60001	3272128298	8/5/2023	Completed
60002	4581087376	9/8/2023	Completed
60003	6794509113	8/12/2023	Completed
60004	6298669175	9/18/2023	Completed
60005	5841485075	8/31/2023	Completed
60006	0206411847	9/28/2023	Completed
60007	9518821704	9/3/2023	Completed
60008	2060518350	8/16/2023	Completed
60009	0501515445	9/7/2023	Completed
60010	7625775992	9/20/2023	Completed
60011	3421889996	8/20/2023	Completed
60012	1230967621	9/20/2023	Completed
60013	2368954716	8/12/2023	Completed
60014	3338568681	9/4/2023	Completed
60015	9474907129	8/8/2023	Completed
60016	4150714827	8/1/2023	Completed
60017	7496934755	9/7/2023	Completed
60018	9912743326	9/28/2023	Completed
60019	6007838935	9/11/2023	Completed

treatment_	nurse_id	treatment_name
13101	2102	physical therapy
13102	2149	Physical Therapy
13103	2117	Rest, Ice, Compression, Elevation (RICE)
13104	2109	Casting or Splinting
13105	2107	Rest and Modified Activity
13106	2136	Rest and Modified Activity
13107	2123	Physical Therapy
13108	2121	Rest and Modified Activity
13109	2118	corticosteroids injections
13110	2125	heat packs
13111	2108	Joint Aspiration
13112	2115	Physical Therapy
13113	2115	Physical Therapy
13114	2124	Nerve Conduction Studies
13115	2127	Physical Therapy
13116	2133	Rest
13117	2147	Stretching Exercises
13118	2126	Splinting or Bracing
13119	2102	Physical Therapy

Query Writing

1. Title: Retrieve List of All Patients for a Specific Doctor

User case:

The clinic's management wants to review the list of all patients consulted by “Dr. Bubeer” in the “Internist” department.

Query:

SELECT p.first_name, p.last_name, p.email, p.phone
FROM patients p
INNER JOIN diagnoses d ON p.patient_id = d.patient_id
INNER JOIN doctors doc ON d.doctor_id = doc.doctor_id
WHERE doc.last_name like 'Bubeer' AND doc.specialty like 'Internist';

Results:

	FIRST_NAME	LAST_NAME	EMAIL	PHONE
1	Saundra	Sainer	ssainerji@miitbeian.gov.cn	786-526-7556
2	Zarla	Manoch	zmanochju@shinystat.com	786-543-1752
3	Bartie	Gownge	bgowngejv@istockphoto.com	305-903-2049
4	Nobie	Catterick	ncatterickjx@hud.gov	850-614-1058
5	Jabez	Kerss	jkerssk3@gov.uk	941-922-0581
6	Norton	Beardsley	nbeardsleyk4@hexun.com	305-267-2079
7	Kerrin	Legion	klegionka@sogou.com	407-538-7662
8	Fionna	Buller	fbullerkh@disqus.com	386-729-9738
9	Philippa	Lambillion	plambillionkk@tuttocitta.it	850-152-0876
10	Garald	Sibyllina	gsibyllinakt@a8.net	727-279-8348
11	Jeana	Blackah	jblackahlt@ebay.com	727-387-7780
12	Amandie	Chace	achacelu@hibu.com	407-136-3059
13	Austin	Cannan	acannanm3@people.com.cn	954-350-6796
14	Maurits	Acutt	macuttm5@theglobeandmail.com	305-990-0231
15	Danna	Colam	dcolamlr@abc.net.au	941-851-3523
16	Selinda	Awcoate	sawcoatekx@ask.com	850-701-3380
17	Janessa	Lawfull	jlawfulllq@wiley.com	407-759-2096

Conclusion: The results in the table show that there are 17 records that Dr. Bubeer consulted when he is in the Internist department. The query provides a comprehensive list of all patients that have been consulted by a specific doctor. This is crucial for both administrative and medical reviews, helping in understanding the doctor's patient load and specialty areas.

2. Title: List Doctors Based on Specialty and Department

User case: If a patient with serious diseases of the eye is visiting, and the receptionist wants to provide a list of all Ophthalmologist in the Ophthalmology department.

Query:

SELECT doc.first_name, doc.last_name, d.name as Department_Name
FROM doctors doc
INNER JOIN departments d
ON doc.department_code = d.department_code
WHERE doc.specialty LIKE 'Ophthalmologist';

Results:

	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
1	Dorelle	Dignam	Ophthalmology
2	Emily	Thompson	Ophthalmology
3	Dillon	Jaulmes	Ophthalmology

Conclusion: We can see there are only three Ophthalmologist in this clinic. However, this query still can quickly identify doctors based on their specialty and department. The results facilitate efficient patient routing and help in ensuring patients receive care from the most suitable professionals.

3. Title: Total Billing Amount for a Specific Patient

User case: The accounting department wants to review the total billed amount for a patient named “Morie Amer” for the current fiscal year.

Query:

SELECT SUM(b.total_amount) AS TotalBilledAmount
FROM billing b
INNER JOIN appointments a ON b.appointment_id = a.appointment_id
INNER JOIN patients p ON a.patient_id = p.patient_id

WHERE p.first_name LIKE 'Morie' AND p.last_name LIKE 'Amer'
AND EXTRACT(YEAR FROM a.appointment_date) = EXTRACT(YEAR FROM SYSDATE);

Results:

	TOTALBILLEDAMOUNT
1	818.04

Conclusion: The total amount of the patient's billing in this year is 818.04. This query helps in understanding the financial aspects related to a specific patient, providing a clear picture of the revenue generated and facilitating financial planning and audits.

4. Title: Patient History Retrieval

User case: A doctor wants to quickly retrieve the entire medical history of a patient before an appointment.

Query:

SELECT p.first_name, p.last_name, d.symptoms, pr.medication_name, t.treatment_name
FROM patients p
JOIN diagnoses d ON p.patient_id = d.patient_id
JOIN prescriptions pr ON d.prescription_id = pr.prescription_id
JOIN treatments t ON d.treatment_id = t.treatment_id
WHERE p.patient_id = 7825607864;

Results:

	FIRST_NAME	LAST_NAME	SYMPTOMS	MEDICATION_NAME	TREATMENT_NAME
1	Wendy	MacInnes	Contact dermatitis	Topical antipruritic	Emollients or moisturizers

Conclusion: The result in table shows the patient's name and his/her records. This query allows doctors to swiftly retrieve a patient's comprehensive medical history, which includes symptoms, prescriptions, and treatments. Such a holistic view facilitates improved diagnosis and care during the patient's appointment.

5. Title: Most Active Doctors

User case: The clinic's management wants to identify which doctors have diagnosed the most patients in September to appreciate their hard work.

Query:

SELECT d.first_name, d.last_name, COUNT(diagnose_id) AS total_diagnoses
FROM doctors d
INNER JOIN diagnoses dn ON d.doctor_id = dn.doctor_id
INNER JOIN appointments a ON dn.patient_id = a.patient_id
WHERE EXTRACT (MONTH FROM a.appointment_date)= 9
GROUP BY d.first_name, d.last_name
ORDER BY total_diagnoses DESC
FETCH FIRST 5 ROWS ONLY;

Results:

	FIRST_NAME	LAST_NAME	TOTAL_DIAGNOSES
1	Chris	McLellan	24
2	Ivar	Basnett	22
3	Sarah	Kim	21
4	Urban	Blandamere	21
5	Alexio	Skinner	20

Conclusion: The results in the table show that the Top five doctors who diagnosed most patients in September. The query provides a list of the top-performing doctors based on the number of patients diagnosed in a specific month. It assists the management in recognizing and rewarding the dedication of these professionals.

6. Title: Departmental Workload Assessment

User case: As a statistic for human resources, the clinic's management wants to know the number of diagnoses made by each department this year.

Query:

```
SELECT dp.name AS department_name, COUNT(dn.diagnose_id) AS total_diagnoses
FROM departments dp
INNER JOIN doctors d ON dp.department_code = d.department_code
INNER JOIN diagnoses dn ON d.doctor_id = dn.doctor_id
INNER JOIN appointments a ON dn.patient_id = a.patient_id
WHERE EXTRACT(YEAR FROM sysdate) = EXTRACT(YEAR FROM a.appointment_date)
GROUP BY dp.name
ORDER BY total_diagnoses DESC;
```

Results:

	DEPARTMENT_NAME	TOTAL_DIAGNOSES
1	Ophthalmology	100
2	Gastroenterology	100
3	Orthopedics	100
4	Endocrinology	100
5	Internal Medicine	100
6	Audiology	100
7	Dermatology	100
8	Pediatrics	100
9	Respiratory	100
10	Urology	100

Conclusion: The results in the table show all the departments and their number of diagnoses for the year. Though the results are seemed like unnormal because we split the number of departments equally for better data generation. The query gives a clear picture of the workload handled by each department in a particular year. This assists the management in resource allocation, ensuring no department is overstretched.

7. Title: Highest Billing Patients After Insurance Deductions

User case: The clinic's management wants to identify patients who have incurred the highest net bills (after insurance deductions) in this year for a loyalty rewards program.

Query:

```
SELECT p.first_name, p.last_name,
SUM(b.total_amount - b.insurance_coverage) AS net_expenditure
FROM patients p
```

```
JOIN billing b ON p.patient_id = b.patient_id
INNER JOIN appointments a ON b.patient_id = a.patient_id
WHERE EXTRACT(YEAR FROM sysdate) = EXTRACT(YEAR FROM a.appointment_date)
GROUP BY p.first_name, p.last_name
HAVING SUM(b.total_amount - b.insurance_coverage) > 100
ORDER BY net_expenditure DESC
FETCH FIRST 5 ROWS ONLY;
```

Results:

	FIRST_NAME	LAST_NAME	NET_EXPENDITURE
1	Junina	Cannop	397.67
2	Vi	Mirrlees	396.78
3	Tadio	Elleton	388.66
4	Nessa	Harmstone	384.2
5	Reinaldo	Hanshaw	368.84

Conclusion: The results in the table show the top five patients with their name and net billing amount. This query ascertains the top-spending patients after insurance deductions in the past year. Recognizing and rewarding these patients can foster a stronger patient-clinic relationship and enhance loyalty.

