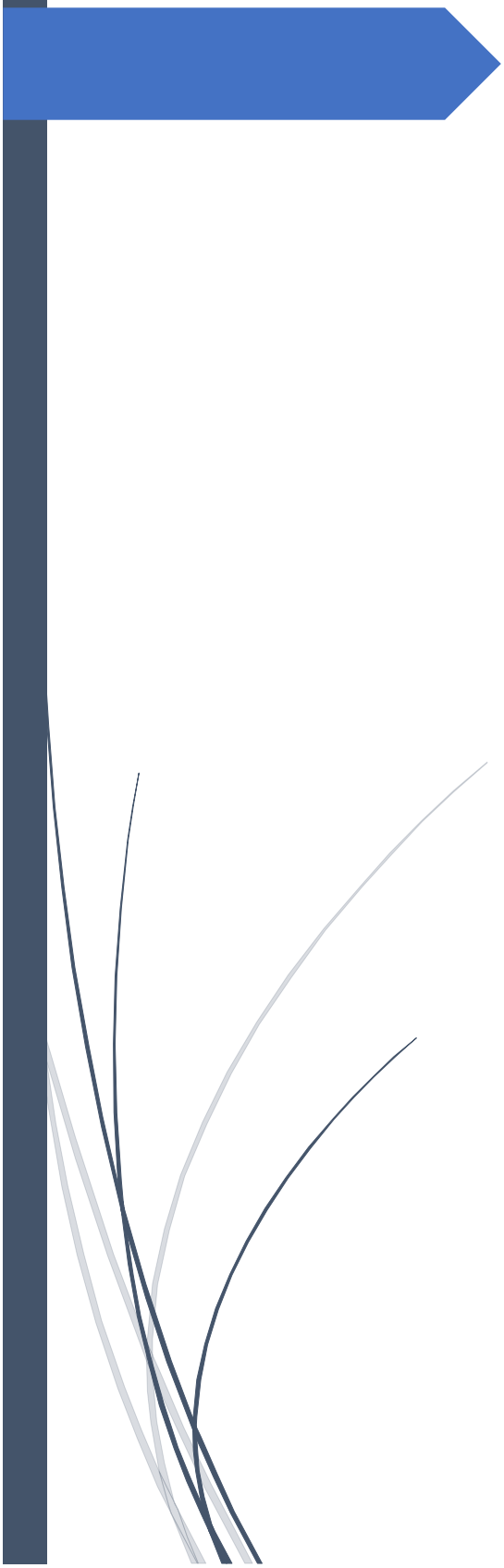
Business Management System



Group 7 – Purple

Safrin Patil
Aastha Bhargav
Karan Patel

Contents

Database Project Group Task

1	Logical DB design – 1.ERD 2.Data Structures 3.Architecture	Safrin Patil
2	Physical DB design – 1.Create Tables, 2.Constraints, 3.Sequences 4. triggers	Safrin Patil
3	Data Generation and loading data into the tables - Procedures, Functions, Excel/CSV imports. 1.Data Generation and loading data – Aastha. 2.Procedures and functions – Safrin 3.Excel/CSV imports – Aastha	Safrin and Aastha
4	Querying	Aastha
5	Data Visualization Power BI/Tableau	Aastha
6	Performance Tuning –Explain plans - Safrin. SQL Tuning - Safrin Indexing - Karan Partitioning of tables - Karan	Safrin and Karan
7	DBA Scripts – (Database Administrators in managing and monitoringdatabase operations) 1.System parameters 2. Privileges 3.Sessions 4. Locks - 5. Profiles 6. Clusters	All
8	Backup and Archival process - Create history tables.	All

Introduction

In today's fast-paced and competitive business landscape, effective business management is vital for organizations to thrive and succeed. A well-structured and efficiently managed business not only ensures operational excellence but also creates a positive impact on customer satisfaction and overall profitability. To streamline and enhance business management processes, our project focuses on the development of a comprehensive Business Management System (BMS).

This report serves as a comprehensive documentation of our journey in developing the Business Management System. It outlines the various aspects of our Business Management System, covering logical design, physical design, data generation and loading, query writing and database programming, performance tuning, database administration (DBA), user interface mockups, data visualization, database security, archival processes, and future scope. Our project aims to revolutionize the way businesses manage their operations, create better relationships with their customers, and provide effective education and training services. We believe that our BMS will significantly enhance efficiency and competitiveness in the business domain.

Key Components

Service Category and Services: The system allows businesses to categorize their services, making it easier to manage and provide a wide range of offerings efficiently. This categorization helps in organizing services, ensuring that customers can easily find the services they require.

Business Management: Business entities can effectively manage their information, including business details, contact information, and location data. Additionally, the system introduces an "is active" attribute to determine the operational status of a business.

Service Levels and Offerings: Services can be categorized into different levels, allowing businesses to offer different tiers of services based on customer needs and preferences. The introduction of pricing information enhances transparency in service offerings.

Employee Management: The system incorporates features for managing employees, including their roles and responsibilities within the organization. Employee information, including hiring date and contact details, is integrated for efficient staff management.

Batches: In the context of education and training, the system enables businesses to organize batches of students. This feature is crucial for institutions offering courses and workshops.

Customer Information: Managing customer data, including contact information and registration details, is a critical aspect of our BMS. The system captures customer data and allows for easy retrieval, enhancing customer relationship management.

Membership Types: Businesses can define different membership types with varying discounts, catering to customer loyalty programs and incentives.

Student Management: This component helps in the administration of students, including their personal information, batch enrollment, and membership types.

Project Requirements

Service Management:

Users should be able to add, edit, and delete service categories and services. Service categorization should allow businesses to organize their offerings effectively. Services can be assigned different service levels with associated prices.

Business Information Management:

The system should support the creation, modification, and deletion of business profiles.
Businesses should be able to specify their location, contact information, and operational status.

Employee Management:

The system should enable businesses to manage employee information, including their roles, contact details, and hiring dates. Employee roles can be defined as full-time or part-time.

Student Management:

Users should have the capability to register students, capture personal details, and associate them with specific batches. Membership types with varying discounts should be available for students to choose from.

Customer Management:

The system should support the management of customer information, including personal details, contact information, and registration dates.
Source tracking should be available to understand the origin of customers.

Data Query and Reporting:

Users should be able to perform queries and generate reports based on various criteria.
The system should provide data filtering, sorting, and exporting options for comprehensive reporting.

Data Visualization:

Users should have access to data visualization tools, such as charts and graphs, to help with data analysis.
Visualization features should enable users to gain insights into business performance and customer behavior.

Archival Process:

The system should include an automated archival process to maintain a manageable database size over time.
Historical data should be stored securely and made accessible for reference.

Assumptions

1. Users are assumed to receive basic training upon system implementation, but additional advanced training may be required for specific functionalities.

2. Users are expected to have access to a stable internet connection, understanding that occasional connectivity issues may occur.

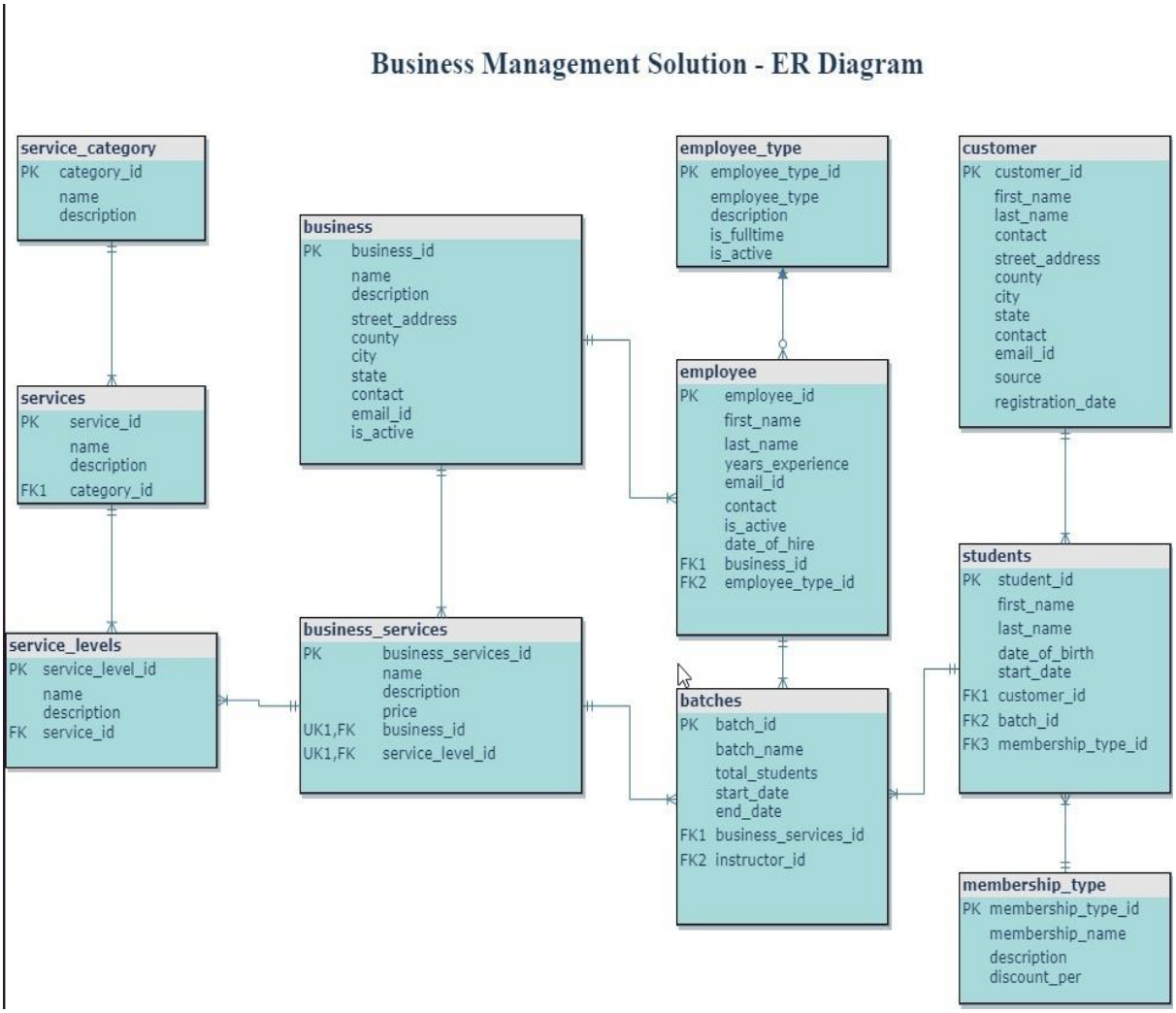
3. Users are presumed to have a basic understanding of security practices, although the system will enforce security measures to protect data.

4. The system is designed to handle moderate growth in terms of users and data volume, with potential expansion planning as the user base increases

LOGICAL DESIGN

ENTITY RELATIONSHIP DIAGRAM

ER Diagram shows relationships among various entity sets that are stored in the database.



Data Dictionary

TABLES

1. service_category:

Colum Name	Datatype	Nullable	Key Constraint
CATEGORY_ID	NUMBER (38,0)	No	Primary Key
NAME	VARCHAR2(50 BYTE)	No	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	

2. services:

Colum Name	Datatype	Nullable	Key Constraint
SERVICE_ID	NUMBER (38,0)	No	Primary Key
NAME	VARCHAR2(50 BYTE)	No	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	
CATEGORY_ID	NUMBER (38,0)	No	Foreign key

3. business:

Colum Name	Datatype	Nullable	Key Constraint
BUSINESS_ID	NUMBER (38,0)	No	Primary Key
NAME	VARCHAR2(50 BYTE)	No	
DESCRIPTION	VARCHAR2(2000 BYTE)	Yes	
STREET_ADDRESS	VARCHAR2(500 BYTE)	Yes	
COUNTY	VARCHAR2(50 BYTE)	Yes	
CITY	VARCHAR2(50 BYTE)	Yes	
STATE	VARCHAR2(50 BYTE)	Yes	
CONTACT	VARCHAR2(50 BYTE)	No	

EMAIL_ID	VARCHAR2(50 BYTE)	No	
IS_ACTIVE	CHAR(1 BYTE)	Yes	

4. service_levels:

Colum Name	Datatype	Nullable	Key Constraint
SERVICE_LEVEL_ID	NUMBER (38,0)	No	Primary Key
NAME	VARCHAR2(50 BYTE)	Yes	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	
SERVICE_ID	NUMBER (38,0)	Yes	Foreign Key

5. business_services:

Colum Name	Datatype	Nullable	Key Constraint
BUSINESS_SERVICES_ID	NUMBER (38,0)	No	Primary Key
NAME	VARCHAR2(100 BYTE)	Yes	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	
BUSINESS_ID	NUMBER (38,0)	Yes	
SERVICE_LEVEL_ID	NUMBER (38,0)	Yes	Foreign Key
PRICE	NUMBER (10,2)	Yes	

6. employee_type:

Colum Name	Datatype	Nullable	Key Constraint
EMPLOYEE_TYPE_ID	NUMBER(38,0)	No	Primary Key
EMPLOYEE_TYPE	VARCHAR2(50 BYTE)	Yes	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	
IS_FULLTIME	CHAR(1 BYTE)	Yes	
IS_ACTIVE	CHAR(1 BYTE)	Yes	

7. employee:

Colum Name	Datatype	Nullable	Key Constraint
EMPLOYEE_ID	NUMBER (38,0)	No	Primary Key
FIRST_NAME	VARCHAR2(50 BYTE)	Yes	
LAST_NAME	VARCHAR2(50 BYTE)	Yes	
EMPLOYEE_TYPE_ID	NUMBER (38,0)	Yes	Foreign Key
CONTACT	VARCHAR2(15 BYTE)	Yes	
YEARS_EXPERIENCE	NUMBER (38,0)	Yes	
BUSINESS_ID	NUMBER (38,0)	Yes	
IS_ACTIVE	CHAR (1 BYTE)	Yes	
DATE_OF_HIRE	DATE	Yes	
EMAIL_ID	VARCHAR2(50 BYTE)	No	

8. batches:

Colum Name	Datatype	Nullable	Key Constraint
BATCH_ID	NUMBER(38,0)	No	Primary Key
BATCH_NAME	VARCHAR2(200 BYTE)	Yes	
TOTAL_STUDENTS	NUMBER(38,0)	Yes	
START_DATE	DATE	Yes	
END_DATE	DATE	Yes	
BUSINESS_SERVICES_ID	NUMBER(38,0)	Yes	Foreign Key
INSTRUCTOR_ID	NUMBER(38,0)	Yes	Foreign Key

9. customer:

Colum Name	Datatype	Nullable	Key Constraint
CUSTOMER_ID	NUMBER(38,0)	No	Primary Key
FIRST_NAME	VARCHAR2(50 BYTE)	Yes	
LAST_NAME	VARCHAR2(50 BYTE)	Yes	
STREET_ADDRESS	VARCHAR2(500 BYTE)	Yes	

COUNTY	VARCHAR2(50 BYTE)	Yes	
CITY	VARCHAR2(50 BYTE)	Yes	
STATE	VARCHAR2(50 BYTE)	Yes	
CONTACT	VARCHAR2(50 BYTE)	No	
EMAIL_ID	VARCHAR2(50 BYTE)	No	
SOURCE	VARCHAR2(50 BYTE)	Yes	
REGISTRATION_DATE	DATE	Yes	

10. membership_type:

Colum Name	Datatype	Nullable	Key Constraint
MEMBERSHIP_TYPE_ID	NUMBER(38,0)	No	Primary Key
MEMBERSHIP_NAME	VARCHAR2(50 BYTE)	Yes	
DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	
DISCOUNT_PER	NUMBER(38,0)	Yes	

11. students

Colum Name	Datatype	Nullable	Key Constraint
STUDENT_ID	NUMBER(38,0)	No	Primary Key
FIRST_NAME	VARCHAR2(50 BYTE)	Yes	
LAST_NAME	VARCHAR2(50 BYTE)	Yes	
DATE_OF_BIRTH	DATE	Yes	
CUSTOMER_ID	NUMBER(38,0)	No	Foreign Key
BATCH_ID	NUMBER(38,0)	No	Foreign Key
MEMBERSHIP_TYPE_ID	NUMBER(38,0)	Yes	Foreign Key
START_DATE	DATE	Yes	

Sequences

Sequences in a database provide a reliable and efficient way to generate unique numeric values, typically for primary key columns. They are essential for ensuring data integrity and consistency, especially in multi-user environments.

Syntax for creating Sequence:

CREATE SEQUENCE SEQ_SERVICES_ID START WITH 1 INCREMENT BY 1;

The different sequences used in the project are as listed below:

1. SEQ_SERVICE_CATEGORY_ID
2. SEQ_SERVICES_ID
3. SEQ_BUSINESS_ID
4. SEQ_SERVICE_LEVELS_ID
5. SEQ_BUSINESS_SERVICES_ID
6. SEQ_EMPLOYEE_TYPE_ID
7. SEQ_EMPLOYEE_ID
8. SEQ_BATCHES_ID
9. SEQ_CUSTOMER_ID
10. SEQ_MEMBERSHIP_TYPE_ID
11. SEQ_STUDENTS_ID

Constraints

Several constraints have been implemented to ensure the integrity and quality of the data in the database. The constraints play a crucial role in maintaining data consistency and preventing incorrect or inconsistent data from being entered. Below, we'll describe the constraints that have been applied:

During the table creation process, we've also incorporated the Primary key, foreign key, default, check, not null, etc. constraints as below example of table creation 'employee'.

```
CREATE TABLE employee
(
   employee_id INTEGER PRIMARY KEY,
   first_name VARCHAR2(50),
   last_name VARCHAR2(50),
   employee_type_id INTEGER,
   contact VARCHAR2(15),
```

```
    years_experience INTEGER,
    business_id INTEGER,
    is_active CHAR(1) DEFAULT 'Y',  -- Added an is_active COLUMN
    date_of_hire DATE,
    email_id VARCHAR2(50) NOT NULL,
    CONSTRAINT chk_employee_email_id CHECK (REGEXP_LIKE(email_id, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-
]+\.[A-Za-z]{2,4}$')),
    CONSTRAINT fk_employee_business_id FOREIGN KEY (business_id) REFERENCES business(business_id),
    CONSTRAINT fk_employee_type_id FOREIGN KEY (employee_type_id) REFERENCES
employee_type(employee_type_id)
);
```

Let's break down the constraints used in above employee table creation:

PRIMARY KEY Constraint: The "employee_id" column is defined as the primary key for the "employee" table. A primary key uniquely identifies each row in the table, ensuring that no two rows can have the same "employee_id." This constraint enforces data uniqueness.

CHECK Constraint (chk_employee_email_id): A CHECK constraint named "chk_employee_email_id" is applied to the "email_id" column. This constraint uses a regular expression to validate that the "email_id" values conform to a specific email format. It ensures that only properly formatted email addresses are allowed in this column.

FOREIGN KEY Constraints: Two FOREIGN KEY constraints are defined to establish relationships with other tables. "fk_employee_business_id" is a foreign key constraint that references the "business" table's "business_id" column. It ensures that the values in the "business_id" column of the "employee" table match values in the "business_id" column of the "business" table. This enforces referential integrity between the two tables.

"fk_employee_type_id" is another foreign key constraint that references the "employee_type" table's "employee_type_id" column. It establishes a relationship between the "employee" table and the "employee_type" table, ensuring that only valid "employee_type_id" values can be entered in the "employee" table.

NOT NULL Constraint: The "email_id" column is specified as NOT NULL, indicating that it must have a value in every row. This constraint prevents the insertion of rows with NULL values in the "email_id" column.

DEFAULT Constraint: The "is_active" column is given a default value of 'Y.' This means that if a new row is inserted without specifying a value for "is_active," it will automatically be set to 'Y' (indicating the employee is active). This provides a default value in case one is not provided during data insertion.

Below table shows all the constraints created in the system.

	TABLE_NAME	COLUMN_NAME	POSITION	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	MEMBERSHIP_TYPE	MEMBERSHIP_TYPE_ID	1	SYS_C00151480	P
2	STUDENTS	CUSTOMER_ID	(null)	SYS_C00151472	C
3	STUDENTS	BATCH_ID	(null)	SYS_C00151473	C
4	STUDENTS	STUDENT_ID	1	SYS_C00151474	P
5	STUDENTS	CUSTOMER_ID	1	FK_STUDENTS_CUSTOMER_ID	R
6	STUDENTS	BATCH_ID	1	FK_STUDENTS_BATCH_ID	R
7	STUDENTS	MEMBERSHIP_TYPE_ID	1	FK_STUDENT_MEM_TYPE_ID	R
8	CUSTOMER	CONTACT	(null)	SYS_C00151467	C
9	CUSTOMER	EMAIL_ID	(null)	SYS_C00151468	C
10	CUSTOMER	EMAIL_ID	(null)	CHK_CUSTOMER_EMAIL_ID	C
11	CUSTOMER	CUSTOMER_ID	1	SYS_C00151470	P
12	BATCHES	BATCH_ID	1	SYS_C00151464	P
13	BATCHES	BUSINESS_SERVICES_ID	1	FK_BATCHES_BUS_SERVICES_ID	R
14	BATCHES	INSTRUCTOR_ID	1	FK_BATCHES_INSTRUCTOR_ID	R
15	EMPLOYEE	EMAIL_ID	(null)	SYS_C00151459	C
16	EMPLOYEE	EMAIL_ID	(null)	CHK_EMPLOYEE_EMAIL_ID	C
17	EMPLOYEE	EMPLOYEE_ID	1	SYS_C00151461	P
18	EMPLOYEE	BUSINESS_ID	1	FK_EMPLOYEE_BUSINESS_ID	R
19	EMPLOYEE	EMPLOYEE_TYPE_ID	1	FK_EMPLOYEE_TYPE_ID	R
20	EMPLOYEE_TYPE	EMPLOYEE_TYPE_ID	1	SYS_C00151458	P
21	SERVICE_LEVELS	SERVICE_LEVEL_ID	1	SYS_C00151453	P
22	SERVICE_LEVELS	SERVICE_ID	1	FK_SERVICE_ID	R
23	BUSINESS	NAME	(null)	SYS_C00151447	C
24	BUSINESS	CONTACT	(null)	SYS_C00151448	C
25	BUSINESS	EMAIL_ID	(null)	SYS_C00151449	C
26	BUSINESS	EMAIL_ID	(null)	CHK_BUSINESS_EMAIL_ID	C
27	BUSINESS	BUSINESS_ID	1	SYS_C00151451	P
28	BUSINESS	NAME	1	SYS_C00151452	U
29	SERVICES	NAME	(null)	SYS_C00151442	C
30	SERVICES	CATEGORY_ID	(null)	SYS_C00151443	C
31	SERVICES	SERVICE_ID	1	SYS_C00151444	P
32	SERVICES	NAME	1	SYS_C00151445	U
33	SERVICES	CATEGORY_ID	1	FK_CATEGORY_ID	R
34	SERVICE_CATEGORY	NAME	(null)	SYS_C00151439	C
35	SERVICE_CATEGORY	CATEGORY_ID	1	SYS_C00151440	P
36	SERVICE_CATEGORY	NAME	1	SYS_C00151441	U

PHYSICAL DESIGN

The Business Management System is a relational database consisting of Business operations and services. This database system is confined to a small-scale environment. Which supports the features of high concurrency and throughput. The BMS database will store historic and current data to facilitate ad-hoc querying, prepare significant reports for administrative purposes, and allow data mining related activities for future analysis and requirements.