8. Title: Monthly Revenue Insights

User case: The accounts department wants to analyze the total revenue generated each month in this year to gauge financial performance.

Query:

```
SELECT TO_CHAR (a.appointment_date, 'Month YYYY') AS billing_month,
        SUM (b.total_amount - b.insurance_coverage) AS net_revenue
FROM billing b
INNER JOIN appointments a ON b.appointment_id = a.appointment_id
WHERE EXTRACT (YEAR FROM a.appointment_date) = EXTRACT(YEAR FROM
SYSDATE)
GROUP BY TO_CHAR (a.appointment_date, 'Month YYYY')
ORDER BY MIN (a.appointment_date);
```

Results:

	BILLING_MONTH	NET_REVENUE
1	August 2023	40692.68
2	September 2023	39868.36

Conclusion: The results in the table show the net revenue in each month this year, and we only have two months records in this year, so we only can see two months. The total net revenue is equal total amount minus insurance coverage. This query presents monthly revenue insights by considering both the total billed amount and insurance coverage. Such financial insights enable the clinic to identify trends and make informed decisions about future investments and strategies.

9. Title: Stored Procedure for Rescheduling an Appointment

User case: A patient contacts the clinic and requests to move their appointment to a different date. Instead of manually updating the database or relying on potentially error-prone application logic, a stored procedure is employed to ensure consistency and encapsulate the database logic.

Query:

```
-- Creating the stored procedure
CREATE OR REPLACE PROCEDURE RescheduleAppointment (
    p_appointment_id NUMBER,
    p_new_date DATE
) AS
BEGIN
    -- Update the appointments table to set the new date
    UPDATE appointments
    SET appointment_date = p_new_date
    WHERE appointment_id = p_appointment_id;

END;
```

Update appointment:

```
CALL RescheduleAppointment (60628,'2023-08-28');
```

Results: The date of appointment id “60628” has been change to Augus 28.

Conclusion: Using a stored procedure for tasks like rescheduling provides several advantages. It ensures data integrity by encapsulating and centralizing the logic in the database. It also provides an additional layer of security as direct table updates can be restricted, forcing the application to use the procedure. Finally, it provides a consistent interface for application developers, which can aid in reducing the occurrence of errors.

10. Title: Adding New Patient and Insurance Information

User Case: The clinic's reception desk needs to add a new patient, Kirk John, into the system along with his insurance information. Kirk is a new patient who has just provided his personal and insurance details. The clinic requires a streamlined process to enter all his information into the database in one go, ensuring both his personal details and insurance information are correctly linked and stored.

Stored Procedure:

```
CREATE OR REPLACE PROCEDURE AddNewPatient (
    p_patient_id IN NUMBER,
    p_first_name IN VARCHAR2,
    p_last_name IN VARCHAR2,
    p_birth IN DATE,
    p_gender IN VARCHAR2,
    p_email IN VARCHAR2,
    p_phone IN VARCHAR2,
    p_street IN VARCHAR2,
    p_city IN VARCHAR2,
    p_state IN VARCHAR2,
    p_zip_code IN VARCHAR2,
    P_insurance_id IN NUMBER,
```

```
	p_provider IN VARCHAR2,
	p_plan_type IN VARCHAR2,
	p_start_date IN DATE,
	p_end_date IN DATE
) AS
	v_patient_id NUMBER;
BEGIN
	-- Insert into patients table and fetch the new patient_id
	INSERT INTO patients (patient_id, first_name, last_name, birth, gender, email, phone, 
street, city, state, zip_code)
	VALUES (p_patient_id,p_first_name, p_last_name, p_birth, p_gender, p_email, p_phone, 
p_street, p_city, p_state, p_zip_code)
	RETURNING patient_id INTO v_patient_id;

	-- Insert into insurance table using the patient_id we just retrieved
	INSERT INTO insurance (insurance_id, patient_id, provider, plan_type, start_date, 
end_date)
	VALUES (P_insurance_id, v_patient_id, p_provider, p_plan_type, p_start_date, 
p_end_date);
END;
/
```

Procedure Call:

```
CALL AddNewPatient(
	'1358089118',			-- p_patient_id
	'Kirk',				-- p_first_name
	'John',				-- p_last_name
	TO_DATE('1995-06-01', 'YYYY-MM-DD'), -- p_birth
	'Male',				-- p_gender
	'john.K@example.com',-- p_email
	'813-382-1234',			-- p_phone
	'15350 Amberly Dr',		-- p_street
	'Tampa',			-- p_city
	'Florida',			-- p_state
	'33647',			-- p_zip_code
	'31001',	-- P_insurance_id
	'BCBS',		-- p_provider
	'POS',			-- p_plan_type
	TO_DATE('2023-01-01', 'YYYY-MM-DD'), -- p_start_date
	TO_DATE('2024-01-01', 'YYYY-MM-DD')   -- p_end_date
);
```

Conclusion: The AddNewPatient stored procedure is an essential tool for the clinic's administrative efficiency. It ensures the accurate capture of patient personal and insurance details. Normally, we will use auto-increment to automatically link these entities using the internally generated patient_id. But in this case, the data is in specific outsources, so we show the manual insert both patient_id and insurance_id. If we use the auto-increment in both tables, this method significantly reduces manual data entry, speeds up the patient registration process, maintains data integrity and helps prevent potential errors, thereby enhancing overall data quality and reliability in patient management.

Performance Tunning

1. Indexing strategies

All the Index creating codes will be concluded into the Physical Design Section.

1) Composite index on the 'specialty' and 'last_name' columns of the "doctors" table.

Purpose: If we want to increase the performance of the queries that interested in filtering doctors based on their specialty and last name. Considering the selectivity of these columns, a composite index is strategically beneficial. It would facilitate quick lookups and minimize the data scanned, especially when the query is focused on filtering through both columns. Additionally, as the "doctors" table grows, the composite index would increasingly demonstrate its efficiency over a full table scan.

Example Query:

SELECT * FROM doctors

WHERE specialty LIKE 'General' AND last_name LIKE 'McLellan';

Results:

	DOCTOR_ID	FIRST_NAME	LAST_NAME	SPECIALTY	DEPARTMENT_CODE
1	1033	Chris	McLellan	General	URO

Before adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				3
TABLE ACCESS	DOCTORS	FULL	1	3
Filter Predicates				
Other XML				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	7

After adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
TABLE ACCESS	DOCTORS	BY INDEX ROWID BATCHED	1	2
INDEX	IDX_DOCTORS_SPECIALTY_LASTNAME	RANGE SCAN	1	1
Access Predicates				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	2

Analysis:

Before adding the index:

- Options: TABLE ACCESS FULL
- Cost: 3

- Session logical reads: 7

After adding the index:

- Options: TABLE ACCESS BY INDEX ROWID BATCHED and INDEX RANGE SCAN
- Cost: 2
- Session logical reads: 2

Summary: The implementation of the composite index on the specialty and last_name columns has had a profound impact on query efficiency, evidenced by the reduction in session logical reads from 7 to just 2. This reduction signifies that the database engine can locate and retrieve the relevant records more directly, avoiding the need to scan the entire table. Additionally, the query's cost metric, which is an indicator of the computational effort required to execute the query, dropped from 3 to 2, indicating a more optimized query plan. The switch from "TABLE ACCESS FULL" to "TABLE ACCESS BY INDEX ROWID BATCHED" and "INDEX RANGE SCAN" also highlights the index's effectiveness, as these methods are generally more efficient in fetching specific records. Overall, this clearly demonstrates the effectiveness of the composite index in enhancing the query performance while justifying the slight overhead in update operations.

2) Single column index on the 'phone' column of the "patients" table.

Purpose: Given that phone numbers serve as unique identifiers within the "**patients**" table, an index on this column will significantly expedite data retrieval operations. The unique nature of the phone numbers ensures high selectivity, making a single column index the optimal choice for this specific query. This enhances lookup speed and minimizes the I/O operations required.

Example Query:

SELECT * FROM patients

WHERE phone LIKE '813-685-8261';

Results:

	PATIENT_ID	FIRST_NAME	LAST_NAME	BIRTH	GENDER	EMAIL	PHONE	STREET	CITY	STATE	ZIP_CODE
1	6442432203	Kelci	Wickling	62-12-13	Female	kwickling1c@gravatar.com	813-685-8261	4566 6th Crossing	Tampa	Florida	33673

Before adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				9
TABLE ACCESS	PATIENTS	FULL	1	9
Filter Predicates				
Other XML				
{info}				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	31

After adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
TABLE ACCESS	PATIENTS	BY INDEX ROWID BATCHED	1	2
INDEX	IDX_PATIENTS_PHONE	RANGE SCAN	1	1
Access Predicates				
PHONE='813-685-8261'				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	4

Analysis:

Before adding the index:

- Options: TABLE ACCESS FULL
- Cost: 9
- Session logical reads: 31

After adding the index:

- Options: TABLE ACCESS BY INDEX ROWID BATCHED and INDEX RANGE SCAN
- Cost: 2
- Session logical reads: 4

Summary: The implementation of a single column index on the phone column has had a dramatic positive effect on the query's performance. Prior to indexing, the query was reliant on a full table scan, which had a computational cost of 9 and necessitated 31 session logical reads. Post-indexing, the query shifted to a more efficient "TABLE ACCESS BY INDEX ROWID BATCHED" and "INDEX RANGE SCAN," slashing the cost to just 2 and reducing the session logical reads to a mere 4. This improvement indicates that the index enables the database engine to locate the relevant record with far greater speed and precision. Overall, the index's impact has significantly bolstered the query's efficiency, justifying the minimal latency introduced during write operations for index maintenance.

3) Composite index on the 'city' and 'birth' columns of the "patients" table.