This system will include basic database objects like tables, indices, integrity constraints and sub routines such as stored procedures. It includes 12 tables, a few of those Business, Business Services, Employee, Customer, Batches, Services etc. We have designed the database by normalizing the table values up to 3NF. To provide smaller response time and outputs, we have proposed the functionality of archiving the data once every 3 months. Because of the inclusion of indexes, we would expedite the query transaction processing. Database administrative activities would be monitored regularly by running DBA scripts for determining current, active, and inactive database sessions, storage statistics, queries that consume the most resources.

Data Generation and Loading

After creation of data structures insert scripts, procedures to generate data and import function through excel available in Oracle SQL Developer was used to generate relevant data. This method for generating data was fast and accurate with some time spent gathering the relevant information from the internet. We generated limited records for development purposes, count of each table are as below:

Table	Count
service_category	20
services	22
business	36
service_levels	65
business_services	49
employee_type	04
employee	140
batches	158
customer	11K
membership_type	10
students	12K

service_category:

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Education');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Sports');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Food');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Entertainment');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Healthcare');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Technology');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Travel');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Fashion');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Finance');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Art');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Automotive');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Home & Garden');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Pets');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Beauty');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Environment');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Science');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Music');

INSERT INTO service_category(category_id, name) **VALUES**(seq_service_category_id.NEXTVAL, 'Government');

Services:

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Swimming', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Fitness', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Mathematics', 1);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'English', 1);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Judo', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Cooking', 4);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Makeup', 3);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'DJ', 5);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Music', 5);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Dancing', 5);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Gymnastics', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Ribbon Dance', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Soccer', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Yoga', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Karate', 2);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Baking', 3);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Hair Styling', 14);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Photography', 10);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Singing', 17);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Painting', 10);

INSERT INTO services(service_id, name, category_id) VALUES(seq_services_id.NEXTVAL, 'Basketball', 2);

INSERT INTO services(service_id, name, category_id) **VALUES**(seq_services_id.NEXTVAL, 'Volleyball', 2);

Business:

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Watermelon Swim - Wesley Chapel', '30016 County Line Road', 'Hillsborough', 'Tampa', 'Florida', '8132297946', 'wesleychapel@watermelonswim.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Srishti Dance Academy - New Tampa', '10329 Cross Creek Blvd, Suite I', 'Hillsborough', 'Tampa', 'Florida', '2012535983', 'srishtidanceacademy@gmail.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Premier Gymnastics', '27033 State Rd 56, Wesley Chapel', 'Hillsborough', 'Tampa', 'Florida', '8139739920', 'info@premiergymnasticsfl.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Fitness First Gym', '123 Fitness Blvd', 'Hillsborough', 'Tampa', 'Florida', '8135556789', 'info@fitnessfirstgym.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Healthy Bites Cafe', '456 Nutritious St', 'Hillsborough', 'Tampa', 'Florida', '8139874321', 'info@healthybitescafe.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Sunset Yoga Studio', '789 Sunset Blvd', 'Hillsborough', 'Tampa', 'Florida', '8134567890', 'info@sunsetyogastudio.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Artistic Expressions Gallery', '101 Art Avenue', 'Hillsborough', 'Tampa', 'Florida', '8136543210', 'info@artisticexpressions.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Tech Innovators', '456 Tech Street', 'Hillsborough', 'Tampa', 'Florida', '8133334444', 'info@techinnovators.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Gourmet Delights Catering', '789 Gourmet Avenue', 'Hillsborough', 'Tampa', 'Florida', '8131112222', 'info@gourmetdelights.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Health and Wellness Center', '123 Wellness Way', 'Hillsborough', 'Tampa', 'Florida', '8138889999', 'info@healthandwellness.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Pet Paradise', '456 Pet Lane', 'Hillsborough', 'Tampa', 'Florida', '8137778888', 'info@petparadise.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'The Green Thumb Garden Center', '789 Garden Road', 'Hillsborough', 'Tampa', 'Florida', '8131234567', 'info@greenthumbgardens.com');

INSERT INTO business (business_id, name, street_address, county, city, st ate, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Financial Wizards', '101 Finance Street', 'Hillsborough', 'Tampa', 'Florida', '8139991111', 'info@financialwizards.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Beauty Haven Salon', '456 Beauty Blvd', 'Hillsborough', 'Tampa', 'Florida', '8132223333', 'info@beautyhaven.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Green Earth Recycling', '789 Recycle Lane', 'Hillsborough', 'Tampa', 'Florida', '8137771111', 'info@greenearthrecycling.com');

INSERT INTO business (business_id, name, street_address, county, city, state, contact, email_id) **VALUES** (seq_business_id.NEXTVAL, 'Science Explorers', '123 Science Street', 'Hillsborough', 'Tampa', 'Florida', '8134445555', 'info@scienceexplorers.com');

Customers

Insert Query

The given SQL query is an "INSERT" statement that is used to add a new row of data into the "CUSTOMER" table in the "PURPLE" schema of a database.

-- **Row 1**

INSERT INTO PURPLE.CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME, STREET_ADDRESS, COUNTY, CITY, STATE, CONTACT, EMAIL_ID, SOURCE, REGISTRATION_DATE)

VALUES (47, 'Beaufort', 'Uzielli', '1824 Dottie Road', 'Washington', 'Boca Raton', 'Florida', '5615451322', 'buzielli0@npr.org', 'Walk-In', TO_DATE('11/25/2022 23:57', 'MM/DD/YYYY HH24:MI'));

-- Row 2

INSERT INTO PURPLE.CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME, STREET_ADDRESS, COUNTY, CITY, STATE, CONTACT, EMAIL_ID, SOURCE, REGISTRATION_DATE)

VALUES (48, 'Laetitia', 'Rignall', '6142 Carey Drive', 'Taylor', 'Fort Lauderdale', 'Florida', '7542939297', 'lrignall1@wix.com', 'Advertisement', TO_DATE('9/5/2023 3:10', 'MM/DD/YYYY HH24:MI'));

-- Row 3

INSERT INTO PURPLE.CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME, STREET_ADDRESS, COUNTY, CITY, STATE, CONTACT, EMAIL_ID, SOURCE, REGISTRATION_DATE)

VALUES (49, 'Mary', 'Bilbrook', '213 Briar Crest Alley', 'Liberty', 'Melbourne', 'Florida', '3213449775', 'mbilbrook2@ning.com', 'Online', TO_DATE('9/6/2022 2:12', 'MM/DD/YYYY HH24:MI'));

-- Row 4

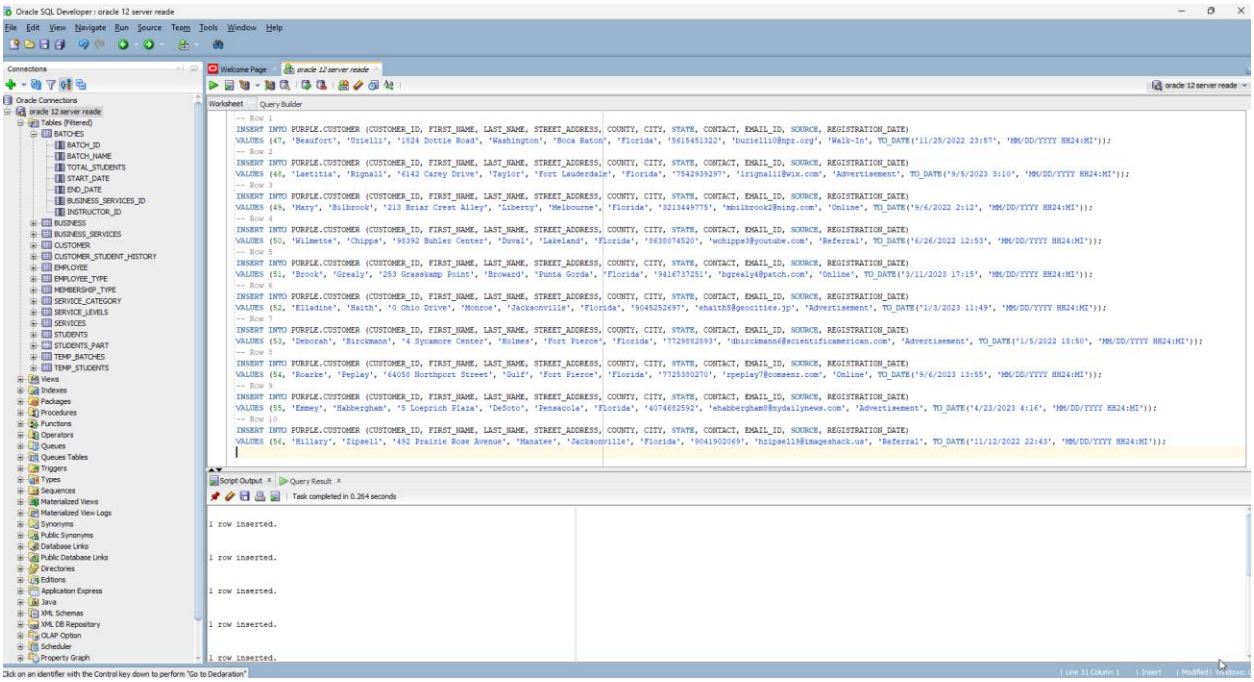
INSERT INTO PURPLE.CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME, STREET_ADDRESS, COUNTY, CITY, STATE, CONTACT, EMAIL_ID, SOURCE, REGISTRATION_DATE)

VALUES (50, 'Wilmette', 'Chipps', '98392 Buhler Center', 'Duval', 'Lakeland', 'Florida', '8638074520', 'wchipps3@youtube.com', 'Referral', TO_DATE('6/26/2022 12:53', 'MM/DD/YYYY HH24:MI'));

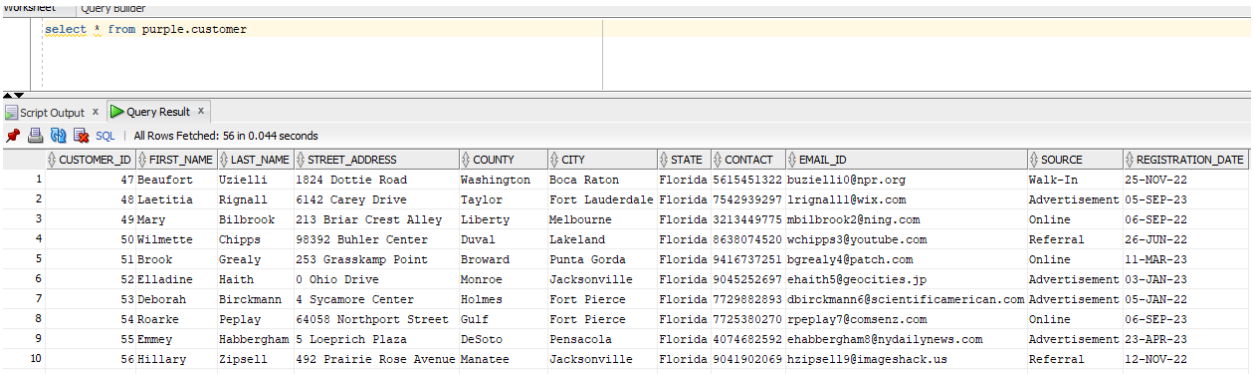
-- Row 5

INSERT INTO PURPLE.CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME, STREET_ADDRESS, COUNTY, CITY, STATE, CONTACT, EMAIL_ID, SOURCE, REGISTRATION_DATE)

VALUES (51, 'Brook', 'Grealy', '253 Grasskamp Point', 'Broward', 'Punta Gorda', 'Florida', '9416737251', 'bgrealy4@patch.com', 'Online', TO_DATE('3/11/2023 17:15', 'MM/DD/YYYY HH24:MI'));



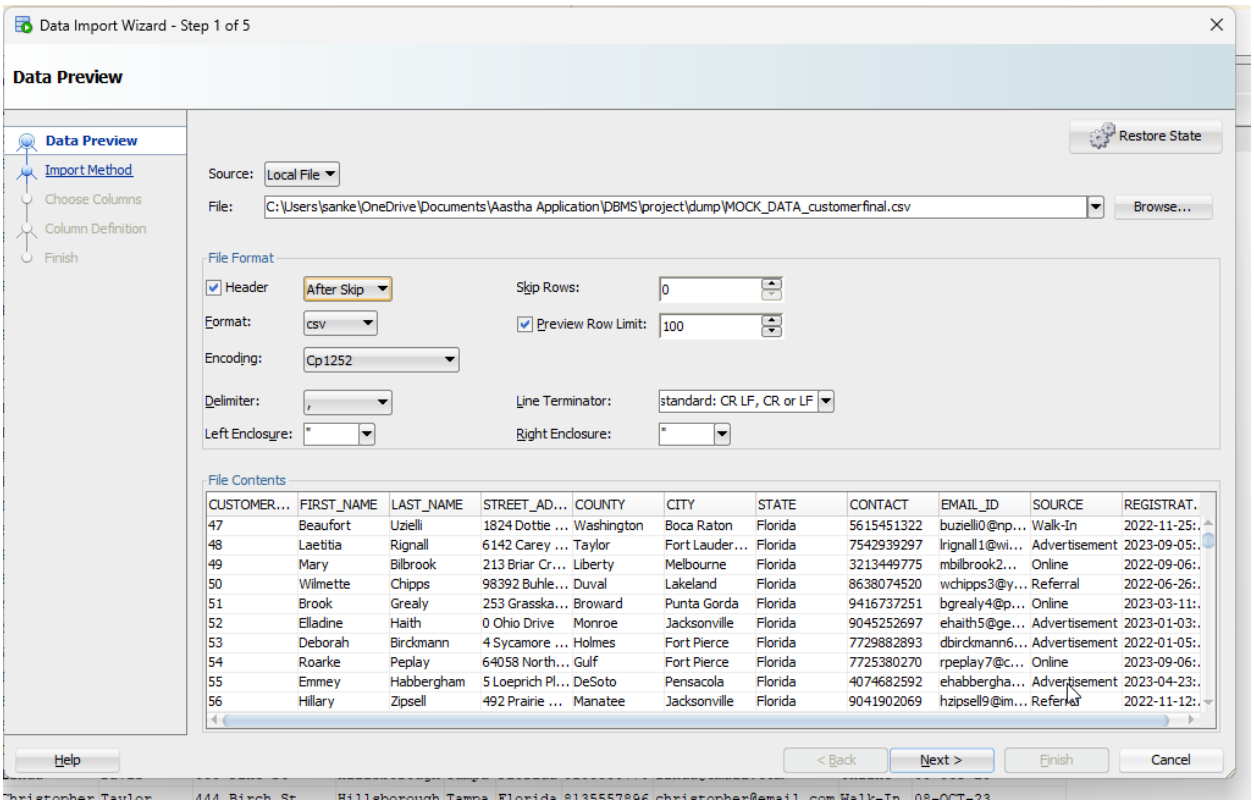
Data Load using CSV

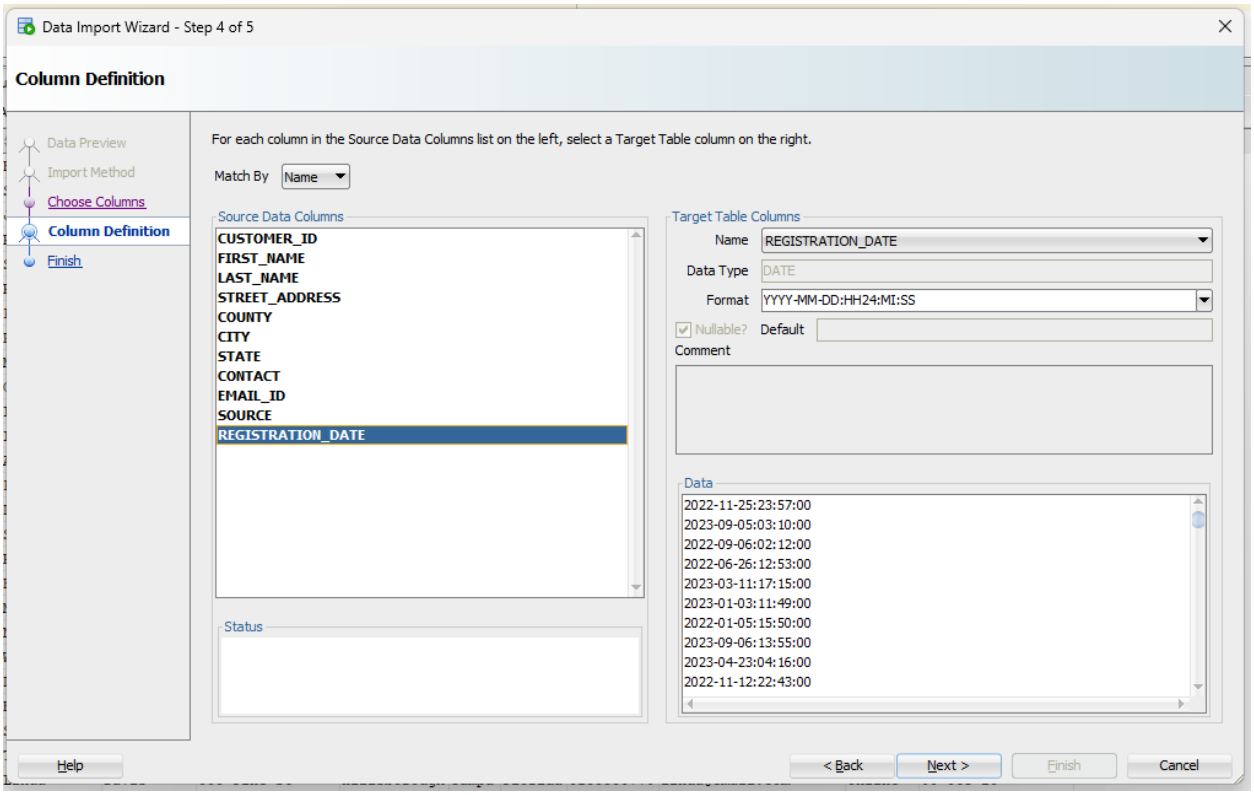


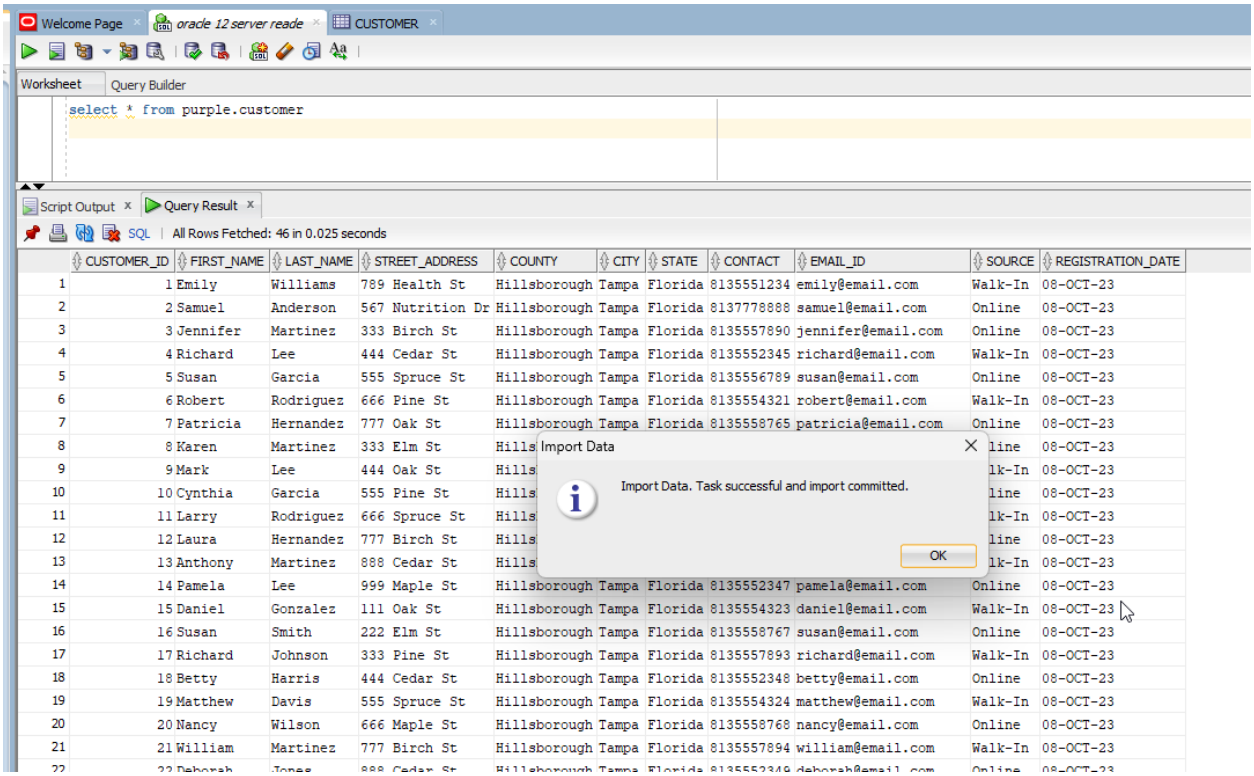
```
select * from purple.customer
```

Script Output | Query Result — All Rows Fetched: 56 in 0.044 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	STREET_ADDRESS	COUNTY	CITY	STATE	CONTACT	EMAIL_ID	SOURCE	REGISTRATION_DATE
1	47	Beaufort	Uzielli	1824 Dottie Road	Washington	Boca Raton	Florida	5615451322	buzielli0@npr.org	Walk-In	25-NOV-22
2	48	Laetitia	Rignall	6142 Carey Drive	Taylor	Fort Lauderdale	Florida	7542939297	lrignall1@wix.com	Advertisement	05-SEP-23
3	49	Mary	Bilbrook	213 Briar Crest Alley	Liberty	Melbourne	Florida	3213449775	mbilbrook2@ning.com	Online	06-SEP-22
4	50	Wilmette	Chipps	98392 Buhler Center	Duval	Lakeland	Florida	8638074520	wchipps3@youtube.com	Referral	26-JUN-22
5	51	Brook	Grealy	253 Grasskamp Point	Broward	Punta Gorda	Florida	9416737251	bgrealy4@patch.com	Online	11-MAR-23
6	52	Elladine	Haith	0 Ohio Drive	Monroe	Jacksonville	Florida	9045252697	ehaith5@geocities.jp	Advertisement	03-JAN-23
7	53	Deborah	Birckmann	4 Sycamore Center	Holmes	Fort Pierce	Florida	7729882893	dbirckmann6@scientificamerican.com	Advertisement	05-JAN-22
8	54	Roarke	Peplay	64058 Northport Street	Gulf	Fort Pierce	Florida	7725380270	rpeplay7@comsenz.com	Online	06-SEP-23
9	55	Emmey	Habbergham	5 Loeprich Plaza	DeSoto	Pensacola	Florida	4074682592	ehabbergham8@nydailynews.com	Advertisement	23-APR-23
10	56	Hillary	Zipsell	492 Prairie Rose Avenue	Manatee	Jacksonville	Florida	9041902069	hzipsell9@imageshack.us	Referral	12-NOV-22


Data Import Wizard - Step 1 of 5
Data Preview
Data Preview
Import Method
Choose Columns
Column Definition
Finish
Restore State
Source: Local File
File: C:\Users\sanke\OneDrive\Documents\Aastha Application\DBMS\project\dump\MOCK_DATA_customerfinal.csv
Browse...
File Format
Header
After Skip
Skip Rows: 0
Format: csv
Preview Row Limit: 100
Encoding: Cp1252
Delimiter: ,
Line Terminator: standard: CR LF, CR or LF
Left Enclosure: "
Right Enclosure: "
File Contents
CUSTOMER... FIRST_NAME LAST_NAME STREET_AD... COUNTY CITY STATE CONTACT EMAIL_ID SOURCE REGISTRAT.
47 Beaufort Uzielli 1824 Dottie ... Washington Boca Raton Florida 5615451322 buzielli0@np... Walk-In 2022-11-25:..
48 Laetitia Rignall 6142 Carey ... Taylor Fort Lauder... Florida 7542939297 lrignall1@wi... Advertisement 2023-09-05:..
49 Mary Bilbrook 213 Briar Cr... Liberty Melbourne Florida 3213449775 mbilbrook2... Online 2022-09-06:..
50 Wilmette Chipps 98392 Buhle... Duval Lakeland Florida 8638074520 wchipps3@y... Referral 2022-06-26:..
51 Brook Grealy 253 Grasska... Broward Punta Gorda Florida 9416737251 bgrealy4@p... Online 2023-03-11:..
52 Elladine Haith 0 Ohio Drive Monroe Jacksonville Florida 9045252697 ehaith5@ge... Advertisement 2023-01-03:..
53 Deborah Birckmann 4 Sycamore ... Holmes Fort Pierce Florida 7729882893 dbirckmann6... Advertisement 2022-01-05:..
54 Roarke Peplay 64058 North... Gulf Fort Pierce Florida 7725380270 rpeplay7@c... Online 2023-09-06:..
55 Emmey Habbergham 5 Loeprich Pl... DeSoto Pensacola Florida 4074682592 ehabbergha... Advertisement 2023-04-23:..
56 Hillary Zipsell 492 Prairie ... Manatee Jacksonville Florida 9041902069 hzipsell9@im... Referral 2022-11-12:..
Help
< Back
Next >
Finish
Cancel


Data Import Wizard - Step 4 of 5
Column Definition
Data Preview
Import Method
Choose Columns
Column Definition
Finish
For each column in the Source Data Columns list on the left, select a Target Table column on the right.
Match By Name
Source Data Columns
CUSTOMER_ID
FIRST_NAME
LAST_NAME
STREET_ADDRESS
COUNTY
CITY
STATE
CONTACT
EMAIL_ID
SOURCE
REGISTRATION_DATE
Target Table Columns
Name REGISTRATION_DATE
Data Type DATE
Format YYYY-MM-DD:HH24:MI:SS
Nullable? Default
Comment
Data
2022-11-25:23:57:00
2023-09-05:03:10:00
2022-09-06:02:12:00
2022-06-26:12:53:00
2023-03-11:17:15:00
2023-01-03:11:49:00
2022-01-05:15:50:00
2023-09-06:13:55:00
2023-04-23:04:16:00
2022-11-12:22:43:00
Status
Help
< Back
Next >
Finish
Cancel



	CUSTOMER_ID	FIRST_NAME	LAST_NAME	STREET_ADDRESS	COUNTY	CITY	STATE	CONTACT	EMAIL_ID	SOURCE	REGISTRATION_DATE
1	1	Emily	Williams	789 Health St	Hillsborough	Tampa	Florida	8135551234	emily@email.com	Walk-In	08-OCT-23
2	2	Samuel	Anderson	567 Nutrition Dr	Hillsborough	Tampa	Florida	8137778888	samuel@email.com	Online	08-OCT-23
3	3	Jennifer	Martinez	333 Birch St	Hillsborough	Tampa	Florida	8135557890	jennifer@email.com	Online	08-OCT-23
4	4	Richard	Lee	444 Cedar St	Hillsborough	Tampa	Florida	8135552345	richard@email.com	Walk-In	08-OCT-23
5	5	Susan	Garcia	555 Spruce St	Hillsborough	Tampa	Florida	8135556789	susan@email.com	Online	08-OCT-23
6	6	Robert	Rodriguez	666 Pine St	Hillsborough	Tampa	Florida	8135554321	robert@email.com	Walk-In	08-OCT-23
7	7	Patricia	Hernandez	777 Oak St	Hillsborough	Tampa	Florida	8135558765	patricia@email.com	Online	08-OCT-23
8	8	Karen	Martinez	333 Elm St	Hills					line	08-OCT-23
9	9	Mark	Lee	444 Oak St	Hills					lk-In	08-OCT-23
10	10	Cynthia	Garcia	555 Pine St	Hills					line	08-OCT-23
11	11	Larry	Rodriguez	666 Spruce St	Hills					lk-In	08-OCT-23
12	12	Laura	Hernandez	777 Birch St	Hills					line	08-OCT-23
13	13	Anthony	Martinez	888 Cedar St	Hills					lk-In	08-OCT-23
14	14	Pamela	Lee	999 Maple St	Hillsborough	Tampa	Florida	8135552347	pamela@email.com	Online	08-OCT-23
15	15	Daniel	Gonzalez	111 Oak St	Hillsborough	Tampa	Florida	8135554323	daniel@email.com	Walk-In	08-OCT-23
16	16	Susan	Smith	222 Elm St	Hillsborough	Tampa	Florida	8135558767	susan@email.com	Online	08-OCT-23
17	17	Richard	Johnson	333 Pine St	Hillsborough	Tampa	Florida	8135557893	richard@email.com	Walk-In	08-OCT-23
18	18	Betty	Harris	444 Cedar St	Hillsborough	Tampa	Florida	8135552348	betty@email.com	Online	08-OCT-23
19	19	Matthew	Davis	555 Spruce St	Hillsborough	Tampa	Florida	8135554324	matthew@email.com	Walk-In	08-OCT-23
20	20	Nancy	Wilson	666 Maple St	Hillsborough	Tampa	Florida	8135558768	nancy@email.com	Online	08-OCT-23
21	21	William	Martinez	777 Birch St	Hillsborough	Tampa	Florida	8135557894	william@email.com	Walk-In	08-OCT-23

STUDENTS

Insert Query

INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)

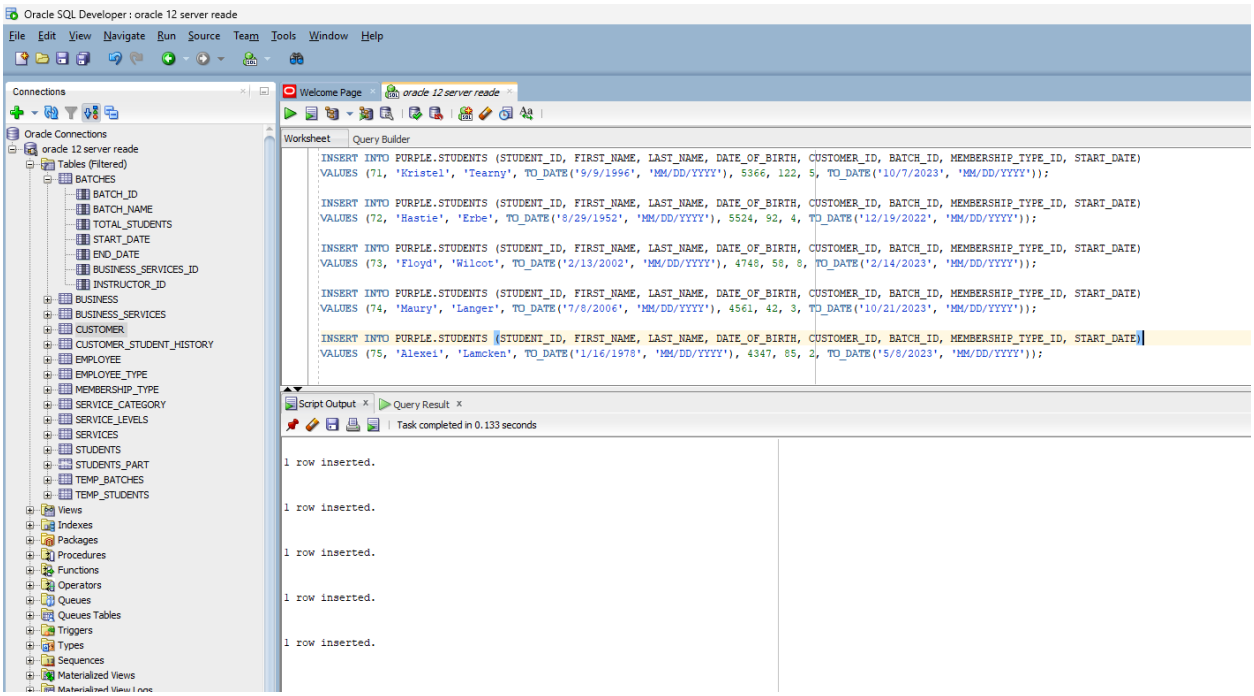
VALUES (71, 'Kristel', 'Tearny', TO_DATE('9/9/1996', 'MM/DD/YYYY'), 5366, 122, 5, TO_DATE('10/7/2023', 'MM/DD/YYYY'));

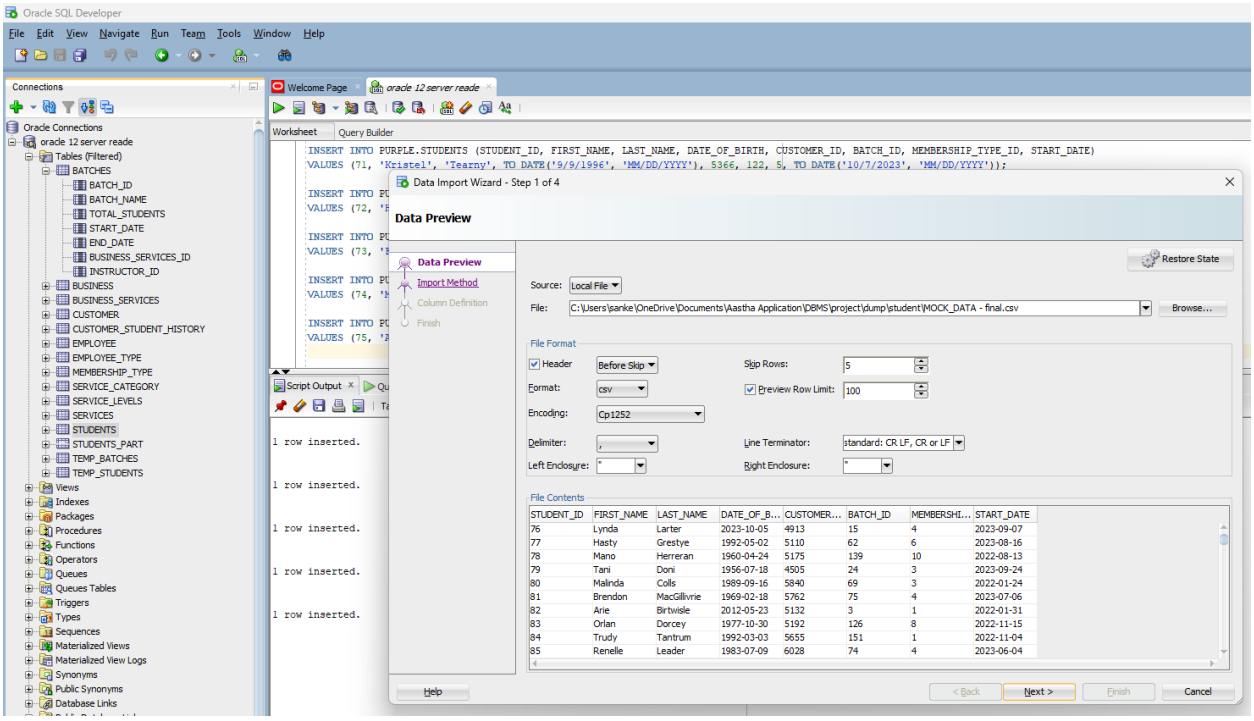
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)

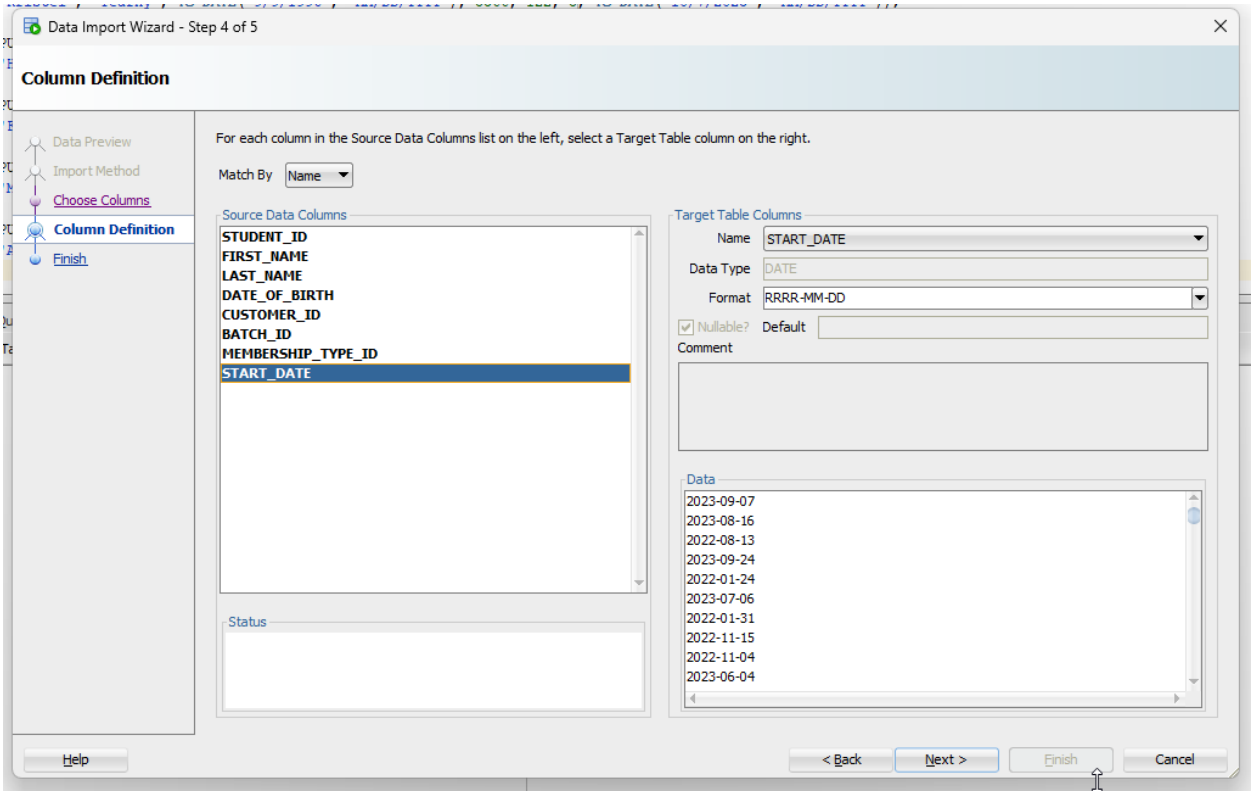
VALUES (72, 'Hastie', 'Erbe', TO_DATE('8/29/1952', 'MM/DD/YYYY'), 5524, 92, 4, TO_DATE('12/19/2022', 'MM/DD/YYYY'));

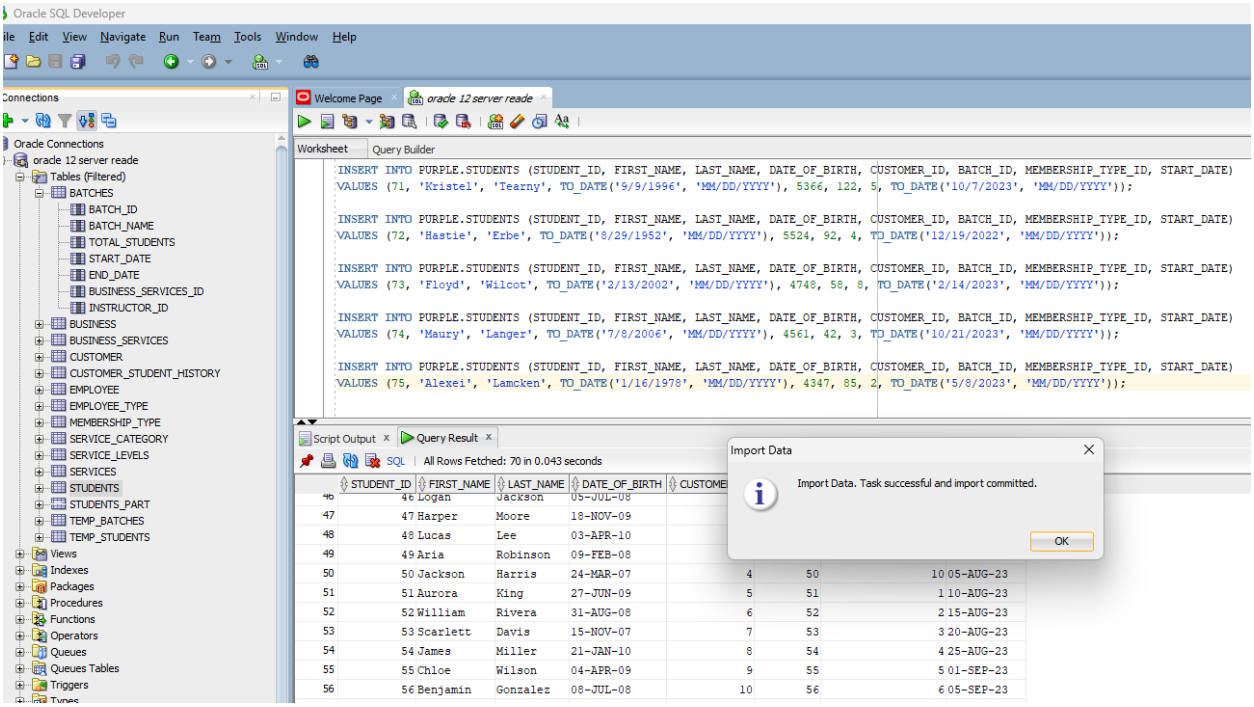
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)

VALUES (73, 'Floyd', 'Wilcot', TO_DATE('2/13/2002', 'MM/DD/YYYY'), 4748, 58, 8, TO_DATE('2/14/2023', 'MM/DD/YYYY'));


Oracle SQL Developer : oracle 12 server reade
File Edit View Navigate Run Source Team Tools Window Help
Connections
Oracle Connections
oracle 12 server reade
Tables (Filtered)
BATCHES
BATCH_ID
BATCH_NAME
TOTAL_STUDENTS
START_DATE
END_DATE
BUSINESS_SERVICES_ID
INSTRUCTOR_ID
BUSINESS
BUSINESS_SERVICES
CUSTOMER
CUSTOMER_STUDENT_HISTORY
EMPLOYEE
EMPLOYEE_TYPE
MEMBERSHIP_TYPE
SERVICE_CATEGORY
SERVICE_LEVELS
SERVICES
STUDENTS
STUDENTS_PART
TEMP_BATCHES
TEMP_STUDENTS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Welcome Page
oracle 12 server reade
Worksheet
Query Builder
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (71, 'Kristel', 'Tearny', TO_DATE('9/9/1996', 'MM/DD/YYYY'), 5366, 122, 5, TO_DATE('10/7/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (72, 'Hastie', 'Erbe', TO_DATE('8/29/1952', 'MM/DD/YYYY'), 5524, 92, 4, TO_DATE('12/19/2022', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (73, 'Floyd', 'Wilcot', TO_DATE('2/13/2002', 'MM/DD/YYYY'), 4748, 58, 8, TO_DATE('2/14/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (74, 'Maury', 'Langer', TO_DATE('7/8/2006', 'MM/DD/YYYY'), 4561, 42, 3, TO_DATE('10/21/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (75, 'Alexei', 'Lamcken', TO_DATE('1/16/1978', 'MM/DD/YYYY'), 4347, 85, 2, TO_DATE('5/8/2023', 'MM/DD/YYYY'));
Script Output
Query Result
Task completed in 0.133 seconds
1 row inserted.
1 row inserted.
1 row inserted.
1 row inserted.
1 row inserted.