Purpose: If we want to increase the performance of queries focus on both geographical (city) and demographic (age) attributes, a composite index is most fitting. For the '**city**' column, an index optimizes filtering operations, especially important when the database holds a diverse set of geographical locations. Additionally, the '**birth**' column is crucial for age computation and thus, range-based filtering. While age is not stored but calculated on-the-fly, indexing the '**birth**' column enables quicker computation and filtering for the range condition. The composite nature of this index thereby streamlines the query, optimizing based on both '**city**' and '**birth**' attributes.

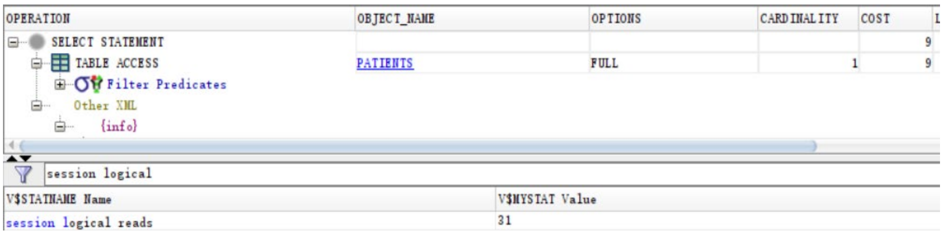
Example Query:

SELECT first_name, last_name,
EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM birth) AS age
FROM patients
WHERE city LIKE 'Tampa' AND
EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM birth) BETWEEN 30 AND 40;

Results:

	FIRST_NAME	LAST_NAME	AGE
1	Ofilia	Laydel	33
2	Rex	Kalinsky	31
3	Luke	Vowell	30
4	Deerdre	Dymidowicz	30
5	Murry	Bartolomeotti	38
6	Kasper	Hiddersley	31
7	Iolande	Scrivinor	39
8	Peadar	Scurman	33
9	Tadio	Elleton	36
10	Josselyn	De Vaan	33
11	Slade	Duxfield	38
12	Kellyann	Youhill	30
13	Ray	Buttel	39
14	Lay	Vaudrey	39
15	Roxie	Bentson	32
16	Catharine	Dispencer	33

Before adding index:

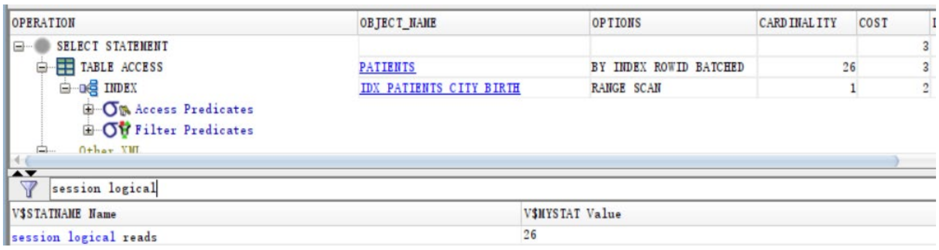


OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				9
TABLE ACCESS	PATIENTS	FULL	1	9
Filter Predicates				
Other XML				
{info}				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	31

After adding index:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				3
TABLE ACCESS	PATIENTS	BY INDEX ROWID BATCHED	26	3
INDEX	IDX_PATIENTS_CITY_BIRTH	RANGE SCAN	1	2
Access Predicates				
Filter Predicates				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	26

Analysis:

Before adding the index:

- Options: TABLE ACCESS FULL
- Cost: 9
- Session logical reads: 31

After adding the index:

- Options: TABLE ACCESS BY INDEX ROWID BATCHED and INDEX RANGE SCAN
- Cost: 3
- Session logical reads: 26

Summary: The creation of a composite index on the city and birth columns has produced a noticeable improvement in the query's performance, particularly in terms of computational cost and execution strategy. Before the index was in place, the query resorted to a full table scan with a computational cost of **9** and required **31** session logical reads. After implementing the index, the query's computational cost decreased to **3**, and the session logical reads were slightly reduced to **26**. Additionally, the execution strategy shifted to the more efficient "TABLE ACCESS BY INDEX ROWID BATCHED" and "INDEX RANGE SCAN" methods. While the improvement in session logical reads is modest, likely due to the range-based nature of the age condition, the index still offers a more optimized query plan and reduced computational cost. Overall, the composite index has demonstrated its utility in enhancing query efficiency, with the benefits outweighing the minimal impact on write operations.

4) Single column index on the ‘symptoms’ column of the “diagnoses” table.

Purpose: If we want to improve the performance of the queries that aim to identify instances of the symptom "Muscle Pain," which constitutes a small fraction (1.2%) of the dataset. Given the high selectivity of this query, an index on the‘symptoms’ column is crucial for performance optimization. Although the query appears to be a point query, the high selectivity necessitates an approach more akin to that for range queries, making the use of an index especially beneficial.

Example Query:

SELECT *

FROM diagnoses
WHERE symptoms LIKE 'Muscle pain';

Results:

	DIAGNOSE_ID	PATIENT_ID	DOCTOR_ID	PRESCRIPTION_ID	TREATMENT_ID	SYMPTOMS
1	50105	5841485075	1014	12105	13105	Muscle pain
2	50113	2368954716	1011	12113	13113	Muscle pain
3	50121	8692605824	1011	12121	13121	Muscle pain
4	50129	3204364933	1023	12129	13129	Muscle pain
5	50137	360130143	1023	12137	13137	Muscle pain
6	50145	1633216772	1023	12145	13145	Muscle pain
7	50153	7442945988	1014	12153	13153	Muscle pain
8	50161	8741657772	1014	12161	13161	Muscle pain
9	50169	6020383938	1014	12169	13169	Muscle pain
10	50177	2864042002	1014	12177	13177	Muscle pain
11	50185	7009002320	1014	12185	13185	Muscle pain
12	50193	6812483893	1011	12193	13193	Muscle pain

Before adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				5
TABLE ACCESS	DIAGNOSES	FULL	12	5
Filter Predicates				
Other XML				
{info}				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	16

After adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				3
TABLE ACCESS	DIAGNOSES	BY INDEX ROWID BATCHED	12	3
INDEX	IDX_DIAGNOSES_SYMPTOMS	RANGE SCAN	12	1
Access Predicates				
Other XML				
{info}				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	3

Analysis:

Before adding the index:

- Options: TABLE ACCESS FULL
- Cost: 5
- Session logical reads: 16

After adding the index:

- Options: TABLE ACCESS BY INDEX ROWID BATCHED and INDEX RANGE SCAN
- Cost: 3
- Session logical reads: 3

Summary: The implementation of a single column index on the 'symptoms" column has significantly optimized this highly selective query. Originally requiring a computational cost of 5 and 16 session logical reads, the query underwent a full table scan. After adding the index, the computational cost dropped to 3, and the session logical reads dramatically reduced to just 3. Furthermore, the execution plan shifted from a less efficient "TABLE ACCESS FULL" to a more effective "TABLE ACCESS BY INDEX ROWID BATCHED" and "INDEX RANGE SCAN." Even though the query is a point query in definition, its high selectivity, capturing only 1.2% of the total dataset, makes the performance considerations akin to those of range queries. Consequently, the index plays an even more critical role in enhancing the query's efficiency.

This strategy has proven to be highly effective, reducing both the computational cost and the I/O operations, and justifies any minor overhead introduced during write operations on the "**diagnoses**" table.

5) A covering composite index on the 'doctor_id', 'first_name', and 'last_name' columns of "Doctors" table; Single column index on the 'doctor_id' column of "Diagnoses" table.

Purpose: If we want to improve the performance of queries that using **JOIN** operation to connect to other tables, and considering the cardinality and the selectivity, then a composite index can handle both single column lookups and combined column searches effectively. With a cardinality of 32, even though the **DOCTORS** table is relatively small, allowing a full scan of the index, bypassing the need for a full table scan. This enhances the performance of **JOIN** operations and groupings in our query. Additionally, given the table size and the query's core operation of joining based on the 'doctor_id', a single column index is appropriate. This index optimizes the table for fast and efficient lookups.

Example Query:

```
SELECT
    d.doctor_id,
    d.first_name || ' ' || d.last_name AS doctor_name,
    COUNT(*) AS diagnosis_count
FROM doctors d
JOIN diagnoses diag ON d.doctor_id = diag.doctor_id
GROUP BY d.doctor_id, d.first_name, d.last_name
ORDER BY doctor_name;
```

Results:

	DOCTOR_ID	DOCTOR_NAME	DIAGNOSIS_COUNT
1	1011	Alexio Skinner	37
2	1030	Amabelle Bubeer	17
3	1020	Bertie de Keep	35
4	1033	Chris McLellan	37
5	1023	Clywd Dagworthy	29
6	1042	David Lopez	38
7	1026	Dillon Jaulmes	44
8	1017	Dorelle Dignam	27
9	1037	Emily Thompson	29
10	1034	Eveleen Elie	36
11	1018	Gaile Pischel	33
12	1019	Garreth Spenton	28
13	1022	Hendrick McKoy	26
14	1021	Isadora Petriello	30

Before adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				10
SORT		ORDER BY	32	10
HASH		GROUP BY	32	10
MERGE JOIN			1000	8
TABLE ACCESS	DOCTORS	BY INDEX ROWID	32	2
INDEX	SYS_C00151606	FULL SCAN	32	1
SORT		JOIN	1000	6
Access Predicates				
Filter Predicates				
TABLE ACCESS	DIAGNOSES	FULL	1000	5
Other XML				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	20

After adding index:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	L
SELECT STATEMENT				6	
SORT		ORDER BY	32	6	
HASH		GROUP BY	32	6	
HASH JOIN			1000	4	
Access Predicates					
INDEX	IDX_DOCTORS_COVERING	FULL SCAN	32	1	
INDEX	IDX_DIAGNOSES_DOCTOR_ID	FAST FULL SCAN	1000	3	

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	8

Analysis:

Before adding the index:

- **Operation Method**: Utilized a "MERGE JOIN" with a full table scan on both DOCTORS and DIAGNOSES tables.
- **Doctors Table**: The access mechanism was "BY INDEX ROWID" with a full scan of an unnamed index (SYS_C0015606), which might not be optimally designed for this specific query.
- **Diagnoses Table**: Conducted a full table scan without leveraging any index.
- **Performance Metrics**:
    - **Cost**: 10
    - **Session Logical Reads**: 20

After adding the index:

- **Operation Method**: Utilized a "HASH JOIN", which is typically faster for larger datasets as it leverages in-memory hash structures.
- **Doctors Table**: A "FULL SCAN" was conducted on a created and optimized index named IDX_DOCTORS_COVERING. This index is tailored for the query, covering all required columns (doctor_id, first_name, and last_name).
- **Diagnoses Table**: Performed a "FAST FULL SCAN" on another tailored index named IDX_DIAGNOSES_DOCTOR_ID, which is designed for efficient lookups based on the doctor_id.
- **Performance Metrics**:
    - **Cost**: 6
    - **Session Logical Reads**: 8

Summary: After introducing tailored indexes (IDX_DOCTORS_COVERING for the DOCTORS table and IDX_DIAGNOSES_DOCTOR_ID for the DIAGNOSES table), the query execution showcased a significant improvement in performance. The use of these indexes shifted the join method from a "MERGE JOIN" to a more efficient "HASH JOIN", resulting in a 40% reduction in cost and a 60% decrease in session logical reads. This optimization demonstrates the pivotal role indexes play in enhancing database performance by streamlining data access patterns and reducing resource consumption. Periodic maintenance and evaluation of these indexes are crucial to sustaining this optimized performance.

6) Several Indexes on different table:

For the Patients table: A covering composite index on the patient_id, first_name, and last_name columns.

For the Diagnoses table: A covering composite index on the patient_id, symptoms,

treatment_id, and prescription_id columns.
For the Treatments table: A single-column index on the treatment_id.
For the Prescription table: A single-column index on the prescription_id.

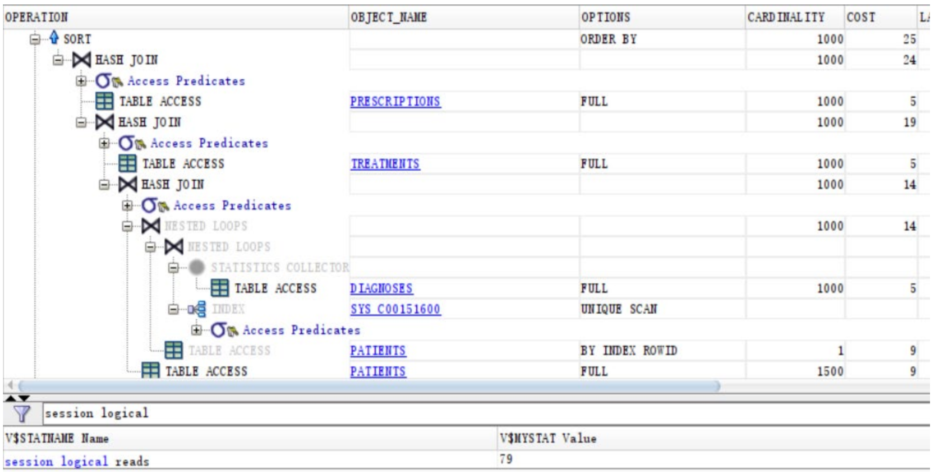
Purpose: The composite index allows efficient retrieval of patients' details and aids in JOIN operations with the diagnoses table. Given that patient information is crucial, quick access is essential, making the covering index a strategic choice. The covering index facilitates JOIN operations with treatments and prescriptions tables, thus enhancing query performance. The index minimizes the need for table scans when filtering based on patient_id or symptoms. Additionally, this single-column index assists in JOIN operations with the diagnoses table. Given that treatment data will be frequently accessed in correlation with diagnoses, an index on treatment_id ensures faster retrieval. Similar with IDX_TREATMENTS_COVERING, the single-column index helps in JOIN operations and enhances query performance by reducing the lookup time for prescriptions associated with specific diagnoses.
Example Query:

```
SELECT
    p.first_name,
    p.last_name,
    di.symptoms,
    t.treatment_name,
    pr.medication_name
FROM patients p
JOIN diagnoses di ON p.patient_id = di.patient_id
JOIN treatments t ON di.treatment_id = t.treatment_id
JOIN prescriptions pr ON di.prescription_id = pr.prescription_id
ORDER BY p.last_name, p.first_name;
```

Results:

	FIRST_NAME	LAST_NAME	SYMPTOMS	TREATMENT_NAME	MEDICATION_NAME
1	Aurelea	Abdey	Diarrhea	Probiotics	Promethazine
2	Maurits	Acutt	Osteoporosis	Dietary management	Metformin
3	Wilfrid	Acutt	Sore throat	Salt water gargling	Acetaminophen
4	Kinnie	Adamek	Tenderness	Rest	Naproxen
5	De witt	Adamson	Fever	Over-the-counter fever r...	Acetaminophen
6	Emeline	Ainsley	Chronic bronchitis	Chest physical therapy	Prednisone
7	Hinze	Alleyne	Swelling around a joint	Joint Aspiration	Ibuprofen
8	Klarrisa	Alliott	Acromegaly	Lifestyle changes	Birth control pills
9	Bing	Alred	Sore throat	Throat lozenges or sprays	Throat lozenges or
10	Denise	Alvar	COPD	Oxygen therapy	Tiotropium
11	Bea	Ambrose	Strabismus	Corrective eyewear	Eyeglasses

Before adding index:

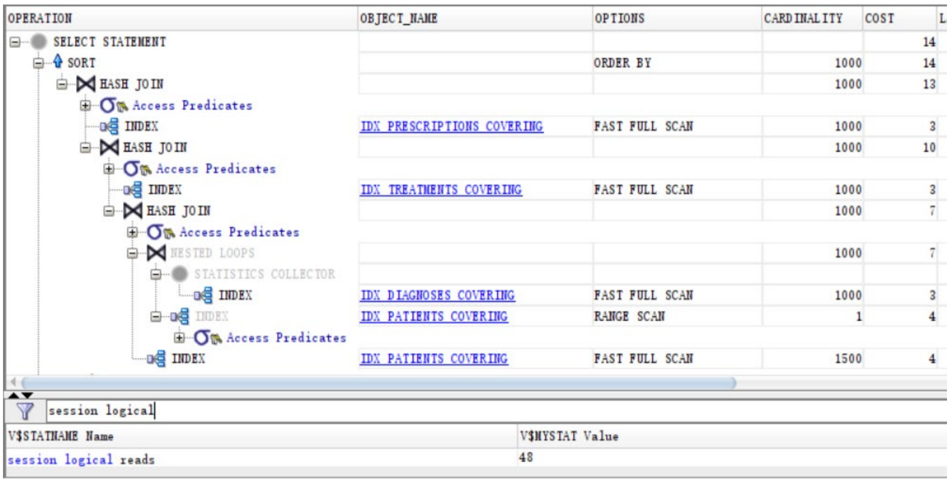


OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	L
SORT		ORDER BY	1000	25	
HASH JOIN			1000	24	
Access Predicates					
TABLE ACCESS	PRESCRIPTIONS	FULL	1000	5	
HASH JOIN			1000	19	
Access Predicates					
TABLE ACCESS	TREATMENTS	FULL	1000	5	
HASH JOIN			1000	14	
Access Predicates					
NESTED LOOPS			1000	14	
NESTED LOOPS					
STATISTICS COLLECTOR					
TABLE ACCESS	DIAGNOSES	FULL	1000	5	
INDEX	SYS_C00151600	UNIQUE SCAN			
Access Predicates					
TABLE ACCESS	PATIENTS	BY INDEX ROWID	1	9	
TABLE ACCESS	PATIENTS	FULL	1500	9	

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	79

After adding index:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	L
SELECT STATEMENT				14	
SORT		ORDER BY	1000	14	
HASH JOIN			1000	13	
Access Predicates					
INDEX	IDX_PRESCRIPTIONS_COVERING	FAST FULL SCAN	1000	3	
HASH JOIN			1000	10	
Access Predicates					
INDEX	IDX_TREATMENTS_COVERING	FAST FULL SCAN	1000	3	
HASH JOIN			1000	7	
Access Predicates					
NESTED LOOPS			1000	7	
STATISTICS COLLECTOR					
INDEX	IDX_DIAGNOSES_COVERING	FAST FULL SCAN	1000	3	
INDEX	IDX_PATIENTS_COVERING	RANGE SCAN	1	4	
Access Predicates					
INDEX	IDX_PATIENTS_COVERING	FAST FULL SCAN	1500	4	

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	48

Analysis:

Before adding the index:

- Sort Operation: The data was sorted using an "ORDER BY" clause, incurring a cost of 24 with a cardinality of 1000.
- Prescriptions Table: A full table scan was performed with a cost of 5.
- Treatments Table: Again, a full table scan was conducted, resulting in a cost of 5.
- Diagnoses Table: An index was utilized (SYS_C0015600) for a unique scan, carrying a cost of 5.
- Patients Table: A full table scan was carried out after accessing the data by an "INDEX ROWID". The cost was 9 with a high cardinality of 1500.
- Total Session Logical Reads: 79

After adding the index:

- Select Statement: A "HASH JOIN" was utilized, with an overall reduced cost of 14.
- Prescriptions Table: The custom index "IDX_PRESCRIPTIONS_COVERING" was utilized for a "FAST FULL SCAN" which reduced the cost to 3.
- Treatments Table: The custom index "IDX_TREATMENTS_COVERING" was applied for a "FAST FULL SCAN", resulting in a more efficient cost of 3.
- Diagnoses Table: The custom index "IDX_DIAGNOSES_COVERING" was employed for a "FAST FULL SCAN" and achieved a reduced cost of 3.