Oracle SQL Developer
File Edit View Navigate Run Team Tools Window Help
Data Import Wizard - Step 1 of 4
Data Preview
Data Preview
Import Method
Column Definition
Finish
Restore State
Source: Local File
File: C:\Users\sanke\OneDrive\Documents\Aastha Application\DBMS\project\dump\student\MOCK_DATA - final.csv
Browse...
File Format
Header
Before Skip
Skip Rows: 5
Format: csv
Preview Row Limit: 100
Encoding: Cp1252
Delimiter: ,
Line Terminator: standard: CR LF, CR or LF
Left Enclosure: "
Right Enclosure: "
File Contents
STUDENT_ID FIRST_NAME LAST_NAME DATE_OF_B... CUSTOMER... BATCH_ID MEMBERSHI... START_DATE
76 Lynda Larter 2023-10-05 4913 15 4 2023-09-07
77 Hasty Grestye 1992-05-02 5110 62 6 2023-08-16
78 Mano Herreran 1960-04-24 5175 139 10 2022-08-13
79 Tani Doni 1956-07-18 4505 24 3 2023-09-24
80 Malinda Colls 1989-09-16 5840 69 3 2022-01-24
81 Brendon MacGillivrie 1969-02-18 5762 75 4 2023-07-06
82 Arie Birtwisle 2012-05-23 5132 3 1 2022-01-31
83 Orlan Dorcey 1977-10-30 5192 126 8 2022-11-15
84 Trudy Tantrum 1992-03-03 5655 151 1 2022-11-04
85 Renelle Leader 1983-07-09 6028 74 4 2023-06-04
Help
< Back
Next >
Finish
Cancel


Data Import Wizard - Step 4 of 5
Column Definition
Data Preview
Import Method
Choose Columns
Column Definition
Finish
For each column in the Source Data Columns list on the left, select a Target Table column on the right.
Match By
Name
Source Data Columns
STUDENT_ID
FIRST_NAME
LAST_NAME
DATE_OF_BIRTH
CUSTOMER_ID
BATCH_ID
MEMBERSHIP_TYPE_ID
START_DATE
Target Table Columns
Name
START_DATE
Data Type
DATE
Format
RRRR-MM-DD
Nullable?
Default
Comment
Data
2023-09-07
2023-08-16
2022-08-13
2023-09-24
2022-01-24
2023-07-06
2022-01-31
2022-11-15
2022-11-04
2023-06-04
Status
Help
< Back
Next >
Finish
Cancel


Oracle SQL Developer
File Edit View Navigate Run Team Tools Window Help
Connections
Oracle Connections
oracle 12 server reade
Tables (Filtered)
BATCHES
BATCH_ID
BATCH_NAME
TOTAL_STUDENTS
START_DATE
END_DATE
BUSINESS_SERVICES_ID
INSTRUCTOR_ID
BUSINESS
BUSINESS_SERVICES
CUSTOMER
CUSTOMER_STUDENT_HISTORY
EMPLOYEE
EMPLOYEE_TYPE
MEMBERSHIP_TYPE
SERVICE_CATEGORY
SERVICE_LEVELS
SERVICES
STUDENTS
STUDENTS_PART
TEMP_BATCHES
TEMP_STUDENTS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Welcome Page
oracle 12 server reade
Worksheet
Query Builder
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (71, 'Kristel', 'Tearny', TO_DATE('9/9/1996', 'MM/DD/YYYY'), 5366, 122, 5, TO_DATE('10/7/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (72, 'Hastie', 'Erbe', TO_DATE('8/29/1952', 'MM/DD/YYYY'), 5524, 92, 4, TO_DATE('12/19/2022', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (73, 'Floyd', 'Wilcot', TO_DATE('2/13/2002', 'MM/DD/YYYY'), 4748, 58, 8, TO_DATE('2/14/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (74, 'Maury', 'Langer', TO_DATE('7/8/2006', 'MM/DD/YYYY'), 4561, 42, 3, TO_DATE('10/21/2023', 'MM/DD/YYYY'));
INSERT INTO PURPLE.STUDENTS (STUDENT_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, CUSTOMER_ID, BATCH_ID, MEMBERSHIP_TYPE_ID, START_DATE)
VALUES (75, 'Alexei', 'Lamcken', TO_DATE('1/16/1978', 'MM/DD/YYYY'), 4347, 85, 2, TO_DATE('5/8/2023', 'MM/DD/YYYY'));
Script Output
Query Result
SQL
All Rows Fetched: 70 in 0.043 seconds
STUDENT_ID FIRST_NAME LAST_NAME DATE_OF_BIRTH
46 Logan Jackson 05-JUL-08
47 47 Harper Moore 18-NOV-09
48 48 Lucas Lee 03-APR-10
49 49 Aria Robinson 09-FEB-08
50 50 Jackson Harris 24-MAR-07 4 50 10 05-AUG-23
51 51 Aurora King 27-JUN-09 5 51 1 10-AUG-23
52 52 William Rivera 31-AUG-08 6 52 2 15-AUG-23
53 53 Scarlett Davis 15-NOV-07 7 53 3 20-AUG-23
54 54 James Miller 21-JAN-10 8 54 4 25-AUG-23
55 55 Chloe Wilson 04-APR-09 9 55 5 01-SEP-23
56 56 Benjamin Gonzalez 08-JUL-08 10 56 6 05-SEP-23
Import Data
Import Data. Task successful and import committed.
OK

QUERY WRITING & DATABASE PROGRAMMING

Queries

Query 1- This query will provide a structured view of businesses, their services, service levels, business service names, and associated rates, with the data organized by the specified order.

```
SELECT b.BUSINESS_ID , b.name,
                s.service_id, s.name AS SERVICE_NAME,
                sl.service_level_id, sl.name AS SERVICE_LEVEL_NAME,
                bs.name AS BUSINESS_SERVICE_NAME, bs.price AS RATE
FROM services s
        JOIN service_levels sl ON s.service_id = sl.service_id
        JOIN business_services bs ON bs.service_level_id = sl.service_level_id
        JOIN business b ON bs.business_id = b.business_id
ORDER BY b.business_id, s.service_id, sl.service_level_id;
```

Output:

	BUSINESS_ID	NAME	SERVICE_ID	SERVICE_NAME	SERVICE_LEVEL_ID	SERVICE_LEVEL_NAME	BUSINESS_SERVICE_NAME	RATE
1	1	Watermelon Swim - Wesley Chapel	1	Swimming	1	Beginner	Adaptive Swimming	200
2	1	Watermelon Swim - Wesley Chapel	1	Swimming	1	Beginner	Adult Swimming	200
3	1	Watermelon Swim - Wesley Chapel	1	Swimming	1	Beginner	Kids Swimming	180
4	1	Watermelon Swim - Wesley Chapel	1	Swimming	2	Intermediate	Kids Swimming	200
5	1	Watermelon Swim - Wesley Chapel	1	Swimming	2	Intermediate	Adaptive Swimming	220
6	1	Watermelon Swim - Wesley Chapel	1	Swimming	2	Intermediate	Adult Swimming	220
7	1	Watermelon Swim - Wesley Chapel	1	Swimming	3	Advanced	Adult Swimming	240
8	1	Watermelon Swim - Wesley Chapel	1	Swimming	3	Advanced	Adaptive Swimming	260
9	1	Watermelon Swim - Wesley Chapel	1	Swimming	3	Advanced	Kids Swimming	240
10	2	Srishti Dance Academy - New Tampa	10	Dancing	28	Dance Basics	Kids Dancing	100
11	2	Srishti Dance Academy - New Tampa	10	Dancing	28	Dance Basics	Adult Dancing	120
12	2	Srishti Dance Academy - New Tampa	10	Dancing	28	Dance Basics	Teen Dancing	120
13	2	Srishti Dance Academy - New Tampa	10	Dancing	29	Intermediate Dance Styles	Teen Dancing	130
14	2	Srishti Dance Academy - New Tampa	10	Dancing	29	Intermediate Dance Styles	Adult Dancing	140
15	2	Srishti Dance Academy - New Tampa	10	Dancing	29	Intermediate Dance Styles	Kids Dancing	110
16	2	Srishti Dance Academy - New Tampa	10	Dancing	30	Advanced Dance Performance	Adult Dancing	160
17	2	Srishti Dance Academy - New Tampa	10	Dancing	30	Advanced Dance Performance	Kids Dancing	120
18	2	Srishti Dance Academy - New Tampa	10	Dancing	30	Advanced Dance Performance	Teen Dancing	150
19	3	Premier Gymnastics	11	Gymnastics	31	Gymnastics Basics	Gymnastics - Bronze	250
20	3	Premier Gymnastics	11	Gymnastics	32	Intermediate Gymnastics	Gymnastics - Silver	270
21	3	Premier Gymnastics	11	Gymnastics	33	Advanced Gymnastics	Gymnastics - Gold	290
22	3	Premier Gymnastics	12	Ribbon Dance	34	Ribbon Dance Fundamentals	Ribbon Dance - Bronze	260
23	3	Premier Gymnastics	12	Ribbon Dance	35	Intermediate Ribbon Dance	Ribbon Dance - Silver	280

Query 2 - This query is valuable for identifying businesses providing fitness services at the specified price point, facilitating comparison among different service providers.

```
SELECT b.BUSINESS_ID , b.name,
```

```
        s.service_id, s.name AS SERVICE_NAME,
        sl.service_level_id, sl.name AS SERVICE_LEVEL_NAME,
        bs.name AS BUSINESS_SERVICE_NAME, bs.price AS RATE
FROM services s
    JOIN service_levels sl ON s.service_id = sl.service_id
    JOIN business_services bs ON bs.service_level_id = sl.service_level_id
    JOIN business b ON bs.business_id = b.business_id
WHERE s.name = 'Fitness' AND bs.price <= 100
ORDER BY b.business_id, s.service_id, sl.service_level_id;
```

Output:

	BUSINESS_ID	NAME	SERVICE_ID	SERVICE_NAME	SERVICE_LEVEL_ID	SERVICE_LEVEL_NAME	BUSINESS_SERVICE_NAME	RATE
1	4	Fitness First Gym	2	Fitness	4	Beginner Fitness	Basic Fitness Training	80
2	4	Fitness First Gym	2	Fitness	5	Intermediate Fitness	Intermediate Fitness Training	100

Query 3 - This query provides an overview of the number of students in each service, making it easy to identify which services have the most students enrolled.

```
WITH student AS (
    SELECT  s.*
    FROM PURPLE.STUDENTS_PART s
),

birthyearwise_student_services AS (
    SELECT
        s.name AS service_name,
        count (*) AS total_students
    FROM services s
        JOIN service_levels sl ON sl.service_id = s.service_id
        JOIN business_services bs ON bs.service_level_id = sl.service_level_id
        JOIN batches b ON b.business_services_id = bs.business_services_id
        JOIN student st ON st.batch_id = b.batch_id
    GROUP BY s.name
    ORDER BY service_name,  total_students DESC
)
SELECT *
FROM birthyearwise_student_services;
```

Output:

	SERVICE_NAME	TOTAL_STUDENTS
1	Dancing	15
2	Fitness	6
3	Gymnastics	9
4	Hair Styling	6
5	Photography	4
6	Singing	2
7	Swimming	22
8	Yoga	6

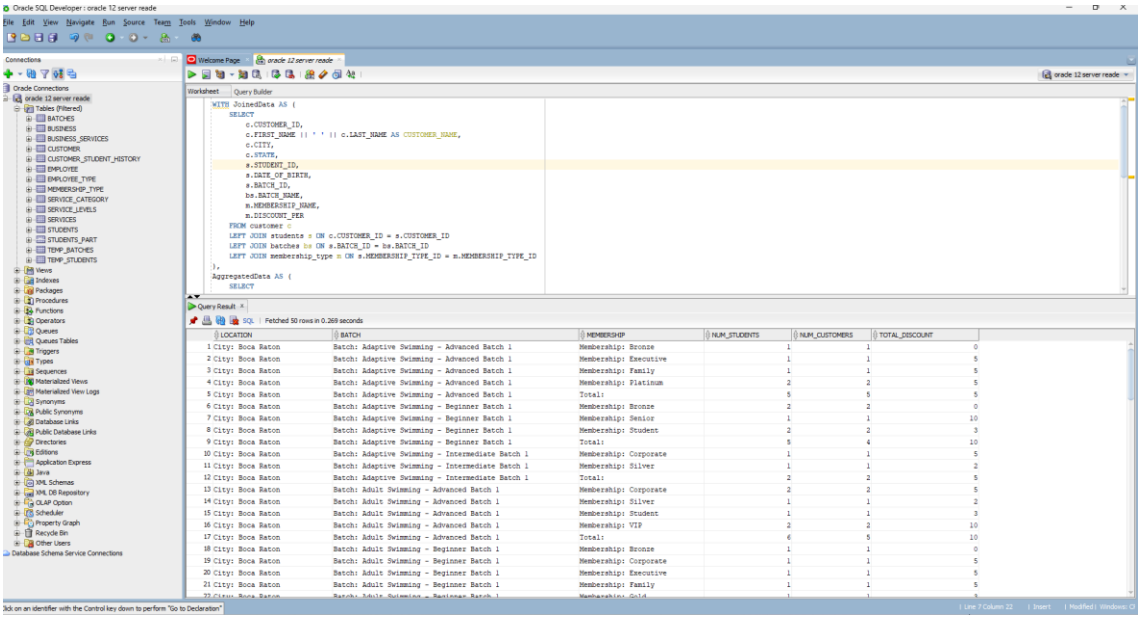
Query 4 –

```
WITH JoinedData AS (
   SELECT
      c.CUSTOMER_ID,
      c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME,
      c.CITY,
      c.STATE,
      s.STUDENT_ID,
      s.DATE_OF_BIRTH,
      s.BATCH_ID,
      bs.BATCH_NAME,
      m.MEMBERSHIP_NAME,
      m.DISCOUNT_PER
   FROM customer c
   LEFT JOIN students s ON c.CUSTOMER_ID = s.CUSTOMER_ID
   LEFT JOIN batches bs ON s.BATCH_ID = bs.BATCH_ID
   LEFT JOIN membership_type m ON s.MEMBERSHIP_TYPE_ID = m.MEMBERSHIP_TYPE_ID
),
AggregatedData AS (
   SELECT
      CITY,
      STATE,
      BATCH_NAME,
      MEMBERSHIP_NAME,
      COUNT(STUDENT_ID) AS NUM_STUDENTS,
      COUNT(DISTINCT CUSTOMER_ID) AS NUM_CUSTOMERS,
      MAX(DISCOUNT_PER) AS TOTAL_DISCOUNT
   FROM JoinedData
```

```
    GROUP BY ROLLUP (CITY, STATE, BATCH_NAME, MEMBERSHIP_NAME)
)
SELECT
    CASE
        WHEN CITY IS NOT NULL THEN 'City: ' || CITY
        ELSE 'Total:'
    END AS LOCATION,
    CASE
        WHEN BATCH_NAME IS NOT NULL THEN 'Batch: ' || BATCH_NAME
        ELSE 'Total:'
    END AS BATCH,
    CASE
        WHEN MEMBERSHIP_NAME IS NOT NULL THEN 'Membership: ' || MEMBERSHIP_NAME
        ELSE 'Total:'
    END AS MEMBERSHIP,
    NUM_STUDENTS,
    NUM_CUSTOMERS,
    TOTAL_DISCOUNT
FROM AggregatedData
ORDER BY CITY NULLS LAST, BATCH_NAME NULLS LAST, MEMBERSHIP_NAME NULLS LAST;
```

Description- This query provides insights and business metrics related to the number of students, customers, and total discounts, grouped by different levels of aggregation, such as city, batch, and membership.

Output:

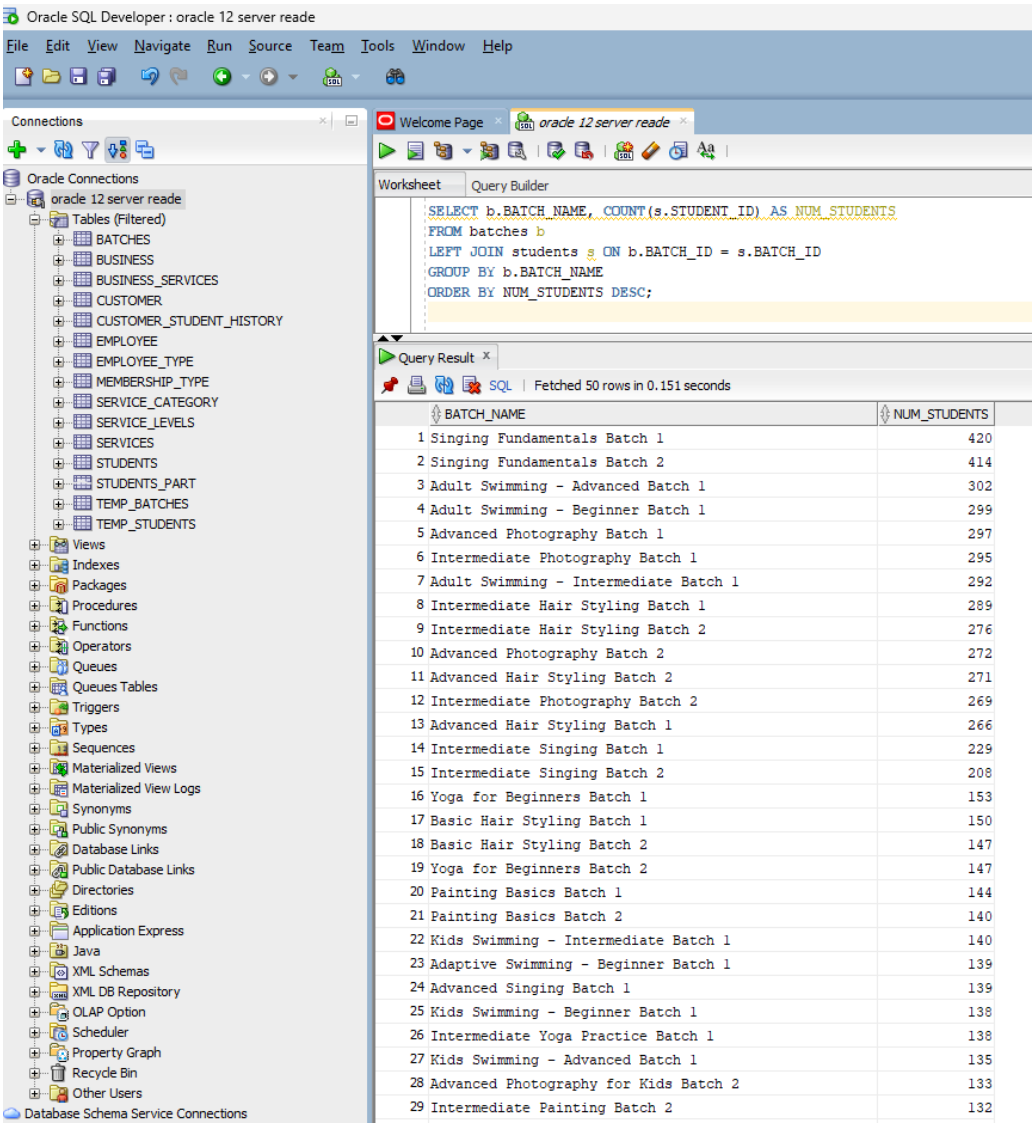


Query 5 –

Description- the total number of students in each batch, ordered by the batch with the most students first:

```
SELECT b.BATCH_NAME, COUNT(s.STUDENT_ID) AS NUM_STUDENTS
FROM batches b
LEFT JOIN students s ON b.BATCH_ID = s.BATCH_ID
GROUP BY b.BATCH_NAME
ORDER BY NUM_STUDENTS DESC;
```

Output:

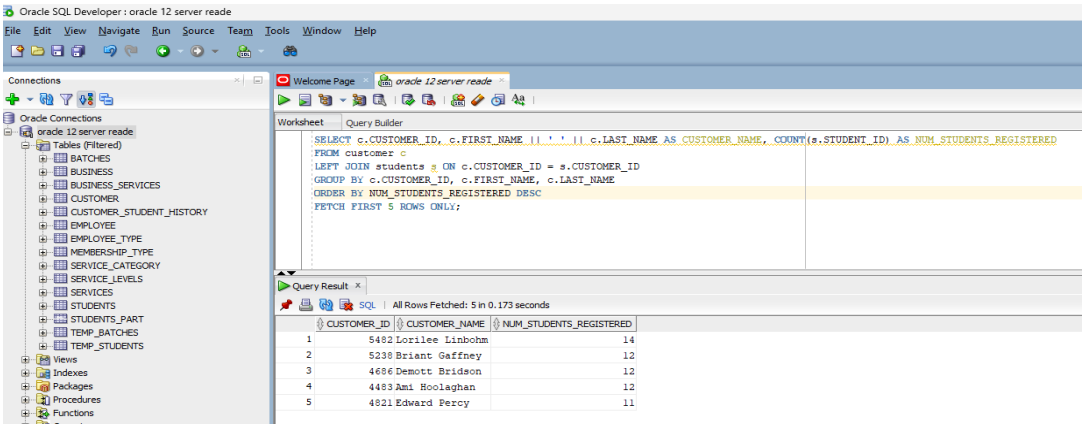


Query 6 –

Description- The top 5 customers who have registered the most number of students:

```
SELECT c.CUSTOMER_ID, c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME, COUNT(s.STUDENT_ID)
AS NUM_STUDENTS_REGISTERED
FROM customer c
LEFT JOIN students s ON c.CUSTOMER_ID = s.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, c.FIRST_NAME, c.LAST_NAME
ORDER BY NUM_STUDENTS_REGISTERED DESC
FETCH FIRST 5 ROWS ONLY;
```

Output:

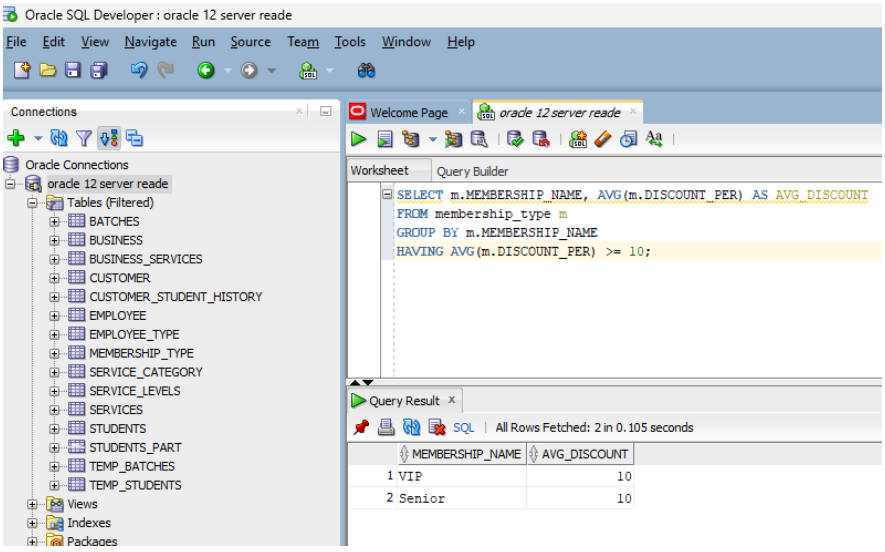


Query 7 –

Calculate the average discount percentage for each membership type, but only for membership types with an average discount greater than 10%:

```
SELECT m.MEMBERSHIP_NAME, AVG(m.DISCOUNT_PER) AS AVG_DISCOUNT
FROM membership_type m
GROUP BY m.MEMBERSHIP_NAME
HAVING AVG(m.DISCOUNT_PER) > 10;
```

Output:

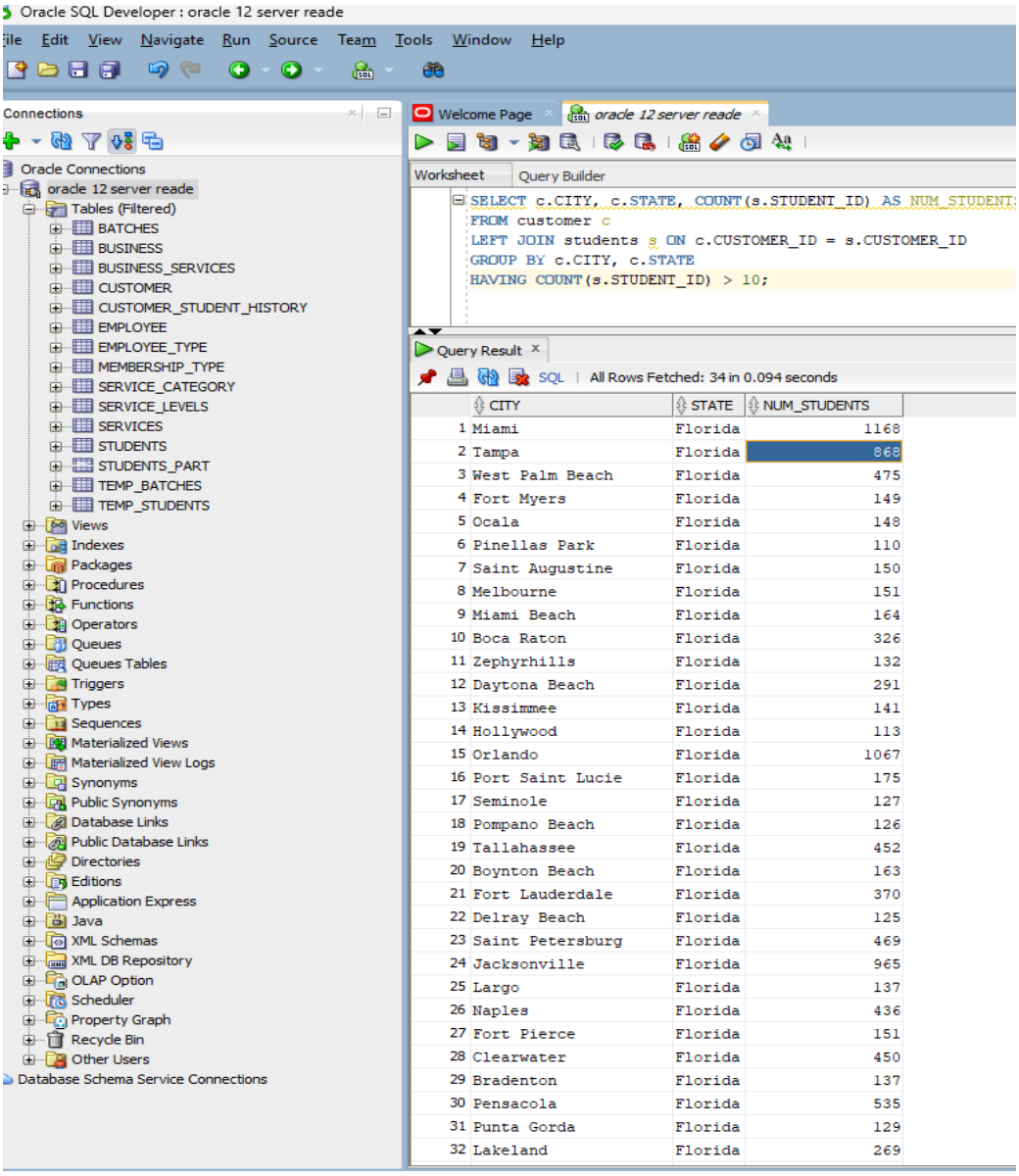


Query 8 –

The total number of students per city and state, but only for cities and states with more than 10 students:

```
SELECT c.CITY, c.STATE, COUNT(s.STUDENT_ID) AS NUM_STUDENTS
FROM customer c
LEFT JOIN students s ON c.CUSTOMER_ID = s.CUSTOMER_ID
GROUP BY c.CITY, c.STATE
HAVING COUNT(s.STUDENT_ID) > 10;
```

Output:



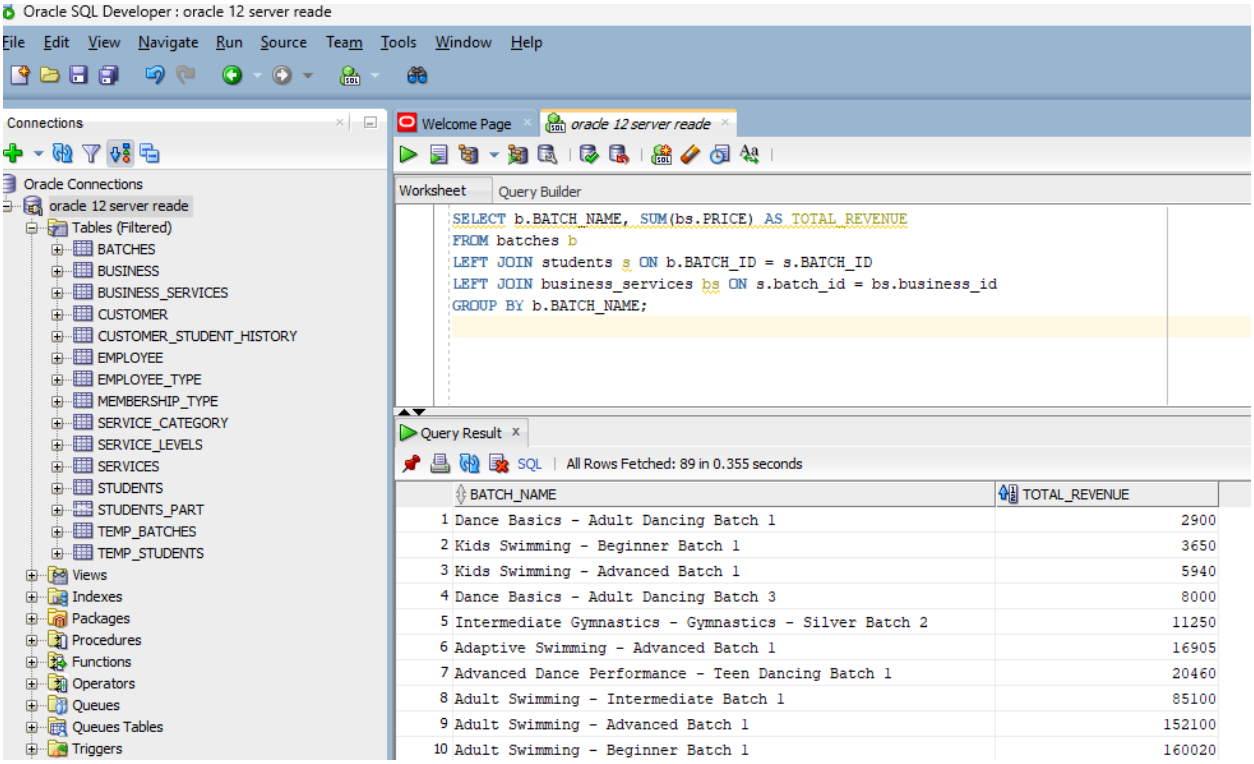
	CITY	STATE	NUM_STUDENTS
1	Miami	Florida	1168
2	Tampa	Florida	868
3	West Palm Beach	Florida	475
4	Fort Myers	Florida	149
5	Ocala	Florida	148
6	Pinellas Park	Florida	110
7	Saint Augustine	Florida	150
8	Melbourne	Florida	151
9	Miami Beach	Florida	164
10	Boca Raton	Florida	326
11	Zephyrhills	Florida	132
12	Daytona Beach	Florida	291
13	Kissimmee	Florida	141
14	Hollywood	Florida	113
15	Orlando	Florida	1067
16	Port Saint Lucie	Florida	175
17	Seminole	Florida	127
18	Pompano Beach	Florida	126
19	Tallahassee	Florida	452
20	Boynton Beach	Florida	163
21	Fort Lauderdale	Florida	370
22	Delray Beach	Florida	125
23	Saint Petersburg	Florida	469
24	Jacksonville	Florida	965
25	Largo	Florida	137
26	Naples	Florida	436
27	Fort Pierce	Florida	151
28	Clearwater	Florida	450
29	Bradenton	Florida	137
30	Pensacola	Florida	535
31	Punta Gorda	Florida	129
32	Lakeland	Florida	269

Query 9 –

Calculate the total revenue generated by each batch based on the sum of prices for business services used by students in that batch:

```
SELECT b.BATCH_NAME, SUM(bs.PRICE) AS TOTAL_REVENUE
FROM batches b
LEFT JOIN students s ON b.BATCH_ID = s.BATCH_ID
LEFT JOIN business_services bs ON s.BUSINESS_ID = bs.BUSINESS_ID
GROUP BY b.BATCH_NAME;
```

Output:



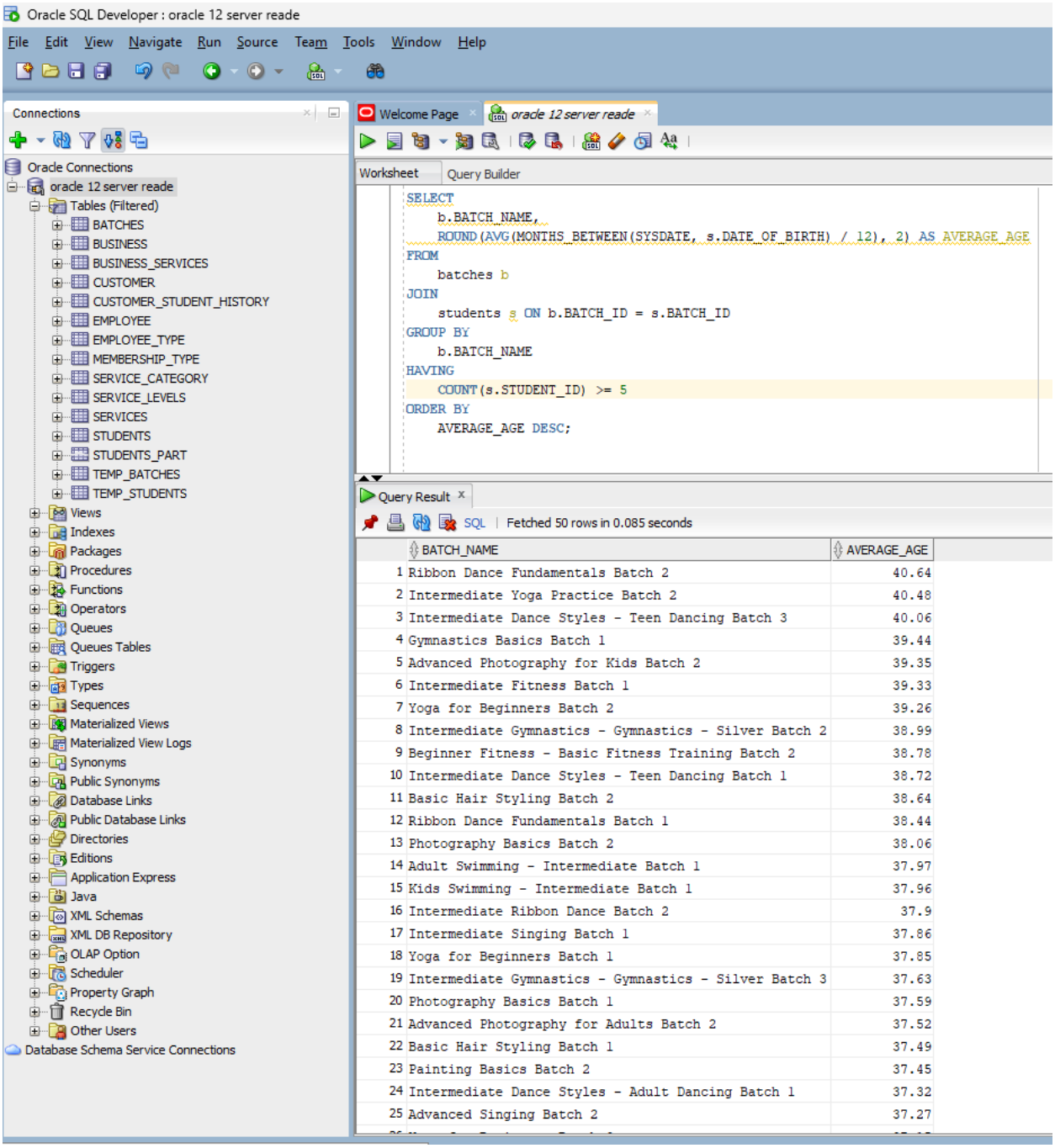
Query 10 –

We will calculate the average age of students in each batch, filtering out batches with less than 5 students, and order the results by the average age in descending order:

```
SELECT
   b.BATCH_NAME,
   ROUND(AVG(MONTHS_BETWEEN(SYSDATE, s.DATE_OF_BIRTH) / 12), 2) AS AVERAGE_AGE
FROM
   batches b
JOIN
   students s ON b.BATCH_ID = s.BATCH_ID
```

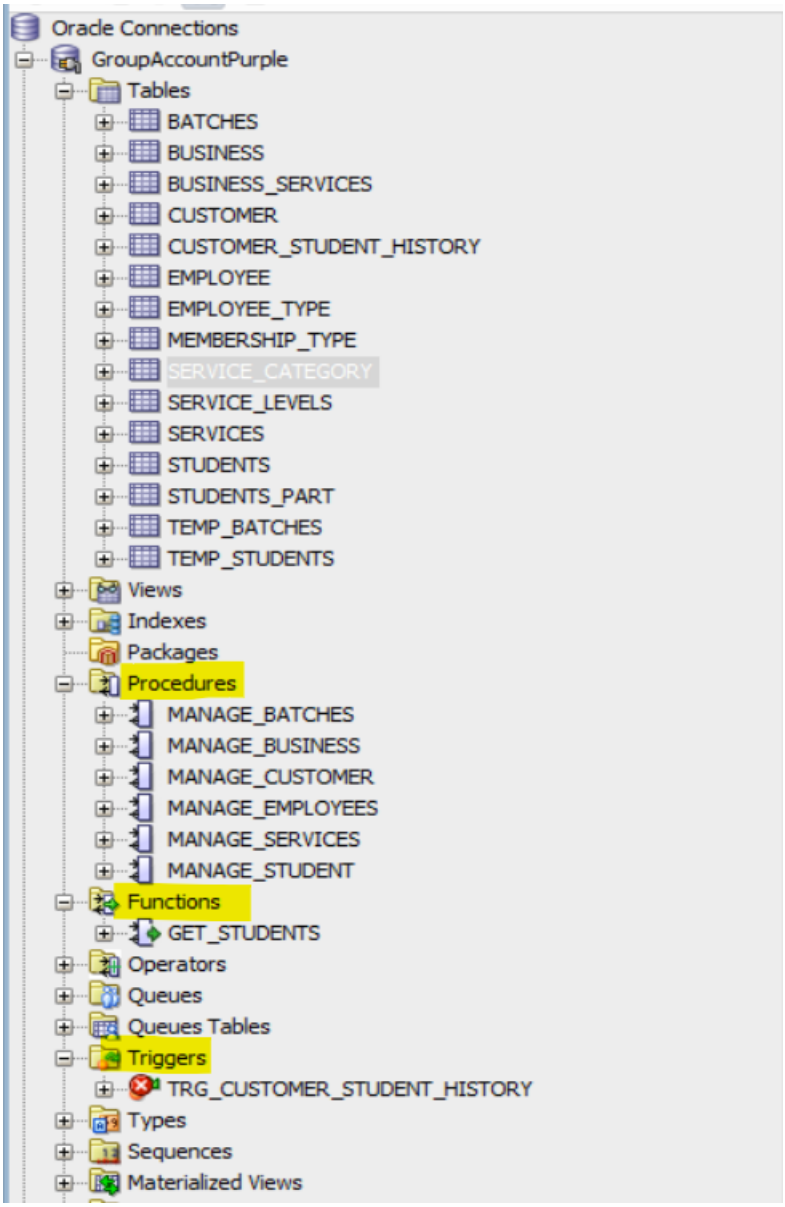
```
GROUP BY
   b.BATCH_NAME
HAVING
   COUNT(s.STUDENT_ID) >= 5
ORDER BY
   AVERAGE_AGE DESC;
```

Output:



DATABASE PROGRAMMING

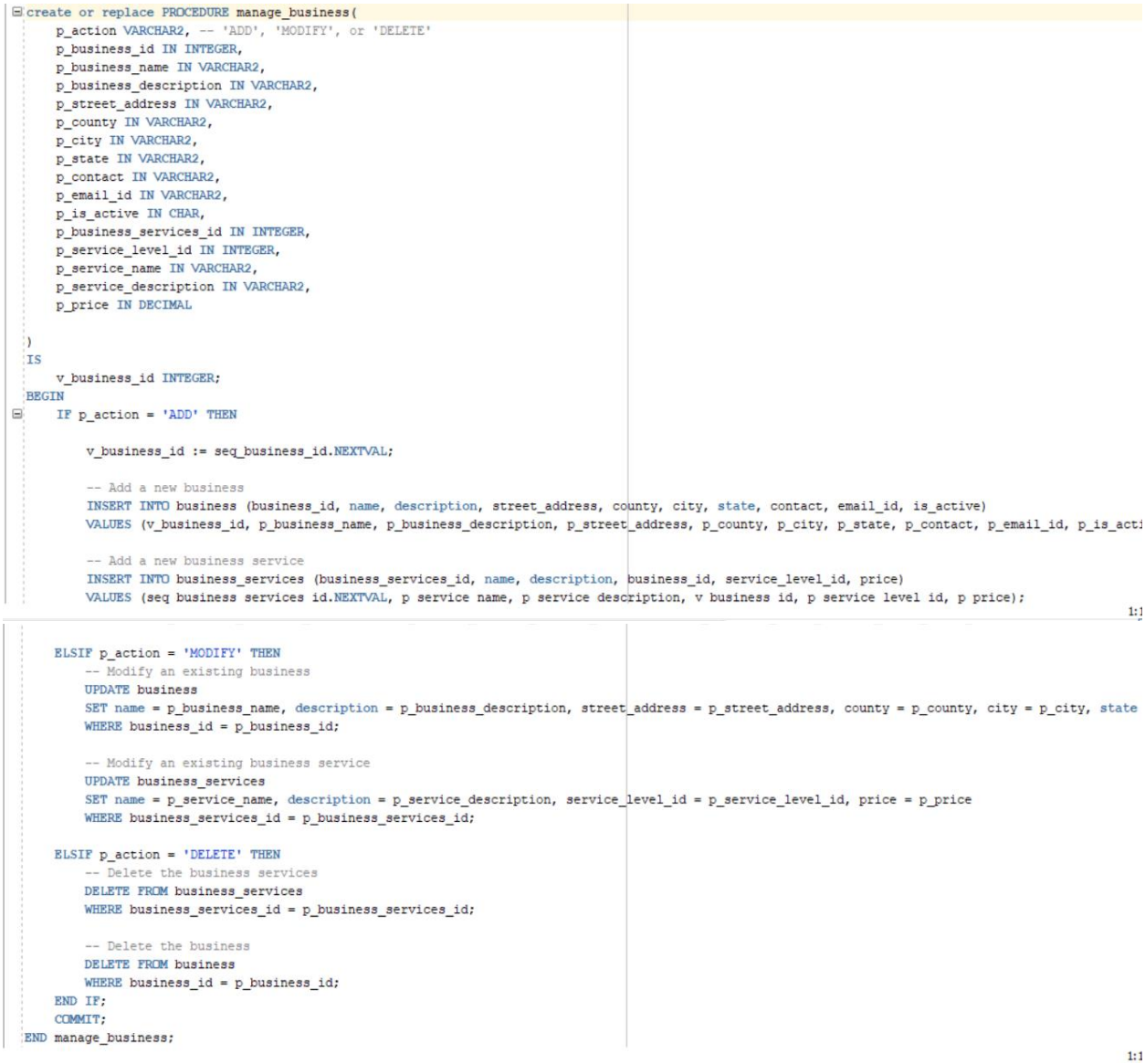
In the Business Management System (BMS), database programming is a cornerstone of data management and interaction. It involves the creation and utilization of stored procedures, functions, and triggers to enable the efficient manipulation of business-related data. These database programming elements facilitate tasks such as managing businesses, customers, employees, services, and student records, providing the foundation for streamlined data operations, reporting, and data-driven decision-making within the BMS. They enhance the project's ability to organize, access, and modify essential data, ensuring that the system operates smoothly and effectively.