- Patients Table: Two different scans were conducted on this table. Firstly, the custom index "IDX_PATIENTS_COVERING" was utilized for a "RANGE SCAN" with a cost of 3. Subsequently, another "FAST FULL SCAN" was carried out with the same index, at a cost of 4.
- Total Session Logical Reads: 48

Summary: After the strategic introduction of custom indexes, the execution of the query to retrieve patients and their corresponding diagnoses, treatments, and prescriptions experienced a marked boost in efficiency. Transitioning to a more adept "HASH JOIN" mechanism minimized the reliance on full table scans and leveraged "FAST FULL SCAN" techniques. This resulted in an approximately 39% reduction in session logical reads, highlighting the transformative impact of precise indexing on database performance. Such improvements emphasize the significance of regular indexing plan evaluations to consistently achieve and maintain optimal query performance.

2. Optimizer changes

1) OPTIMIZER_MODE = FIRST_ROWS_1

Purpose: We decided to use the FIRST_ROWS_1 mode for optimizing the plan in such a way that the first row or few rows of the query result set is returned as quickly as possible. This becomes especially significant in situations where immediate data availability is crucial—for example, in real-time decision-making systems or in applications where user experience is dramatically affected by speed, such as dashboards or web pages that need to populate data instantly.

Example Query:

```
SELECT p.FIRST_NAME,
       p.LAST_NAME,
       p.EMAIL
FROM   patients p
       INNER JOIN billing b
              ON p.PATIENT_ID = b.PATIENT_ID
WHERE  FLOOR(MONTHS_BETWEEN(SYSDATE, p.BIRTH)/12) < 18
       AND b.TOTAL_AMOUNT > 850 ;
```

Results:

	FIRST_NAME	LAST_NAME	EMAIL
1	Faber	O' Connell	foconnell2@omniture.com
2	Denis	Hampe	dhampeg@microsoft.com
3	Moritz	Chiles	mchilesx@cyberchimps.com
4	Johnna	Sego	jsego14@ucoz.ru
5	Bernhard	Tivnan	btivnan1j@sogou.com
6	Berky	Joncic	bjoncic11@mysql.com
7	Raynell	Langdridge	rlangdridge1m@scientificamerican.com
8	Annelise	Tofanelli	atofanelli1y@godaddy.com
9	Stevy	Garside	sgarside28@paypal.com
10	Allyce	Furze	afurze2b@dropbox.com

The methods of applying this Technique:

ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_1;

Before applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			75	14
HASH JOIN			75	14
Access Predicates				
TABLE ACCESS	PATIENTS	FULL	75	9
Filter Predicates				
TABLE ACCESS	BILLING	FULL	187	5
Filter Predicates				

After applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	5
NESTED LOOPS			3	5
NESTED LOOPS			3	5
TABLE ACCESS	BILLING	FULL	3	2
Filter Predicates				
INDEX	SYS_C00151651	UNIQUE SCAN	1	0
Access Predicates				
TABLE ACCESS	PATIENTS	BY INDEX...	1	1
Filter Predicates				

Analysis:

Before applying the Technique:

- Operation: USING **HASH JOIN**
- Cost: 14

After applying the Technique:

- Operation: USING **NESTED LOOPS** and **INDEX**
- Cost: 5

Summary:

In the context of performance tuning for a SQL query aimed at quickly identifying minors with high medical bills, we initially employed Oracle's **OPTIMIZER_MODE = FIRST_ROWS_1**. This strategic shift led to a change in the join algorithm used, transitioning from **Hash Join** to **Nested Loops** and **reducing** the optimizer's cost estimate from **14 to 5**. While this approach effectively sped up the retrieval of the initial rows—important in scenarios requiring immediate data, such as healthcare analytics or emergency case assessments—it's worth considering that for **point queries**, **indexing** might generally offer superior performance enhancements. However, it's pertinent to note that when the requirement is to retrieve only a single row or a **very limited set of rows**, using the **FIRST_ROWS_1 optimizer mode** can indeed be efficient. Hence, while indexing could provide a robust, long-term performance gain for a broad range of query types, the optimizer mode change is particularly effective for very specific use-cases requiring rapid retrieval of limited data.

2) OPTIMIZER_MODE = FIRST_ROWS_10

Purpose: We developed a query which involves multiple joins across different tables, and each table contains attributes that are essential for composing a comprehensive patient report, making this query quite complex. The primary performance requirement here is to quickly render the first few rows of data. To meet this requirement, the OPTIMIZER_MODE = FIRST_ROWS_10 is an apt choice. This mode prioritizes the quick retrieval of the first ten rows of the query result set, which is particularly useful in applications that depend on real-time or near-real-time data access, such as a dashboard or a reporting interface that requires immediate display of preliminary results.

Example Query:

```
SELECT p.LAST_NAME,
        b.TOTAL_AMOUNT,
        p.EMAIL,
        p.PHONE,
        d.symptoms,
        t.treatment_name
FROM    patients p
        INNER JOIN appointments a
                ON p.PATIENT_ID = a.PATIENT_ID
        INNER JOIN billing b
                ON p.PATIENT_ID = b.PATIENT_ID
        INNER JOIN diagnoses d
                ON p.PATIENT_ID = d.PATIENT_ID
        INNER JOIN treatments t
                ON d.TREATMENT_ID = t.TREATMENT_ID ;
```

Results:

	LAST_NAME	TOTAL_AMOUNT	EMAIL	PHONE	SYMPTOMS	TREATMENT_NAME
1	Lambie	684.57	alambie6r@arstechnica.com	904-262-6307	Constipation	Stool softeners
2	Bordone	782.95	fbordone6s@miitbeian.gov.cn	786-559-6878	Abdominal pain	Dietary modifica
3	Thaine	492.37	bthaine6t@hubpages.com	407-245-5603	Heartburn	Dietary modifica
4	Jirik	472.66	ljirik6u@flickr.com	850-570-5708	GERD	Dietary modifica
5	Linner	658.83	wlinner6v@mozilla.com	727-947-0041	Ulcers	Antispasmodic me
6	Crinson	783.67	gcrinson6w@godaddy.com	386-564-2389	IBS	Antibiotics (fo
7	Fosdyke	378.15	hfosdyke6x@about.me	561-912-9502	IBD	Biologics
8	Middas	788.32	cmiddas6y@imdb.com	239-672-8629	Nausea	Ginger suppleme
9	Undy	345.94	rundy6z@whitehouse.gov	850-722-1592	Vomiting	Dietary modifica
10	Abdey	996.74	aabdey70@dropbox.com	904-228-9377	Diarrhea	Probiotics
11	Vowell	488.57	lvowell71@newyorker.com	813-999-2580	Constipation	Laxatives
12	Cousens	724.13	kcousens72@wunderground.com	850-938-3486	Abdominal pain	Warm compresses
13	Frankton	590.95	ffrankton73@4shared.com	941-274-3691	Heartburn	H2 blockers
14	Rumbold	419.78	crumbold74@icq.com	850-563-2069	GERD	Elevating the he
15	Beeke	385.65	pbeeke75@creativecommons.org	941-921-4417	Ulcers	Protective agent
16	Wicklin	527.08	pwicklin76@purevolume.com	727-450-0873	IBS	Probiotics
17	Neilus	857.42	mneilus77@un.org	754-180-1411	IBD	Aminosalicylate
18	Stogill	844.57	estogill78@bravesites.com	305-335-0832	Nausea	Antiemetic medi

The methods of applying this Technique:

ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_10;

Before applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			667	25
HASH JOIN			667	25
Access Predicates				
TABLE ACCESS	APPOINTMENTS	FULL	1500	5
HASH JOIN			667	20
Access Predicates				
INDEX	IDX_TREATMENTS_COVERING	FAST FUL...	1000	3
HASH JOIN			667	17
Access Predicates				
INDEX	IDX_DIAGNOSES_COVERING	FAST FUL...	1000	3
HASH JOIN			1000	14
Access Predicates				
NESTED LOOPS			1000	14
NESTED LOOPS				
STATISTICS CO				
TABLE ACCE	BILLING	FULL	1000	5
INDEX	SYS_C00151651	UNIQUE SCAN		
Access Predicates				
TABLE ACCESS	PATIENTS	BY INDEX...	1	9
TABLE ACCESS	PATIENTS	FULL	1500	9
Other XML				

After applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	12
NESTED LOOPS			1	12
NESTED LOOPS			1	7
NESTED LOOPS			1	6
NESTED LOOPS			2	4
TABLE ACCESS	BILLING	FULL	750	2
TABLE ACCESS	PATIENTS	BY INDEX...	1	1
INDEX	SYS_C00151651	UNIQUE SCAN	1	0
Access Predicates				
INDEX	IDX_DIAGNOSES_COVERING	RANGE SCAN	1	1
Access Predicates				
INDEX	IDX_TREATMENTS_COVERING	RANGE SCAN	1	1
Access Predicates				
TABLE ACCESS	APPOINTMENTS	FULL	1	5
Filter Predicates				

Analysis:

Before applying the Technique:

- Operation: USING **HASH JOIN**
- Cost: 25

After applying the Technique:

- Operation: USING **NESTED LOOPS**
- Cost: 12

Summary:

We observe a substantial change in the cost metrics associated with query execution. Before applying the **FIRST_ROWS** optimizer mode, the operation utilized a **HASH JOIN** with a computational cost of **25**. After applying the technique, the query shifted to using **NESTED LOOPS** with a reduced cost to **12**. This cut the **COST** by more than half, corroborating the efficacy of the optimizer mode in rapidly delivering initial rows. However, it's important to note that while the optimizer mode expedited the retrieval of the first rows by altering the join algorithm, this approach might elevate the total query cost for complete datasets and affect other operations such as updates or deletions. Therefore, the trade-offs introduced by the optimizer should be meticulously evaluated within the context of the overall application workload.

3. Table partitioning:

1) Range partitioning on the 'birth' attribute in "Patients" table

Purpose: Give that the data type for birth is DATE and the range spans from the '60s to '22, we decided to create partitions by decade. It will allow Oracle to skip partitions that do not contain relevant data, thereby increasing query performance. The attribute (birth year) used for partitioning is directly involved in the query condition, making partition pruning highly effective. Meanwhile, range partitioning is straightforward to set up and offers better query performance when the partition key is part of the query conditions. The rationale for selecting partitioning by range on birth year is to improve the query speed. Since the birth year is a common filtering condition in our query, range partitioning is a natural fit. The data within each partition is sorted by the range value, making range scans efficient.

Example Query:

SELECT last_name,gender, email, phone

```
FROM patients
WHERE EXTRACT(YEAR FROM birth) BETWEEN 1960 AND 1969
AND city = 'Miami';
```

Results:

	LAST_NAME	GENDER	EMAIL	PHONE
1	Grenfell	Male	tgrenfell7q@com.com	305-608-6445
2	Popescu	Female	spopescugs@ameblo.jp	786-293-2891
3	Cranham	Male	rcranham6a@1688.com	305-399-1871
4	Howle	Male	mhowlenk@prweb.com	305-557-7527
5	Viant	Male	rviant1i@mail.ru	786-831-3995
6	Huddy	Female	dhuddy52@nymag.com	305-109-6659
7	Elder	Male	selderc1@usatoday.com	305-446-6265
8	Dye	Female	vdyeaj@blog.com	786-614-0545
9	Sabater	Male	fsabater11@storify.com	786-461-6009
10	Ulyatt	Female	sulyatte8@dell.com	305-218-5727
11	Camplejohn	Male	gcamplejohndg@yellowpages.com	786-359-7951
12	Roseby	Male	broseby91@tumblr.com	305-999-7071
13	Fradson	Female	sfradsonfc@usda.gov	305-892-6523
14	Robyns	Female	arobyns2g@symantec.com	305-967-3492

The methods of applying this Technique:

Creating the partitioning table:

```
CREATE TABLE patients_birth_year_part (
    patient_id NUMBER,
    last_name VARCHAR2(50),
    birth DATE,
    gender VARCHAR2(10),
    email VARCHAR2(100),
    phone VARCHAR2(15),
    city VARCHAR2(50),
    birth_year NUMBER (4,0)
)
PARTITION BY RANGE (birth_year) (
    PARTITION p60s VALUES LESS THAN (1970),
    PARTITION p70s VALUES LESS THAN (1980),
    PARTITION p80s VALUES LESS THAN (1990),
    PARTITION p90s VALUES LESS THAN (2000),
    PARTITION p00s VALUES LESS THAN (2010),
    PARTITION p10s VALUES LESS THAN (2020),
    PARTITION p20s VALUES LESS THAN (MAXVALUE)
);
```

Insert the Data:

```
INSERT INTO patients_birth_year_part (patient_id, last_name, birth, gender, email, phone, city, birth_year)
SELECT patient_id, last_name, birth, gender, email, phone, city, EXTRACT(YEAR FROM birth)
FROM patients;
COMMIT;
```

Analyze the table:

```
ANALYZE TABLE patients_birth_year_part COMPUTE STATISTICS;
```

The Statistic of Patients table:

1	NUM_ROWS	1500
2	BLOCKS	28
3	AVG_ROW_LEN	115
4	SAMPLE_SIZE	1500
5	LAST_ANALYZED	23-10-28
6	LAST_ANALYZED_SINCE	23-10-28

The partition tables:

	PARTITION_...	LAST_ANALYZED	NUM_ROWS	BLOCKS	SAMPLE_SIZE	HIGH_VALUE
1	P00S	23-10-28	189	124	189	2010
2	P10S	23-10-28	55	124	55	2020
3	P20S	23-10-28	21	124	21	MAXVALUE
4	P60S	23-10-28	316	124	316	1970
5	P70S	23-10-28	286	250	286	1980
6	P80S	23-10-28	314	250	314	1990
7	P90S	23-10-28	319	124	319	2000

Before applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				9
TABLE ACCESS	PATIENTS	FULL	32	9
Filter Predicates				
Other XML				

session logical

V$STATNAME Name	V$MYSTAT Value
session logical reads	36

After applying the Technique:

SELECT last_name, gender, email, phone

FROM **PATIENTS_BIRTH_YEAR_PART**

Where **birth_year** between 1960 and 1969

and city = 'Miami';

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	PARTITION_START	COST	PARTITION_STOP
SELECT STATEMENT					35	
PARTITION RANGE		SINGLE	32	1	35	1
TABLE ACCESS	PATIENTS_BIRTH_YEAR_PART	FULL	32	1	35	1
Filter Predicates						
Other XML						

session l

V$STATNAME Name	V$MYSTAT Value
session logical reads	75

Change the query to the range of year in 1960-1975:

SELECT last_name, gender, email, phone

FROM **PATIENTS_BIRTH_YEAR_PART**

Where **birth_year** between 1960 and 1975

and city = 'Miami';

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	PARTITION_START	PARTITION_STOP
SELECT STATEMENT				103		
PARTITION RANGE		ITERATOR	60	103	1	2
TABLE ACCESS	PATIENTS_BIRTH_YEAR_PART	FULL	60	103	1	2
Filter Predicates						
Other XML						

Analysis:

Before applying the Technique:

- Operation: **TABLE ACCESS FULL**
- Cost: 9
- Session logical reads: 36

After applying the Technique:

- Operation: **PARTITON RANGE SINGLE or INTERATOR**
- Cost: 35
- Session logical reads: 75

Summary: In the application of range **partitioning** to optimize query performance, the observed results were paradoxical. Despite the anticipation that partitioning would reduce query costs, both the **COST and Session Logical Reads increased** substantially compared to running the query on the non-partitioned table. This discrepancy can be primarily attributed to an unexpectedly high number of **BLOCKS** in each partition, rendering the current **COST and Session Logical Reads** metrics **unreliable** for this analysis. While this warrants further investigation, it's important to note that the SQL optimizer did engage the **partitioning** mechanism during query execution, as evidenced by the **EXPLAIN PLAN**. Thus, while the current metrics are not ideal, there is an underlying belief that partitioning will prove beneficial in larger databases where the impact of partition pruning becomes more pronounced.

4. Parallel Execution (PX):

1) Setting Degree of Parallelism at "appointments" Table.

Purpose: We want to set a default DOP of 4 for the appointments table. This means any query run against this table will, by default, employ 4 parallel processes unless overridden by a session or query level parallel hint. The table appointments could potentially contain a large dataset, particularly for extensive date ranges. When performing full-table scans or range queries, parallel execution can significantly reduce the time taken. By setting the Degree of Parallelism (DOP) to 4 at the table level, it will allow Oracle's query optimizer to automatically use parallel processing for operations on this table, thereby potentially improving query performance.

Example Query:

SELECT * FROM appointments
WHERE appointment_date
BETWEEN TO_DATE('2023-08-01', 'YYYY-MM-DD')
AND TO_DATE('2023-08-10', 'YYYY-MM-DD');

Results:

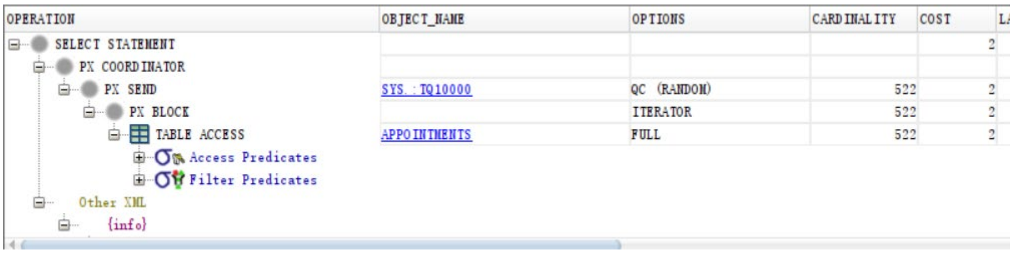
	APPOINTM...	PATIENT_ID	APPOINTMENT_DATE	STATUS
1	60421	5478798665	23-08-06	Completed
2	60428	4802279574	23-08-04	Completed
3	60432	6283989071	23-08-01	Completed
4	60441	3477055632	23-08-02	Completed
5	60453	2087781931	23-08-03	Completed
6	60458	9365145147	23-08-07	Completed
7	60459	2621544568	23-08-01	Completed
8	60460	8590646289	23-08-09	Completed
9	60467	4511266107	23-08-04	Completed
10	60473	5588640233	23-08-04	Completed
11	60475	9639791016	23-08-05	Completed
12	60477	3439113310	23-08-02	Completed
13	60481	5814541148	23-08-04	Completed
14	60484	4250774821	23-08-02	Completed
15	60489	271153547	23-08-04	Completed
16	60490	7465740770	23-08-05	Completed

The methods of applying this Technique:
ALTER TABLE appointments PARALLEL (DEGREE 4);

Before applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				5
TABLE ACCESS	APPOINTMENTS	FULL	522	5
Filter Predicates				
Other XML				
{info}				
info type="db_version"				
12.1.0.2				
info type="parse_schema"				
"SQL188"				

After applying the Technique:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	L
SELECT STATEMENT				2	
PX COORDINATOR					
PX SEND	SYS.:TQ10000	QC (RANDOM)	522	2	
PX BLOCK		ITERATOR	522	2	
TABLE ACCESS	APPOINTMENTS	FULL	522	2	
Access Predicates					
Filter Predicates					
Other XML					
{info}					

Analysis:
Before applying the Technique:

- Operation: **TABLE ACCESS FULL**
- Cost: 5

After applying the Technique:

- Operation: By using **PX steps**
- Cost: 2

Summary:
In our case of querying appointments for a specific date range, implementing a **Degree of Parallelism (DOP) of 4** at the table level exhibited remarkable improvements in query performance. The execution plan transitioned from a **"TABLE ACCESS FULL"** operation to utilizing **"PX steps,"** culminating in a **reduction** of query **cost** from **5 to 2**. However, it's imperative to underline that this performance gain comes with its own set of trade-offs. Specifically, the 'Session logical reads' metric escalated from 4 to 57, which serves as a testament to the overhead introduced by parallel execution.
While the lower query cost demonstrates the efficacy of parallelism, we also need to be cognizant of its impact on resource utilization. The technique increased CPU and I/O consumption, which could interfere with other database operations, especially during periods of high demand. Additionally, potential complications in locking mechanisms could affect the concurrency of INSERT, UPDATE, or DELETE operations.
As we navigate through increasingly complex query structures, attention to transitions between parallel and serial execution steps will become indispensable for fine-tuning performance. These transitions could very well be the linchpin for future performance enhancements, and therefore, warrant meticulous monitoring and scrutiny.