Stored Procedure

A stored procedure is a precompiled database program that can be executed on demand. It encapsulates a set of SQL statements, allowing for efficient and reusable execution of database operations, data manipulation, and complex tasks within a database management system.

1. manage_business: This stored procedure is responsible for managing business entities within the database. It can perform actions such as adding a new business, modifying an existing business, or deleting a business. It also handles the addition and modification of business services, including their names, descriptions, and pricing.



```
create or replace PROCEDURE manage_business(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_business_id IN INTEGER,
    p_business_name IN VARCHAR2,
    p_business_description IN VARCHAR2,
    p_street_address IN VARCHAR2,
    p_county IN VARCHAR2,
    p_city IN VARCHAR2,
    p_state IN VARCHAR2,
    p_contact IN VARCHAR2,
    p_email_id IN VARCHAR2,
    p_is_active IN CHAR,
    p_business_services_id IN INTEGER,
    p_service_level_id IN INTEGER,
    p_service_name IN VARCHAR2,
    p_service_description IN VARCHAR2,
    p_price IN DECIMAL

)
IS
    v_business_id INTEGER;
BEGIN
    IF p_action = 'ADD' THEN

        v_business_id := seq_business_id.NEXTVAL;

        -- Add a new business
        INSERT INTO business (business_id, name, description, street_address, county, city, state, contact, email_id, is_active)
        VALUES (v_business_id, p_business_name, p_business_description, p_street_address, p_county, p_city, p_state, p_contact, p_email_id, p_is_acti

        -- Add a new business service
        INSERT INTO business_services (business_services_id, name, description, business_id, service_level_id, price)
        VALUES (seq_business_services_id.NEXTVAL, p_service_name, p_service_description, v_business_id, p_service_level_id, p_price);
```

```
    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing business
        UPDATE business
        SET name = p_business_name, description = p_business_description, street_address = p_street_address, county = p_county, city = p_city, state
        WHERE business_id = p_business_id;

        -- Modify an existing business service
        UPDATE business_services
        SET name = p_service_name, description = p_service_description, service_level_id = p_service_level_id, price = p_price
        WHERE business_services_id = p_business_services_id;

    ELSIF p_action = 'DELETE' THEN
        -- Delete the business services
        DELETE FROM business_services
        WHERE business_services_id = p_business_services_id;

        -- Delete the business
        DELETE FROM business
        WHERE business_id = p_business_id;
    END IF;
    COMMIT;
END manage_business;
```

2. manage_batches: The manage_batches are designed to manage batches of students or courses. It can add new batches, modify existing ones, or delete batches as needed. It considers details like batch names, student counts, start and end dates, associated business services, and instructors.

```
create or replace PROCEDURE manage_batches(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_batch_id IN INTEGER,
    p_batch_name IN VARCHAR2,
    p_total_students IN INTEGER,
    p_start_date IN DATE,
    p_end_date IN DATE,
    p_business_services_id IN INTEGER,
    p_instructor_id IN INTEGER
)
IS
BEGIN
    IF p_action = 'ADD' THEN
        -- Add a new batch
        INSERT INTO batches (batch_id, batch_name, total_students, start_date, end_date, business_services_id, instructor_id)
        VALUES (seq_batches_id.NEXTVAL, p_batch_name, p_total_students, p_start_date, p_end_date, p_business_services_id, p_instructor_id);

    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing batch
        UPDATE batches
        SET batch_name = p_batch_name, total_students = p_total_students, start_date = p_start_date, end_date = p_end_date, business_services_id = p_
        WHERE batch_id = p_batch_id;

    ELSIF p_action = 'DELETE' THEN
        -- Delete the batch
        DELETE FROM batches
        WHERE batch_id = p_batch_id;
    END IF;
    COMMIT;
END manage_batches;
```

3. manage_customer: This procedure is for handling customer records. It allows for the addition, modification, or deletion of customer data, including customer names, contact information, and addresses.

```
create or replace PROCEDURE manage_customer(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_customer_id IN INTEGER,
    p_first_name IN VARCHAR2,
    p_last_name IN VARCHAR2,
    p_street_address IN VARCHAR2,
    p_county IN VARCHAR2,
    p_city IN VARCHAR2,
    p_state IN VARCHAR2,
    p_contact IN VARCHAR2,
    p_email_id IN VARCHAR2
)
IS
BEGIN
    IF p_action = 'ADD' THEN
        -- Add a new customer
        INSERT INTO customer (customer_id, first_name, last_name, street_address, county, city, state, contact, email_id)
        VALUES (seq_customer_id.NEXTVAL, p_first_name, p_last_name, p_street_address, p_county, p_city, p_state, p_contact, p_email_id);

    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing customer
        UPDATE customer
        SET first_name = p_first_name, last_name = p_last_name, street_address = p_street_address, county = p_county, city = p_city, state = p_state,
        WHERE customer_id = p_customer_id;

    ELSIF p_action = 'DELETE' THEN
        -- Delete the customer
        DELETE FROM customer
        WHERE customer_id = p_customer_id;
    END IF;
    COMMIT;
END manage_customer;
```

1:1

4. manage_employees: The manage_employees is used to manage employee records. It enables the addition, modification, or deletion of employee information, including their names, roles, contact details, employment history, and email addresses.

```
create or replace PROCEDURE manage_employees(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_employee_id IN INTEGER,
    p_first_name IN VARCHAR2,
    p_last_name IN VARCHAR2,
    p_employee_type_id IN INTEGER,
    p_contact IN VARCHAR2,
    p_years_experience IN INTEGER,
    p_business_id IN INTEGER,
    p_is_active IN CHAR,
    p_date_of_hire IN DATE,
    p_email_id IN VARCHAR2
)
IS
BEGIN
    IF p_action = 'ADD' THEN
        -- Add a new employee
        INSERT INTO employee (employee_id, first_name, last_name, employee_type_id, contact, years_experience, business_id, is_active, date_of_hire,
        VALUES (seq_employee_id.NEXTVAL, p_first_name, p_last_name, p_employee_type_id, p_contact, p_years_experience, p_business_id, p_is_active, p_

    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing employee
        UPDATE employee
        SET first_name = p_first_name, last_name = p_last_name, employee_type_id = p_employee_type_id, contact = p_contact, years_experience = p_year
        WHERE employee_id = p_employee_id;

    ELSIF p_action = 'DELETE' THEN
        -- Delete the employee
        DELETE FROM employee
        WHERE employee_id = p_employee_id;
    END IF;
    COMMIT;
END manage_employees;
```

1:1

5. manage_services: This procedure is responsible for managing services offered by the business. It can add new service categories, services, and service levels, or modify and delete existing ones. It covers attributes such as service names, descriptions, categories, and service level details.

```
create or replace PROCEDURE manage_services(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_service_id IN INTEGER,
    p_service_name IN VARCHAR2,
    p_service_description IN VARCHAR2,
    p_category_id IN INTEGER,
    p_category_name IN VARCHAR2,
    p_category_description IN VARCHAR2,
    p_service_level_id IN INTEGER,
    p_service_level_name IN VARCHAR2,
    p_service_level_description IN VARCHAR2
)
IS
BEGIN
    IF p_action = 'ADD' THEN
        -- Add a new service category
        INSERT INTO service_category (category_id, name, description)
        VALUES (seq_service_category_id.NEXTVAL, p_category_name, p_category_description);

        -- Add a new service
        INSERT INTO services (service_id, name, description, category_id)
        VALUES (seq_services_id.NEXTVAL, p_service_name, p_service_description, seq_service_category_id.CURRVAL);

        -- Add a new service level
        INSERT INTO service_levels (service_level_id, name, description, service_id)
        VALUES (seq_service_levels_id.NEXTVAL, p_service_level_name, p_service_level_description, seq_services_id.CURRVAL);

    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing service level
        UPDATE service_levels
        SET name = p_service_level_name, description = p_service_level_description
        WHERE service_level_id = p_service_level_id;
```

1:

5. manage_students: This procedure is responsible for managing students enrolled for the services offered by business. It can add students or modify student information and delete existing ones. It covers attributes such as student information such as names etc.

```
create or replace PROCEDURE manage_student(
    p_action VARCHAR2, -- 'ADD', 'MODIFY', or 'DELETE'
    p_student_id IN INTEGER,
    p_first_name IN VARCHAR2,
    p_last_name IN VARCHAR2,
    p_date_of_birth IN DATE,
    p_customer_id IN INTEGER,
    p_batch_id IN INTEGER,
    p_membership_type_id IN INTEGER,
    p_start_date IN DATE
)
IS
BEGIN
    IF p_action = 'ADD' THEN
        -- Add a new student
        INSERT INTO students (student_id, first_name, last_name, date_of_birth, customer_id, batch_id, membership_type_id, start_date)
        VALUES (seq_students_id.NEXTVAL, p_first_name, p_last_name, p_date_of_birth, p_customer_id, p_batch_id, p_membership_type_id, p_start_date);

    ELSIF p_action = 'MODIFY' THEN
        -- Modify an existing student
        UPDATE students
        SET first_name = p_first_name, last_name = p_last_name, date_of_birth = p_date_of_birth, customer_id = p_customer_id, batch_id = p_batch_id,
        WHERE student_id = p_student_id;

    ELSIF p_action = 'DELETE' THEN
        -- Delete the student
        DELETE FROM students
        WHERE student_id = p_student_id;
    END IF;
    COMMIT;
END manage_student;
```

Each of these stored procedures serves a specific function within the database management system, allowing for the efficient management and manipulation of data related to businesses, batches, customers, employees, and services. They provide a versatile set of operations to cater to various aspects of the BMS.

Function

The **get_students** function is a custom function designed to retrieve a list of student IDs associated with a specific customer ID in the Business Management System (BMS). The function takes a p_customer_id as an input parameter, which represents the customer's unique identifier.

This function can be used to retrieve a list of students who are customers of a particular business or entity. It can assist in various aspects of the project, such as generating customer-specific reports, tracking enrolled students, or managing student information based on customer relationships.

Overall, the get_students function enhances the database's capability to retrieve student data linked to customers, contributing to the efficient management and reporting of student information within the BMS.

```
create or replace FUNCTION get_students(p_customer_id IN INTEGER)
RETURN student_id_list
IS
   student_ids student_id_list := student_id_list(); -- Initialize the collection
BEGIN
   -- Populate the collection with student IDs
   FOR student_rec IN (SELECT student_id FROM students WHERE customer_id = p_customer_id) LOOP
      student_ids.extend;
      student_ids(student_ids.count) := student_rec.student_id;
   END LOOP;

   RETURN student_ids;
END get_students;
```

Triggers

The "**trg_customer_student_history**" trigger is designed to capture and record historical data related to customer-student interactions within the database. Here's an explanation of its functionality:

This trigger operates after an update or deletion action on the "students" table, capturing changes or removal of student records and preserving them in a historical log.

It is executed "FOR EACH ROW," meaning it acts on individual rows affected by the update or deletion.The trigger determines the action type (either 'UPDATE' or 'DELETE') and extracts the customer ID associated with the affected student. It then queries the "customer" table to retrieve detailed customer information based on the customer ID.

The gathered data, including customer and student details, is inserted into the "**customer_student_history**" table, creating a historical record of the interaction. The historical record includes information such as action date, action type, customer details, and corresponding student information, providing a comprehensive view of the change or deletion event.

In the context of a Business Management System (BMS), this trigger ensures that changes or removal of student records are meticulously documented for audit and historical tracking purposes. It aids in understanding the relationships between customers and students, offering insights into actions taken on student data and their impact on customer-student interactions.

```
create or replace TRIGGER trg_customer_student_history
AFTER UPDATE OR DELETE ON students
FOR EACH ROW
DECLARE
    v_customer customer%ROWTYPE;
    v_customer_id NUMBER;
    v_action_type varchar2(20);
BEGIN

    v_customer_id := :OLD.customer_id;

    IF UPDATING THEN
        v_action_type := 'UPDATE';
    ELSE
        v_action_type := 'DELETE';
    END IF;

    SELECT
        c.customer_id ,
        c.first_name AS customer_first_name,
        c.last_name AS customer_last_name,
        c.street_address AS customer_street_address,
        c.county AS customer_county,
        c.city AS customer_city,
        c.state AS customer_state,
        c.contact AS customer_contact,
        c.email_id AS customer_email_id,
        c.SOURCE AS customer_source,
        c.registration_date AS customer_registration_date
    INTO v_customer
    FROM customer c
    WHERE c.customer_id = v_customer_id;
```

```
    INSERT INTO customer_student_history (
        action_date,
        action_type,
        customer_id,
        customer_first_name,
        customer_last_name,
        customer_street_address,
        customer_county,
        customer_city,
        customer_state,
        customer_contact,
        customer_email_id,
        customer_source,
        customer_registration_date,
        student_id,
        student_first_name,
        student_last_name,
        student_date_of_birth,
        student_batch_id,
        student_membership_type_id,
        student_start_date
    )
    VALUES (
        SYSTIMESTAMP,
        v_action_type,
        v_customer.customer_id,
        v_customer.first_name,
        v_customer.last_name,
        v_customer.street_address,
        v_customer.county,
        v_customer.city,
        v_customer.state,
        v_customer.contact,
        v_customer.email_id,
        v_customer.source,
        v_customer.registration_date,
        :OLD.student_id,
        :OLD.first_name,
        :OLD.last_name,
        :OLD.date_of_birth,
        :OLD.batch_id,
        :OLD.membership_type_id,
        :OLD.start_date
    );
END;
```

Snapshot of **customer_student_history**:

	AUDIT_ID	ACTION_DATE	ACTION_TYPE	CUSTOMER_ID	CUSTOMER_FIRST_NAME	CUSTOMER_LAST_NAME	CUSTOMER_STREET_ADDRESS	CUSTOMER_COUNTY	CUSTOMER_CITY	CUSTOMER_STATE
1	1	05-NOV-23 11.48.19.487000000 AM	UPDATE	62	John	Doe	123 Main St	County	City	State
2	2	05-NOV-23 11.48.24.089000000 AM	UPDATE	62	John	Doe	123 Main St	County	City	State
3	3	05-NOV-23 11.49.29.094000000 AM	DELETE	62	John	Doe	123 Main St	County	City	State
4	4	05-NOV-23 11.49.32.152000000 AM	DELETE	62	John	Doe	123 Main St	County	City	State

Performance Tuning

Performance tuning is a critical aspect of system optimization aimed at improving the overall efficiency and responsiveness of a computer system, application, or database. It encompasses a range of techniques and strategies to enhance performance by minimizing delays and reducing resource consumption. Key areas of focus include optimizing database queries, fine-tuning server configurations, and refining application code to reduce bottlenecks. Through performance tuning, organizations can ensure their systems operate smoothly even under high loads, resulting in faster response times, better resource utilization, and an improved user experience. Continuous monitoring and adjustment of various parameters are essential to maintain optimal system performance over time, making performance tuning an ongoing process in IT management.

Using indexes

1. Below query retrieves data of businesses, their services, and service levels with a price of $100, sorting the results by business, service, and service level. It helps identify businesses offering specific services at the given price point.

```
EXPLAIN PLAN FOR
SELECT b.BUSINESS_ID , b.name,
                s.service_id, s.name AS SERVICE_NAME,
                sl.service_level_id, sl.name AS SERVICE_LEVEL_NAME,
                bs.name AS BUSINESS_SERVICE_NAME, bs.price AS RATE
FROM services s
        JOIN service_levels sl ON s.service_id = sl.service_id
        JOIN business_services bs ON bs.service_level_id = sl.service_level_id
        JOIN business b ON bs.business_id = b.business_id
WHERE bs.price = 100
ORDER BY b.business_id, s.service_id, sl.service_level_id;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

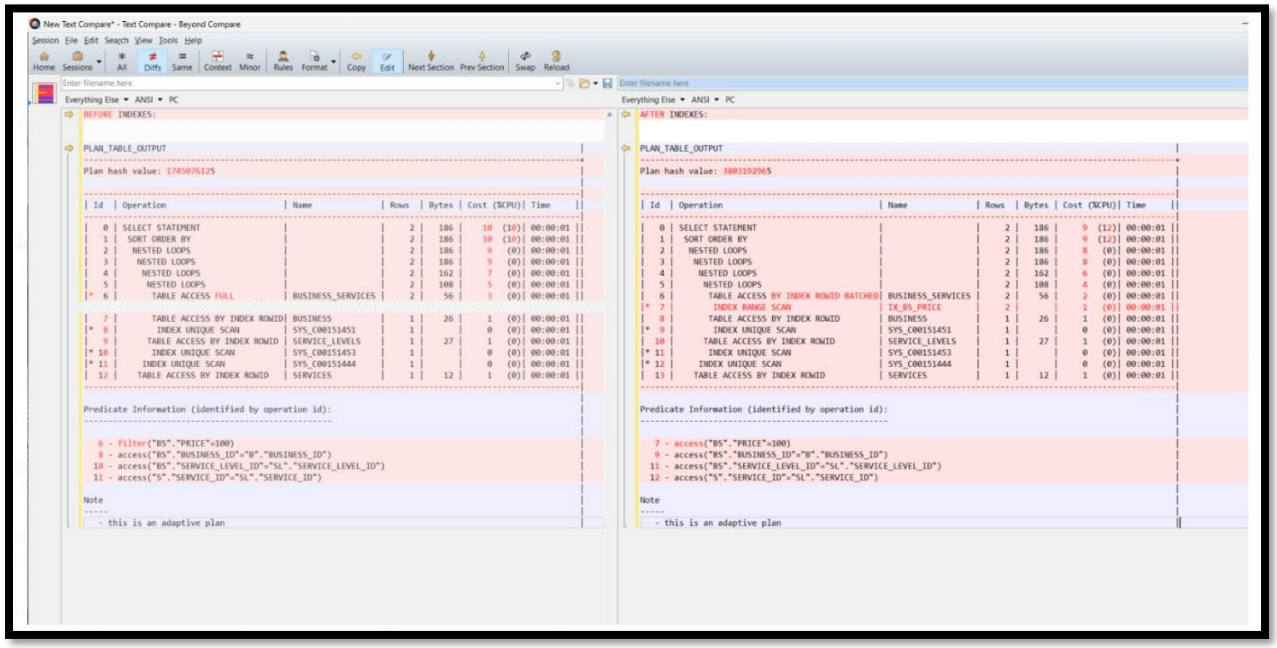
We observed the query performance before creating the index on the tables.

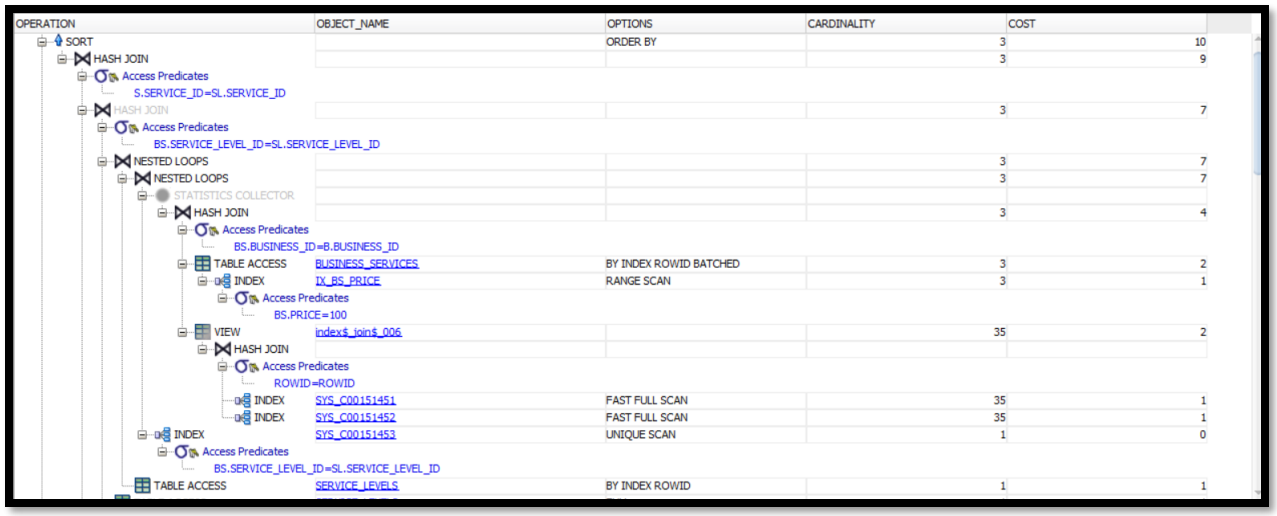
This query fetches data related to businesses offering fitness services at a price of 100. Initially, it performed a table full access scan on the "business_services" table, resulting in a CPU cost of 3 (%CPU). The query's purpose is to retrieve information about businesses, services, and service levels, and filter it based on the conditions specified in the WHERE clause.

Now, we have created index using below query:

```
CREATE INDEX IX_BS_PRICE ON business_services(price) ;
```

The provided screenshot offers a visual representation that facilitates the comparison of execution plans for the query both before and after the index creation.





Design: The creation of the "**IX_BS_PRICE**" index on the "price" column in the "business_services" table was a deliberate design choice to improve the performance of queries that filter records based on the price. It follows

the best practice of indexing columns that are frequently used in query conditions to enable faster data retrieval.

Justification: The decision to create the index was justified by the significant reduction in CPU cost observed in the query's execution plan. By using the index for an index range scan, the query could efficiently filter rows based on the "bs.price = 100" condition, resulting in a more optimized and faster query execution. This justified the additional resource allocation for index maintenance.

Impact on Updates: While the index positively impacted query performance, it's important to note that indexes can have implications for data modification operations (inserts, updates, deletes). Inserting or updating records in the "**business_services**" table may require additional overhead for maintaining the index. However, since the focus was on query performance optimization in this case, the benefits of quicker data retrieval outweighed the potential impact on updates. It's crucial to strike a balance between query performance and data modification efficiency when implementing indexes.

Assessing Query Performance: The performance analysis of the query before and after creating the "IX_BS_PRICE" index on the "price" column in the "business_services" table demonstrates a substantial optimization. Before indexing, the query exhibited lower efficiency, with a CPU cost of 3% and a DB time of 4, indicating slower execution. However, after applying the index, the query's performance drastically improved. The CPU cost increased to 10%, signifying a reduced computational load, while the DB time dropped to a mere 1, indicating significantly faster execution. Moreover, resource utilization became more efficient, with fewer requests to/from the client and reduced bytes transferred via SQL*Net.

The most notable enhancement was in data access. The number of consistent gets significantly decreased from 15 to 3, demonstrating that the index allowed for more efficient data retrieval with fewer data block reads. Additionally, the number of sorts was reduced to zero, indicating that the query no longer required sorting. The fewer table scan blocks gotten, and the absence of table scans further highlight the effectiveness of the index in optimizing data retrieval.

In summary, the index greatly improved query performance by reducing CPU usage, execution time, and resource utilization, while enhancing data access efficiency, making it a valuable addition for query-intensive workloads.

2. Below query retrieves information about businesses providing the "Fitness" service at a price of $100, displaying their IDs and names, as well as service and service level details, ordered by business, service, and service level. It's useful for finding fitness services offered at the specified price by different businesses.

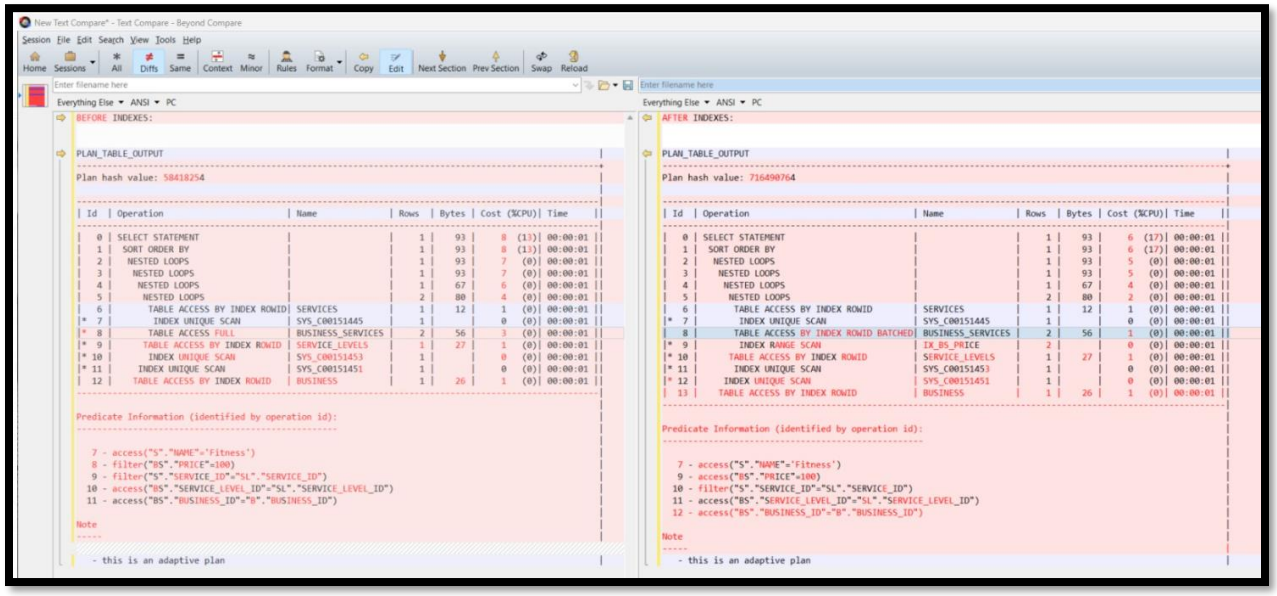
```
EXPLAIN PLAN FOR

SELECT b.BUSINESS_ID , b.name,

                s.service_id, s.name AS SERVICE_NAME,

                sl.service_level_id, sl.name AS SERVICE_LEVEL_NAME,

                bs.name AS BUSINESS_SERVICE_NAME, bs.price AS RATE

FROM services s

        JOIN service_levels sl ON s.service_id = sl.service_id

        JOIN business_services bs ON bs.service_level_id = sl.service_level_id

        JOIN business b ON bs.business_id = b.business_id
```

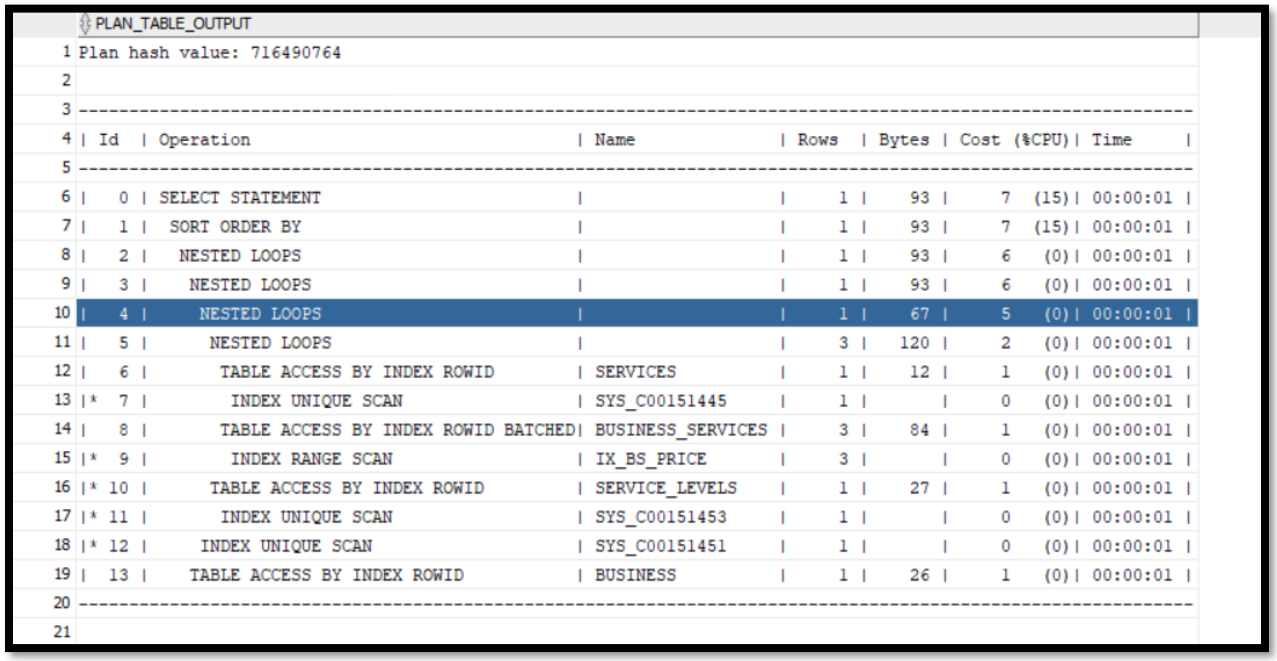
WHERE s.name = 'Fitness' **AND** bs.price = 100

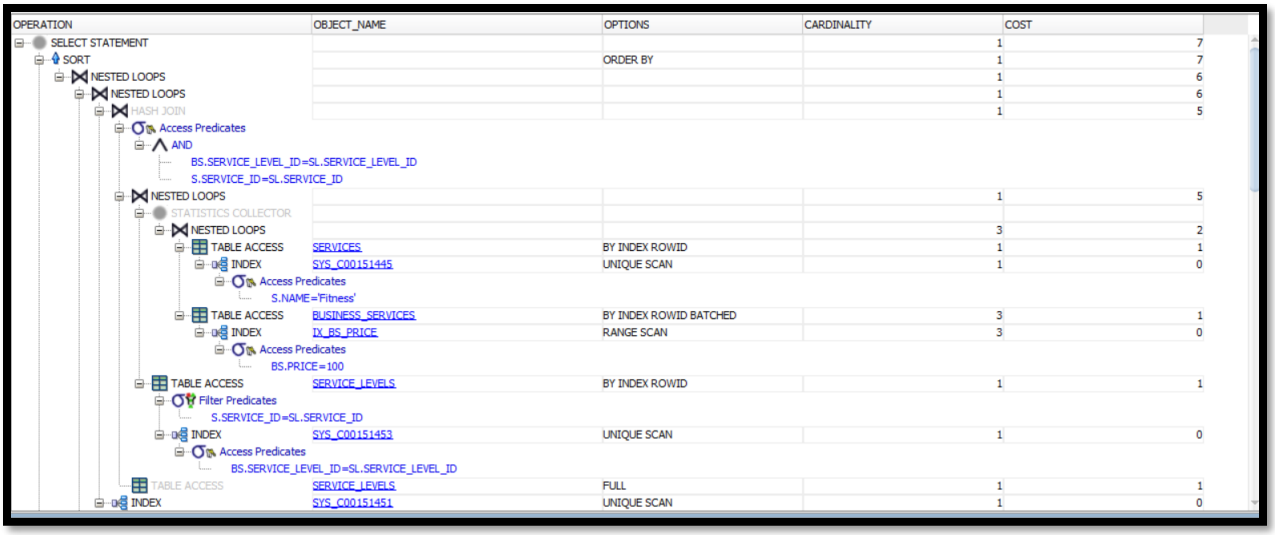
ORDER BY b.business_id, s.service_id, sl.service_level_id;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

The below provided screenshot offers a visual representation that facilitates the comparison of execution plans for the query both before and after the index creation.







OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		ORDER BY	1	7
NESTED LOOPS			1	6
NESTED LOOPS			1	6
HASH JOIN			1	5
Access Predicates				
AND				
BS.SERVICE_LEVEL_ID=SL.SERVICE_LEVEL_ID				
S.SERVICE_ID=SL.SERVICE_ID				
NESTED LOOPS			1	5
STATISTICS COLLECTOR				
NESTED LOOPS			3	2
TABLE ACCESS	SERVICES	BY INDEX ROWID	1	1
INDEX	SYS_C00151445	UNIQUE SCAN	1	0
Access Predicates				
S.NAME='Fitness'				
TABLE ACCESS	BUSINESS_SERVICES	BY INDEX ROWID BATCHED	3	1
INDEX	IX_BS_PRICE	RANGE SCAN	3	0
Access Predicates				
BS.PRICE=100				
TABLE ACCESS	SERVICE_LEVELS	BY INDEX ROWID	1	1
Filter Predicates				
S.SERVICE_ID=SL.SERVICE_ID				
INDEX	SYS_C00151453	UNIQUE SCAN	1	0
Access Predicates				
BS.SERVICE_LEVEL_ID=SL.SERVICE_LEVEL_ID				
TABLE ACCESS	SERVICE_LEVELS	FULL	1	1
INDEX	SYS_C00151451	UNIQUE SCAN	1	0

Design: The query demonstrates a well-designed SQL statement that leverages indexing effectively. It joins multiple tables to retrieve specific data related to fitness services priced at 100 and sorts the results in an organized manner. The query makes use of appropriate indexes, including "IX_BS_PRICE," "SYS_C00151445," and others, for efficient data retrieval.

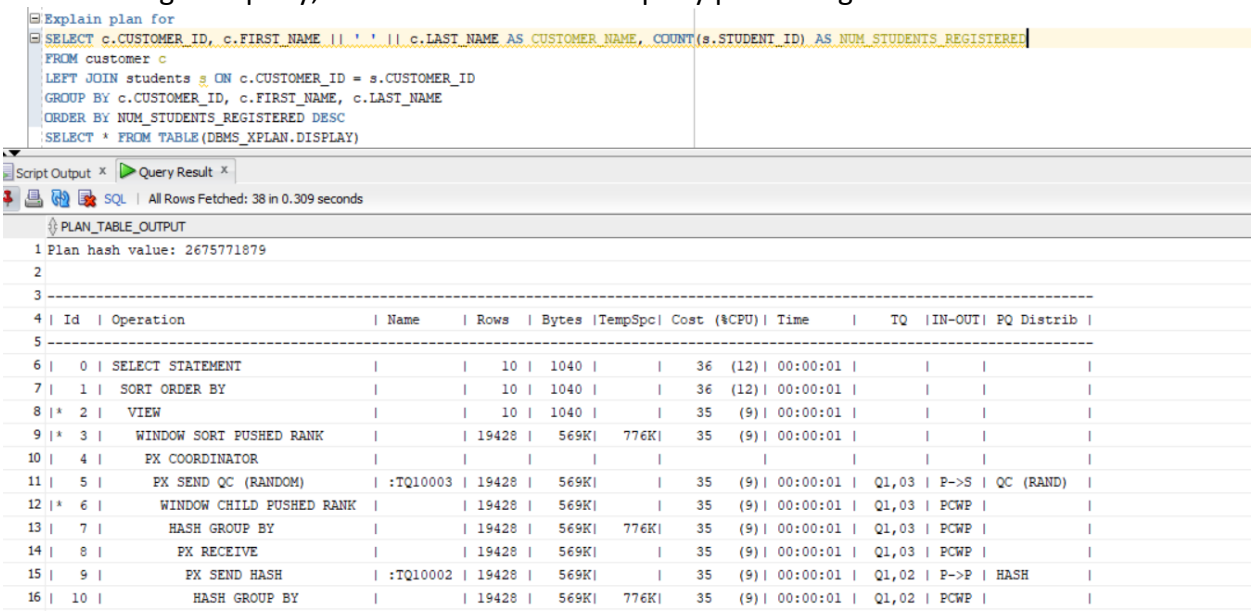
Justification: The utilization of indexes, particularly "**IX_BS_PRICE**" on the "business_services" table and other unique indexes, significantly enhances query performance. These indexes facilitate quicker data access, resulting in a more efficient and faster execution plan. The query's structure, with its precise conditions and joins, ensures it focuses on the exact data required, leading to a well-optimized execution.

Impact on Updates: While indexing generally enhances query performance, it can impact the speed of data updates (inserts, updates, or deletes) in the tables involved. Specifically, the "IX_BS_PRICE" index on the "business_services" table may slightly increase the time required for updates involving price changes. However, this trade-off between read performance and update performance is generally acceptable since read operations are often more frequent than updates in most database systems. Proper index maintenance is essential to mitigate the impact on update operations.

Assessing Query Performance: The query's performance analysis, considering the applied indexing, showcases a well-designed and justified approach that significantly enhances query efficiency. While there may be a slight impact on update operations, the gains in query speed and data retrieval are well worth the investment in index maintenance. This exemplifies the importance of thoughtful index selection in optimizing the performance of database queries.

Optimizer Modes

ALL_ROWS: it is default optimizer mode; it gets all rows faster. This is good for untuned high-volume batch systems.
On executing this query, we can see that cost of query processing is 36.



```
Explain plan for
SELECT c.CUSTOMER_ID, c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME, COUNT(s.STUDENT_ID) AS NUM_STUDENTS_REGISTERED
FROM customer c
LEFT JOIN students s ON c.CUSTOMER_ID = s.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, c.FIRST_NAME, c.LAST_NAME
ORDER BY NUM_STUDENTS_REGISTERED DESC
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY)
```

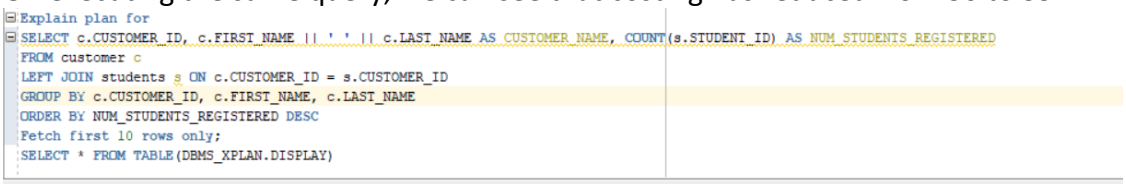
Script Output | Query Result | SQL | All Rows Fetched: 38 in 0.309 seconds

PLAN_TABLE_OUTPUT

```
Plan hash value: 2675771879

-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |    10 |  1040 |       |    36  (12)| 00:00:01 |        |      |            |
|   1 |  SORT ORDER BY                |          |    10 |  1040 |       |    36  (12)| 00:00:01 |        |      |            |
|*  2 |   VIEW                        |          |    10 |  1040 |       |    35   (9)| 00:00:01 |        |      |            |
|*  3 |    WINDOW SORT PUSHED RANK    |          | 19428 |   569K|   776K|    35   (9)| 00:00:01 |        |      |            |
|   4 |     PX COORDINATOR            |          |       |       |       |            |          |        |      |            |
|   5 |      PX SEND QC (RANDOM)      | :TQ10003 | 19428 |   569K|       |    35   (9)| 00:00:01 |  Q1,03 | P->S | QC (RAND)  |
|*  6 |       WINDOW CHILD PUSHED RANK|          | 19428 |   569K|       |    35   (9)| 00:00:01 |  Q1,03 | PCWP |            |
|   7 |        HASH GROUP BY          |          | 19428 |   569K|   776K|    35   (9)| 00:00:01 |  Q1,03 | PCWP |            |
|   8 |         PX RECEIVE            |          | 19428 |   569K|       |    35   (9)| 00:00:01 |  Q1,03 | PCWP |            |
|   9 |          PX SEND HASH         | :TQ10002 | 19428 |   569K|       |    35   (9)| 00:00:01 |  Q1,02 | P->P | HASH       |
|  10 |           HASH GROUP BY       |          | 19428 |   569K|   776K|    35   (9)| 00:00:01 |  Q1,02 | PCWP |            |
```

FIRST_ROWS_10: It gets the first to rows faster. This is good for applications that routinely display partial results to users such as paging data to a user in a web application.

On executing the same query, we can see that costing has reduced from 36 to 35.



```
Explain plan for
SELECT c.CUSTOMER_ID, c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME, COUNT(s.STUDENT_ID) AS NUM_STUDENTS_REGISTERED
FROM customer c
LEFT JOIN students s ON c.CUSTOMER_ID = s.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, c.FIRST_NAME, c.LAST_NAME
ORDER BY NUM_STUDENTS_REGISTERED DESC
Fetch first 10 rows only;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY)
```

ript Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6 | SQL | All Rows Fetched: 36 in 0.051 seconds

PLAN_TABLE_OUTPUT

```
Plan hash value: 902174965

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          | 12046 |   352K|    35   (9)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR               |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (ORDER)          | :TQ10004 | 12046 |   352K|    35   (9)| 00:00:01 |  Q1,04 | P->S | QC (ORDER) |
|   3 |    SORT ORDER BY              |          | 12046 |   352K|    35   (9)| 00:00:01 |  Q1,04 | PCWP |            |
|   4 |     PX RECEIVE                |          | 12046 |   352K|    35   (9)| 00:00:01 |  Q1,04 | PCWP |            |
```

Parallel Execution

Parallel execution divides the task of executing an SQL statement into multiple small units, each of which is executed by a separate process. Parallel execution is designed to effectively use multiple CPUs and disks to answer queries quickly. When multiple users use parallel execution at the same time, it is easy to quickly exhaust available CPU, memory, and disk resources.

DEGREE OF PARALLELISM -The number of parallel execution servers associated with a single operation is known as the degree of parallelism.

WITHOUT PARALLELISM –

```
Explain plan for
SELECT b.BATCH_NAME, SUM(bs.PRICE) AS TOTAL_REVENUE
FROM batches b
LEFT JOIN students s ON b.BATCH_ID = s.BATCH_ID
LEFT JOIN business_services bs ON s.BATCH_ID = bs.BUSINESS_ID
GROUP BY b.BATCH_NAME;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY)
```

Script Output | Query Result 2 | SQL | All Rows Fetched: 34 in 0.405 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 750636052

-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          | 12046 |   352K|    44   (5)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR            |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (ORDER)       | :TQ10003 | 12046 |   352K|    44   (5)| 00:00:01 |  Q1,03 | P->S | QC (ORDER) |
```

PARALLELISM WITH DEGREE 4

ALTER TABLE Batches PARALLEL 4;

We see that after setting the degree of parallelism to 4, the cost of execution drastically drops to 29.