2) Setting Degree of Parallelism at Multiple Table.
Purpose: We decided to set a default DOP of 4 for the eight tables that are “patients”,

“appointments”, “billing, diagnoses”, “treatments”, “doctors”, “insurance”, and “departments” table. This means the query involves multiple JOIN operations and aggregation functions like MAX, parallel execution will likely improve query performance. Since the size of some table is large (ranging from 32 rows to 1,500 rows), parallel execution can expedite data retrieval. By dividing the tasks into smaller chunks, Oracle can work on these tasks simultaneously, reducing the overall execution time.

Example Query:

```
SELECT p.LAST_NAME AS "Patient's Last Name",
       p.FIRST_NAME AS "Patient's First Name",
     d.FIRST_NAME || ' ' || d.LAST_NAME AS "Doctor's Name",
       d.specialty AS DOCTOR_SPECIALTY,
       dep.name AS DEPARTMENT_NAME,
       MAX(b.TOTAL_AMOUNT) AS HIGHEST_BILL,
       i.PROVIDER AS INSURANCE_PROVIDER,
       MAX(a.appointment_DATE) AS LATEST_APPOINTMENT,
       t.TREATMENT_NAME
FROM     patients p
       INNER JOIN appointments a ON p.PATIENT_ID = a.PATIENT_ID
       INNER JOIN billing b ON p.PATIENT_ID = b.PATIENT_ID
       INNER JOIN diagnoses dgn ON p.PATIENT_ID = dgn.PATIENT_ID
       INNER JOIN treatments t ON dgn.TREATMENT_ID = t.TREATMENT_ID
       INNER JOIN doctors d ON dgn.DOCTOR_ID = d.DOCTOR_ID
       INNER JOIN departments dep ON d.DEPARTMENT_CODE =
dep.DEPARTMENT_CODE
       LEFT JOIN insurance i ON p.PATIENT_ID = i.PATIENT_ID
GROUP   BY p.LAST_NAME, p.FIRST_NAME, d.SPECIALTY, dep.NAME,
d.FIRST_NAME || ' ' || d.LAST_NAME, i.PROVIDER, t.TREATMENT_NAME
ORDER   BY p.LAST_NAME;
```

Results:

	Patient's Last Name	Patient's First ...	Doctor's Name	DOCTOR_SPECIALTY	DEPARTMENT_NAME	HIGHEST_BILL	INSURANCE_PROVIDER	LATEST_APPOINTMENT	TREATMENT_NAME
1	Abdey	Aurelea	David Lopez	Gastroent...	Gastroente...	996.74	BCBS	23-08-03	Probiotics
2	Acutt	Maurits	Amabelle B...	Internist	Internal M...	709.79	UHC	23-08-04	Dietary management
3	Acutt	Wilfrid	Ivar Basnett	Pediatrician	Pediatrics	665.64	AET	23-09-03	Salt water gargling
4	Adamek	Kinnie	Clywd Dagw...	General	Orthopedics	295.95	UHC	23-08-25	Rest
5	Adamson	De witt	Sonnie The...	General	Pediatrics	455.57	CI	23-09-26	Over-the-counter fever :
6	Ainsley	Emeline	Urban Blan...	General	Respiratory	235.85	KP	23-09-04	Chest physical therapy
7	Alleyne	Hinze	Clywd Dagw...	General	Orthopedics	672.87	CI	23-08-13	Joint Aspiration
8	Alliott	Klarrisa	Sarah Kim	Endocrino...	Endocrinol...	258.87	UHC	23-09-09	Lifestyle changes
9	Alred	Bing	Wilma Klouz	Pediatrician	Pediatrics	829.67	BCBS	23-09-06	Throat lozenges or spra
10	Alvar	Denise	Eveleen Elie	Pulmonolo...	Respiratory	670.14	KP	23-09-23	Oxygen therapy
11	Ambrose	Bea	Dillon Jau...	Ophthalmo...	Ophthalmol...	227.18	BCBS	23-09-16	Corrective eyewear
12	Amer	Morie	Marlie Chr...	Internist	Internal M...	818.04	USAA	23-09-25	Allergy shots (immunoth
13	Amps	Jarib	David Lopez	Gastroent...	Gastroente...	884.48	CNC	23-08-11	Dietary modifications
14	Andrichak	Reagan	Gaile Pischel	Dermatolo...	Dermatology	288.72	BCBS	23-08-04	Minoxidil (Rogaine)
15	Andries	Avram	Clywd Dagw...	General	Orthopedics	520.75	UHC	23-09-05	Rest and Modified Activ:
16	Antognozzii	Rogers	Gaile Pischel	Dermatolo...	Dermatology	909.18	CI	23-09-14	Topical corticosteroids
17	Aronstam	Kennan	Oralee Cat...	Orthopedist	Orthopedics	499.86	CNC	23-09-07	Surgery
18	Arton	Michelle	Sarah Kim	Endocrino...	Endocrinol...	355.48	AET	23-08-26	Insulin pump therapy
19	Ascraft	Nady	Oralee Cat...	Orthopedist	Orthopedics	567.64	BCBS	23-08-10	Physical Therapy
20	Aslin	Paulina	Dorelle Di...	Ophthalmo...	Ophthalmol...	458.72	UHC	23-08-05	Corrective eyewear

The methods of applying this Technique:

```
ALTER TABLE patients PARALLEL (DEGREE 4);
ALTER TABLE appointments PARALLEL (DEGREE 4);
ALTER TABLE billing PARALLEL (DEGREE 4);
```

ALTER TABLE diagnoses PARALLEL (DEGREE 4);
ALTER TABLE treatments PARALLEL (DEGREE 4);
ALTER TABLE doctors PARALLEL (DEGREE 4);
ALTER TABLE insurance PARALLEL (DEGREE 4);
ALTER TABLE departments PARALLEL (DEGREE 4);

Before applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				34
SORT		GROUP BY	667	34
HASH JOIN			667	33
Access Predicates				
TABLE ACCESS	APPOINTMENTS	FULL	1500	5
HASH JOIN		RIGHT OUTER	667	28
Access Predicates				
TABLE ACCESS	INSURANCE	FULL	1000	5
HASH JOIN			667	23
Access Predicates				
INDEX	IDX_TREATMENTS_COVERING	FAST FUL...	1000	3
HASH JOIN			667	20
Access Predicates				
TABLE ACCESS	BILLING	FULL	1000	5
HASH JOIN			1000	15
Access Predicates				
INDEX	IDX_PATIENTS_COVERING	FAST FUL...	1500	4
HASH JOIN			1000	11
Access Predicates				
MERGE JOI			32	6
TABLE	DEPARTMENTS	BY INDEX...	10	2
IND	SYS_C00151648	FULL SCAN	10	1
SORT		JOIN	32	4
Access Predicates				
Filter Predicates				
TAB	DOCTORS	FULL	32	3
TABLE ACC	DIAGNOSES	FULL	1000	5
Other XML				

After applying the Technique:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			667	18
PX COORDINATOR				
PX SEND	SYS.:TQ10012	QC (ORDER)	667	18
SORT		GROUP BY	667	18
PX RECEIVE			667	18
PX SEND	SYS.:TQ10011	RANGE	667	18
HASH		GROUP BY	667	18
HASH JOIN			667	17
Access Predicates				
PX RECEIVE			1500	2
PX SEND	SYS.:TQ10008	HASH	1500	2
PX BLOCK		ITERATOR	1500	2
TABLE ACCESS	APPOINTMENTS	FULL	1500	2
HASH JOIN		RIGHT OUTER	667	15
Access Predicates				
PX RECEIVE			1000	2
PX SEND	SYS.:TQ10009	HASH	1000	2
PX BLOCK		ITERATOR	1000	2
TABLE ACCESS	INSURANCE	FULL	1000	2
PX RECEIVE			667	13
PX SEND	SYS.:TQ10010	HASH	667	13
HASH JOIN		BUFFERED	667	13
Access Predicates				

Analysis:

Before applying the Technique:

- Operation: Using **HASH JOIN Only**
- Cost: 34

After applying the Technique:

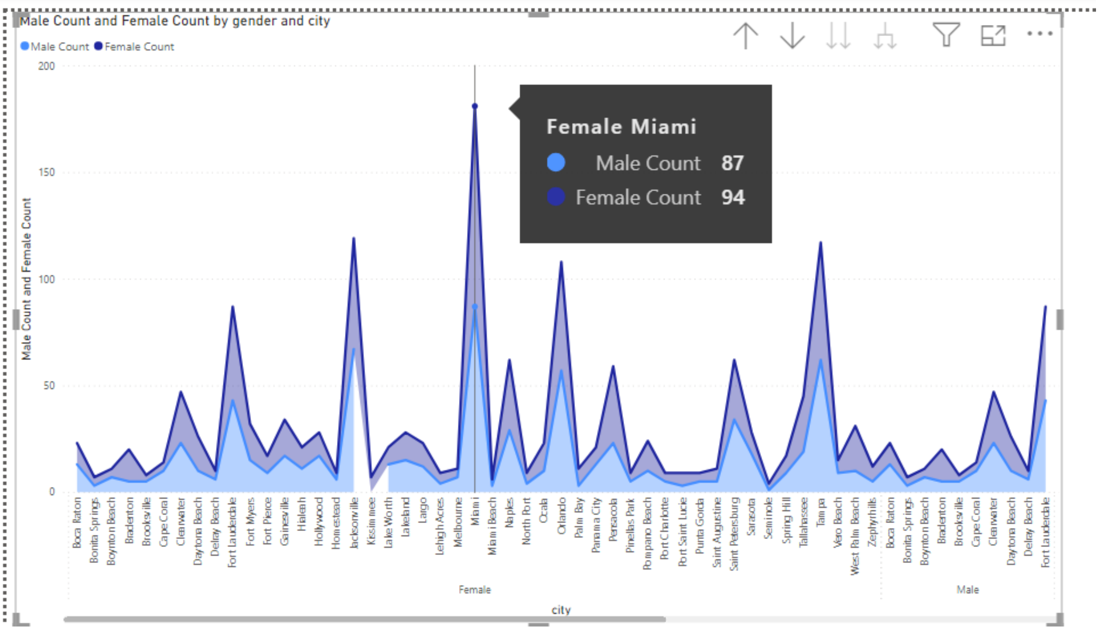
- Operation: By using **PX steps and the HASH JOIN**
- Cost: 18

Summary: Implementing parallel execution yielded a significant performance improvement. The overall query **cost** was reduced from **34** to **18**, indicating a more efficient data retrieval process. However, it's crucial to note that parallel execution incurs startup, coordination, and shutdown costs (**increasing the I/O**). These overheads were not directly measured but must be considered for a comprehensive performance assessment.
Moreover, the query's complexity opens avenues for further optimization, especially around execution step transitions. These transitions, such as from serial-to-parallel or parallel-to-serial, serve as indicators for tweaking possible execution strategies and warrant scrutiny for future performance enhancements.

Other Topics

1. Data Visualization

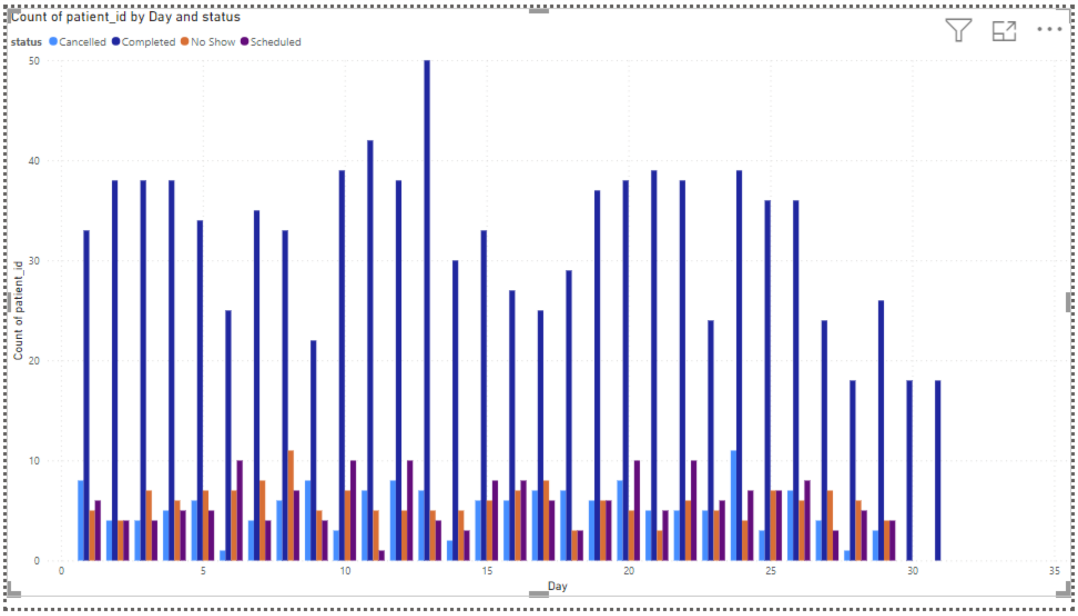
1) Gender Demographics Analysis in Florida Cities



Analysis:
The Power BI report shows the total male and female count for each city in the Florida. The cities are listed on the x-axis, and the male and female counts are shown on the y-axis. The report also shows a line graph that shows the average male and female count for each city.
The cities with the highest male counts are Fort Pierce, Punta Gorda, and Cape Coral. The cities with the highest female counts are Spring Hill, Port St. Lucie, and North Port.
The report can be used to identify cities with a high or low male or female population. It can also be used to track changes in the male and female population over time.
Additional insights

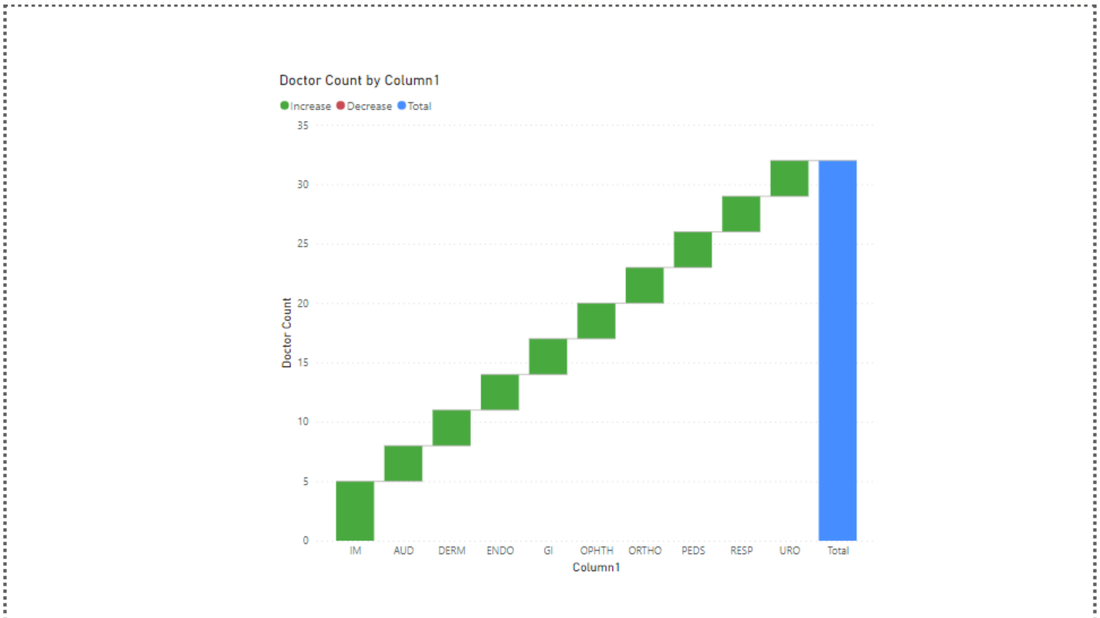
- The city with the highest total male and female count is Miami, Florida.
- The city with the lowest total male and female count is Vero Beach, Florida.

2) Appointment Status Distribution by Day



Analysis:
This Power BI report shows the relationship between the appointments in a day and counts of patients for that day.
The report shows that on 13 Aug 2023, there are highest number of patients (i.e., 50)

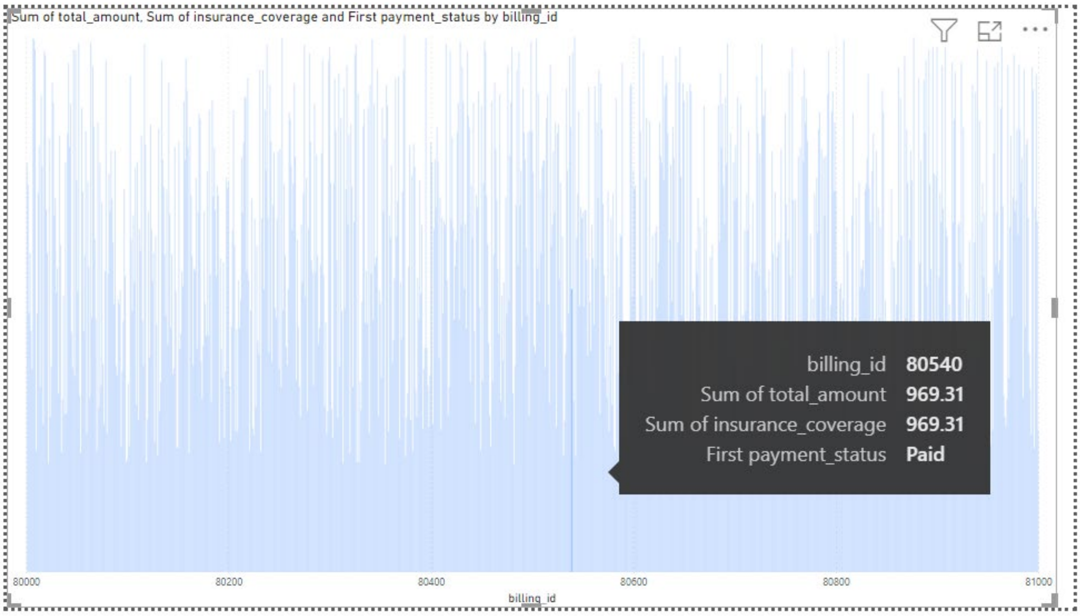
3) Doctor Distribution Across Specialties



Analysis:
The Power BI report shows the number of doctors grouped by Speciality in this clinic.

The graph shows that the most popular specialty for doctors is ophthalmology, followed by dermatology, endocrinology, and gastroenterology. The least popular specialty for doctors is urology.

4) Insurance Coverage vs. Total Bill Amount



Analysis:

The Power BI report shows the total amount of bills and the sum of insurance coverage for first payments. The report is grouped by billing ID.

The report shows that the total amount of bills for billing ID 80540 is $969.31, and the insurance coverage for the first payment is also $969.31. This means that the insurance company is covering the full cost of the first payment for this bill.

Additional insights:

- The billing ID with the highest total amount of bills is 80540.
- All the billing IDs with unpaid bills have a total amount of bills of zero.
- The billing ID with the highest insurance coverage for the first payment is 80540.