Worksheet | Query Builder

```
Explain plan for
SELECT b.BATCH_NAME, SUM(bs.PRICE) AS TOTAL_REVENUE
FROM batches b
LEFT JOIN students s ON b.BATCH_ID = s.BATCH_ID
LEFT JOIN business_services bs ON s.BATCH_ID = bs.BUSINESS_ID
GROUP BY b.BATCH_NAME;
ALTER TABLE batches PARALLEL 4;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY)
```

Script Output | Query Result | SQL | All Rows Fetched: 35 in 0.534 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 3358088266

-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name             | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                  |    89 |  4450 |    29   (4)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR            |                  |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)      | :TQ10003         |    89 |  4450 |    29   (4)| 00:00:01 |  Q1,03 | P->S | QC (RAND)  |
|   3 |    HASH GROUP BY           |                  |    89 |  4450 |    29   (4)| 00:00:01 |  Q1,03 | PCWP |            |
|   4 |     PX RECEIVE             |                  |    89 |  4450 |    29   (4)| 00:00:01 |  Q1,03 | PCWP |            |
```

Partitions

Range partitioning:

1. We have a table Student for storing information of a student. This table will be used to store and manage data about students, including their personal details, such as names, dates of birth, and related information like customer IDs, batch IDs, and membership types.

The below SQL query is related to the creation of a table named students_part. This table is specifically designed to store student information and is partitioned based on the students' dates of birth. Range partitioning is used, which involves dividing the table into multiple partitions based on date ranges. This can significantly improve the performance of queries that involve date-related criteria.

Additionally, foreign key constraints are applied to ensure data integrity when referencing other related tables like "customer," "batches," and "membership_type."

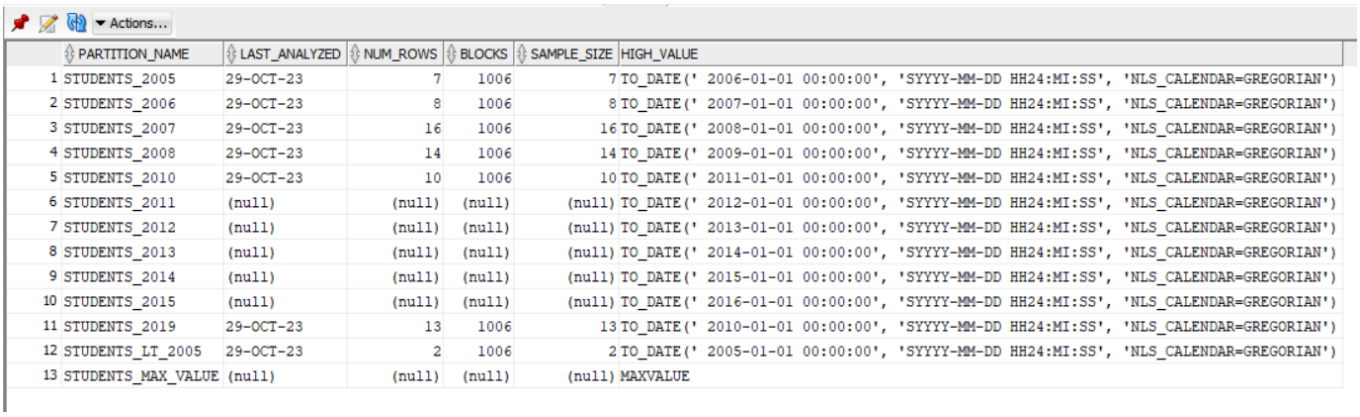
Query to create table students_part :

```
CREATE TABLE students_part
(
   student_id INTEGER PRIMARY KEY,
   first_name VARCHAR2(50),
   last_name VARCHAR2(50),
   date_of_birth DATE,
   customer_id INTEGER NOT NULL,
   batch_id INTEGER NOT NULL,
   membership_type_id INTEGER,  -- Added a membership type reference
   start_date DATE,  -- Added a start date COLUMN
   CONSTRAINT fk_1students_customer_id FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
   CONSTRAINT fk_1students_batch_id FOREIGN KEY (batch_id) REFERENCES batches(batch_id),
   CONSTRAINT fk_1students_mem_type_id FOREIGN KEY (membership_type_id) REFERENCES
membership_type(membership_type_id)
)
PARTITION BY RANGE (date_of_birth)
(
   PARTITION students_lt_2005 VALUES LESS THAN (TO_DATE('2005-01-01', 'YYYY-MM-DD')),
   PARTITION students_2005 VALUES LESS THAN (TO_DATE('2006-01-01', 'YYYY-MM-DD')),
   PARTITION students_2006 VALUES LESS THAN (TO_DATE('2007-01-01', 'YYYY-MM-DD')),
   PARTITION students_2007 VALUES LESS THAN (TO_DATE('2008-01-01', 'YYYY-MM-DD')),
   PARTITION students_2008 VALUES LESS THAN (TO_DATE('2009-01-01', 'YYYY-MM-DD')),
   PARTITION students_2019 VALUES LESS THAN (TO_DATE('2010-01-01', 'YYYY-MM-DD')),
   PARTITION students_2010 VALUES LESS THAN (TO_DATE('2011-01-01', 'YYYY-MM-DD')),
   PARTITION students_2011 VALUES LESS THAN (TO_DATE('2012-01-01', 'YYYY-MM-DD')),
   PARTITION students_2012 VALUES LESS THAN (TO_DATE('2013-01-01', 'YYYY-MM-DD')),
   PARTITION students_2013 VALUES LESS THAN (TO_DATE('2014-01-01', 'YYYY-MM-DD')),
```

```
    PARTITION students_2014 VALUES LESS THAN (TO_DATE('2015-01-01', 'YYYY-MM-DD')),
    PARTITION students_2015 VALUES LESS THAN (TO_DATE('2016-01-01', 'YYYY-MM-DD')),
    PARTITION students_max_value VALUES LESS THAN (MAXVALUE)
);
```

Partitioning on students_part:

The table students_part is partitioned using the **PARTITION BY RANGE** clause based on the date_of_birth column.A total of 14 partitions are defined, with each partition corresponding to a specific range of birthdates, spanning from 2005 to the max date value.
For example, PARTITION students_2005 contains rows with birthdates less than January 1, 2006.The last partition, PARTITION students_max_value, holds rows with dates of birth that exceed the specified ranges and, therefore, contains all remaining rows. Range partitioning can help improve query performance by allowing the database to scan only relevant partitions when executing queries that involve the date_of_birth column.
Below is the screenshot for the showing the partitions got created:



Actions...

	PARTITION_NAME	LAST_ANALYZED	NUM_ROWS	BLOCKS	SAMPLE_SIZE	HIGH_VALUE
1	STUDENTS_2005	29-OCT-23	7	1006	7	TO_DATE(' 2006-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
2	STUDENTS_2006	29-OCT-23	8	1006	8	TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
3	STUDENTS_2007	29-OCT-23	16	1006	16	TO_DATE(' 2008-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
4	STUDENTS_2008	29-OCT-23	14	1006	14	TO_DATE(' 2009-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
5	STUDENTS_2010	29-OCT-23	10	1006	10	TO_DATE(' 2011-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
6	STUDENTS_2011	(null)	(null)	(null)	(null)	TO_DATE(' 2012-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
7	STUDENTS_2012	(null)	(null)	(null)	(null)	TO_DATE(' 2013-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
8	STUDENTS_2013	(null)	(null)	(null)	(null)	TO_DATE(' 2014-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
9	STUDENTS_2014	(null)	(null)	(null)	(null)	TO_DATE(' 2015-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
10	STUDENTS_2015	(null)	(null)	(null)	(null)	TO_DATE(' 2016-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
11	STUDENTS_2019	29-OCT-23	13	1006	13	TO_DATE(' 2010-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
12	STUDENTS_LT_2005	29-OCT-23	2	1006	2	TO_DATE(' 2005-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
13	STUDENTS_MAX_VALUE	(null)	(null)	(null)	(null)	MAXVALUE

Now, we are performing an INSERT operation to transfer all data from an existing table named students to a new table called students_part. This operation essentially moves the records from one table to another.

```
INSERT INTO students_part
SELECT * FROM students;
```

Below provided SQL query analyzes student services by birth year. It extracts data on student birth years, categorizes services by birth year groups, and calculates the number of students using each service within those groups. The results are organized and presented for insights into service popularity and performance across different age groups, aiding in data-driven decision-making and resource optimization.

Query with partitioning:

```
EXPLAIN PLAN FOR
WITH student AS (
        SELECT  s.*,
                EXTRACT(YEAR FROM s.date_of_birth) AS student_birth_year
        FROM PURPLE.STUDENTS_PART s
```

```
        WHERE DATE_OF_BIRTH < TO_DATE('01-01-2009','DD-MM-YYYY')
),

birthyearwise_student_services AS (
        SELECT
                s.name AS service_name,
                student_birth_year,
                count(*) AS total_students
        FROM services s
                JOIN service_levels sl ON sl.service_id = s.service_id
                JOIN business_services bs ON bs.service_level_id = sl.service_level_id
                JOIN batches b ON b.business_services_id = bs.business_services_id
                JOIN student st ON st.batch_id = b.batch_id
        GROUP BY s.name, student_birth_year
        ORDER BY service_name, student_birth_year, total_students DESC
)

SELECT *
FROM birthyearwise_student_services ;

SELECT * FROM  TABLE(DBMS_XPLAN.DISPLAY);
```

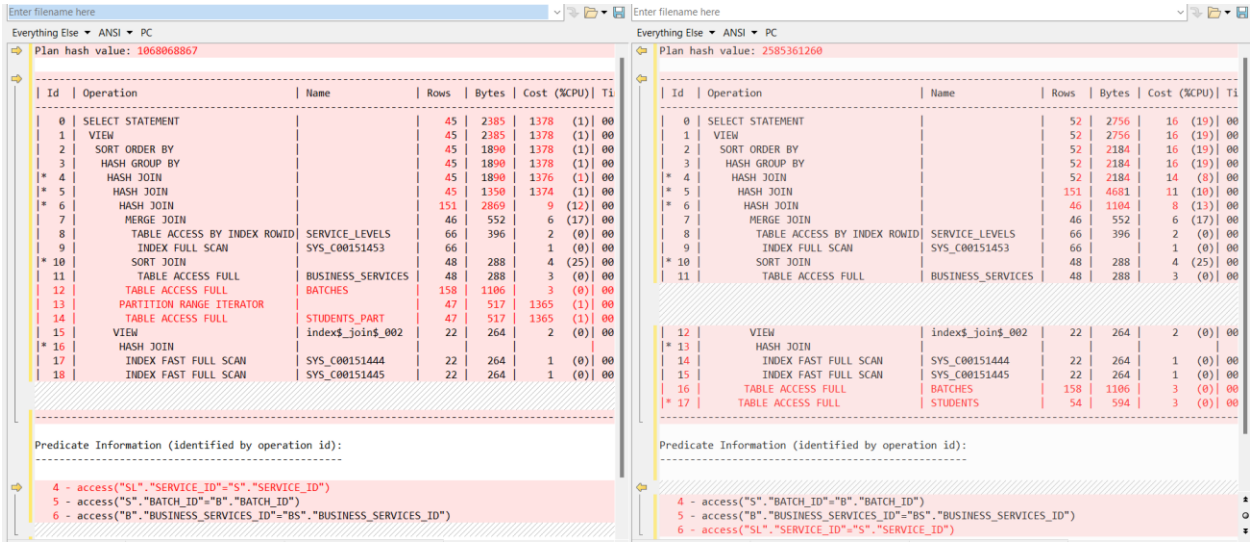
Query without partitioning:

```
EXPLAIN PLAN FOR
WITH student AS (
        SELECT
                s.*,
                EXTRACT(YEAR FROM s.date_of_birth) AS student_birth_year
        FROM PURPLE.STUDENTS s
        WHERE DATE_OF_BIRTH < TO_DATE('01-01-2009','DD-MM-YYYY')
),

birthyearwise_student_services AS (
        SELECT
                s.name AS service_name,
                student_birth_year,
                count(*) AS total_students
        FROM services s
                JOIN service_levels sl ON sl.service_id = s.service_id
                JOIN business_services bs ON bs.service_level_id = sl.service_level_id
                JOIN batches b ON b.business_services_id = bs.business_services_id
                JOIN student st ON st.batch_id = b.batch_id
        GROUP BY s.name, student_birth_year
        ORDER BY service_name, student_birth_year, total_students DESC
)
```

SELECT *
FROM birthyearwise_student_services ;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

Execution plan comparison:



```
Plan hash value: 1068068867

| Id | Operation                      | Name              | Rows | Bytes | Cost (%CPU)| Ti
|  0 | SELECT STATEMENT               |                   |   45 |  2385 |  1378   (1)| 00
|  1 |  VIEW                          |                   |   45 |  2385 |  1378   (1)| 00
|  2 |   SORT ORDER BY                |                   |   45 |  1890 |  1378   (1)| 00
|  3 |    HASH GROUP BY               |                   |   45 |  1890 |  1378   (1)| 00
|* 4 |     HASH JOIN                  |                   |   45 |  1890 |  1376   (1)| 00
|* 5 |      HASH JOIN                 |                   |   45 |  1350 |  1374   (1)| 00
|* 6 |       HASH JOIN                |                   |  151 |  2869 |     9  (12)| 00
|  7 |        MERGE JOIN              |                   |   46 |   552 |     6  (17)| 00
|  8 |         TABLE ACCESS BY INDEX ROWID| SERVICE_LEVELS |   66 |   396 |     2   (0)| 00
|  9 |          INDEX FULL SCAN       | SYS_C00151453     |   66 |       |     1   (0)| 00
|* 10 |        SORT JOIN               |                   |   48 |   288 |     4  (25)| 00
|  11 |         TABLE ACCESS FULL      | BUSINESS_SERVICES |   48 |   288 |     3   (0)| 00
|  12 |       TABLE ACCESS FULL        | BATCHES           |  158 |  1106 |     3   (0)| 00
|  13 |       PARTITION RANGE ITERATOR |                   |   47 |   517 |  1365   (1)| 00
|  14 |        TABLE ACCESS FULL       | STUDENTS_PART     |   47 |   517 |  1365   (1)| 00
|  15 |      VIEW                      | index$_join$_002  |   22 |   264 |     2   (0)| 00
|* 16 |       HASH JOIN                |                   |      |       |            |
|  17 |        INDEX FAST FULL SCAN    | SYS_C00151444     |   22 |   264 |     1   (0)| 00
|  18 |        INDEX FAST FULL SCAN    | SYS_C00151445     |   22 |   264 |     1   (0)| 00

Predicate Information (identified by operation id):

   4 - access("SL"."SERVICE_ID"="S"."SERVICE_ID")
   5 - access("S"."BATCH_ID"="B"."BATCH_ID")
   6 - access("B"."BUSINESS_SERVICES_ID"="BS"."BUSINESS_SERVICES_ID")
```

```
Plan hash value: 2585361260

| Id | Operation                      | Name              | Rows | Bytes | Cost (%CPU)| Ti
|  0 | SELECT STATEMENT               |                   |   52 |  2756 |    16  (19)| 00
|  1 |  VIEW                          |                   |   52 |  2756 |    16  (19)| 00
|  2 |   SORT ORDER BY                |                   |   52 |  2184 |    16  (19)| 00
|  3 |    HASH GROUP BY               |                   |   52 |  2184 |    16  (19)| 00
|* 4 |     HASH JOIN                  |                   |   52 |  2184 |    14   (8)| 00
|* 5 |      HASH JOIN                 |                   |  151 |  4681 |    11  (10)| 00
|* 6 |       HASH JOIN                |                   |   46 |  1104 |     8  (13)| 00
|  7 |        MERGE JOIN              |                   |   46 |   552 |     6  (17)| 00
|  8 |         TABLE ACCESS BY INDEX ROWID| SERVICE_LEVELS |   66 |   396 |     2   (0)| 00
|  9 |          INDEX FULL SCAN       | SYS_C00151453     |   66 |       |     1   (0)| 00
|* 10 |        SORT JOIN               |                   |   48 |   288 |     4  (25)| 00
|  11 |         TABLE ACCESS FULL      | BUSINESS_SERVICES |   48 |   288 |     3   (0)| 00
|  12 |       VIEW                     | index$_join$_002  |   22 |   264 |     2   (0)| 00
|* 13 |        HASH JOIN               |                   |      |       |            |
|  14 |         INDEX FAST FULL SCAN   | SYS_C00151444     |   22 |   264 |     1   (0)| 00
|  15 |         INDEX FAST FULL SCAN   | SYS_C00151445     |   22 |   264 |     1   (0)| 00
|  16 |      TABLE ACCESS FULL         | BATCHES           |  158 |  1106 |     3   (0)| 00
|* 17 |     TABLE ACCESS FULL          | STUDENTS          |   54 |   594 |     3   (0)| 00

Predicate Information (identified by operation id):

   4 - access("S"."BATCH_ID"="B"."BATCH_ID")
   5 - access("B"."BUSINESS_SERVICES_ID"="BS"."BUSINESS_SERVICES_ID")
   6 - access("SL"."SERVICE_ID"="S"."SERVICE_ID")
```

Execution Plan with Partitioning (STUDENTS_PART):

Partition Range Iteration (Operation 13):

Utilizes partitioning to select specific partitions (Pstart and Pstop) based on the date range criteria.
Optimizes the query to process a reduced data volume, contributing to better performance.

Hash Joins (Operations 4, 5, 6, 16):

Involves hash join operations for combining data from different tables.
Partitioning significantly reduces the data involved in these joins due to the selection of specific partitions, resulting in faster join operations.

Efficient Index Usage (Operations 17 and 18):

Utilizes index fast full scans on partitioned indexes for data access.
These index operations are particularly efficient as they access specific index partitions, enhancing overall performance.

Partition Pruning:

"**PARTITION PRUNING**" with actual Pstart and Pstop values eliminates unnecessary partitions during query execution. Reduces data processed by filtering out partitions that don't match the date range criteria, enhancing efficiency.

Execution Plan without Partitioning (STUDENTS):

Hash Joins (Operations 4, 5, 6, 13):

The plan involves multiple hash join operations for combining data from different tables.
Unlike the partitioned query, there is no partitioning to reduce the data involved in these joins.
This results in higher costs in terms of CPU and time.

Efficient Index Usage (Operations 8, 9, 14, 15):

The plan utilizes indexes, but they are not partitioned and do not benefit from partition pruning.
Accessing non-partitioned indexes may lead to slightly higher costs compared to partitioned indexes.

Table Access (Operations 16, 17):

The plan includes full table scans for both the BATCHES and STUDENTS tables.
Without partitioning, the database must scan the entire tables, resulting in higher costs in terms of time and CPU usage.

Cost-Based Optimization:

The optimizer still uses cost-based decisions for query execution, but without partitioning, it may not take full advantage of partition-specific optimizations.

Filtering (Operation 17):

The plan includes a filter condition for the STUDENTS table, which limits the results to students born before January 1, 2009.This condition is evaluated without partitioning, and the optimizer must scan the entire table.

Assess Query Performance:

The execution plan with partitioning benefits from partition pruning, which significantly reduces the data processed and enhances efficiency. It accesses specific partitions, which is particularly advantageous when dealing with large datasets. In contrast, the execution plan without partitioning requires full table scans and may involve higher CPU usage and longer execution times. Partitioning plays a crucial role in improving query performance, especially for queries that involve filtering and joining tables.

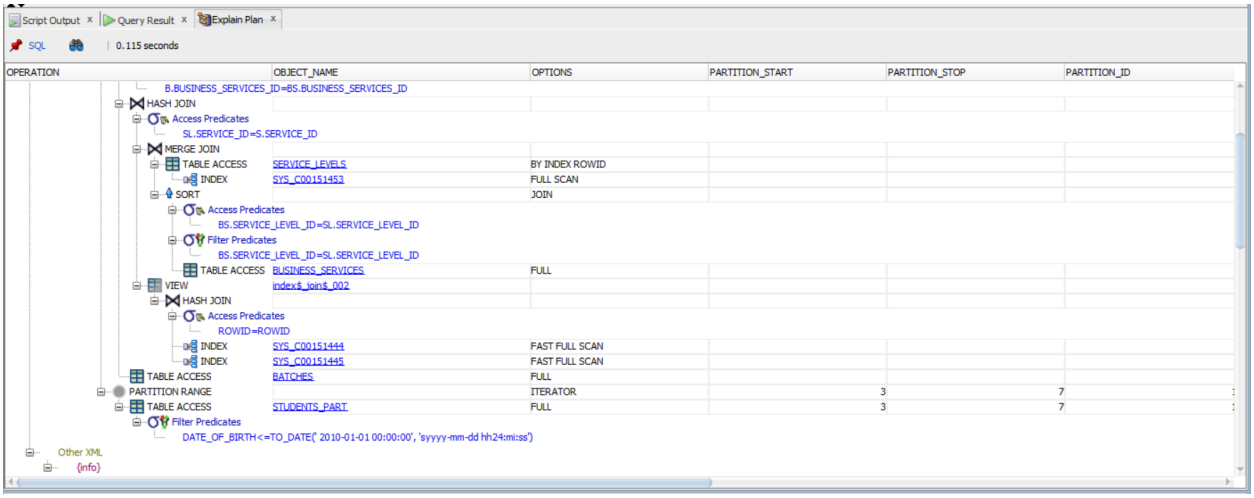
2. The below query is designed to showcase the effectiveness of partitioning by utilizing a date range query. It retrieves data from partitioned tables and demonstrates the benefits of optimized data access and improved query performance.

The query retrieves a list of services and the count of students associated with each service, grouped by the birth year of the students, for those students whose birth dates fall within the date range of January 1, 2006, to January 1, 2010. The results are sorted by service name, student birth year, and the total number of students in descending order.

Query with partitioned tables:

```
WITH student AS (
        SELECT
                s.*,
                EXTRACT(YEAR FROM s.date_of_birth) AS student_birth_year
        FROM PURPLE.STUDENTS_PART s
```

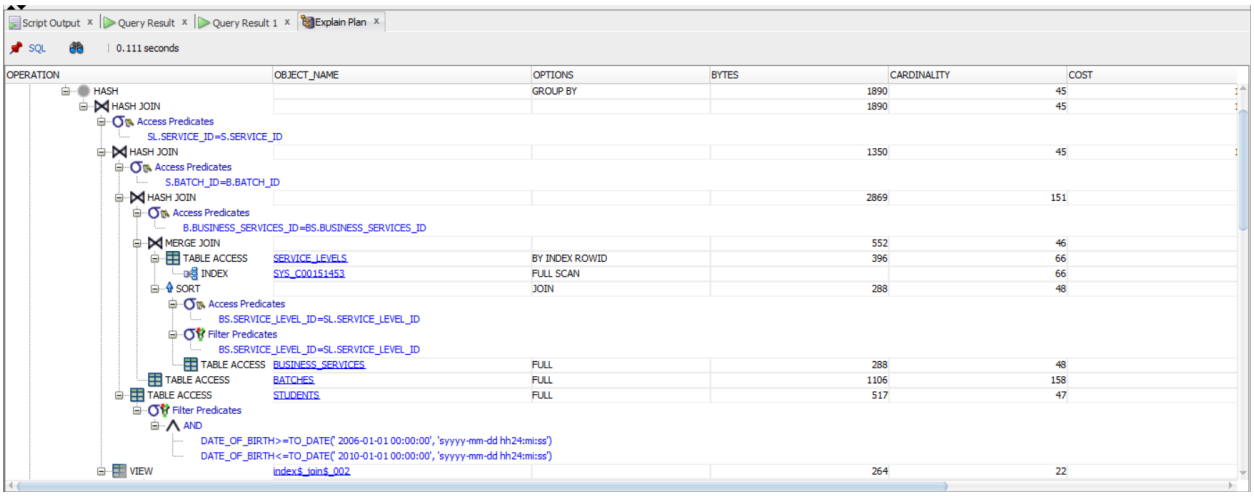
```
        WHERE DATE_OF_BIRTH BETWEEN TO_DATE('01-01-2006','DD-MM-YYYY') AND TO_DATE('01-01-
2010','DD-MM-YYYY')
),

birthyearwise_student_services AS (
        SELECT
                s.name AS service_name,
                student_birth_year,
                count(*) AS total_students
        FROM services s
                JOIN service_levels sl ON sl.service_id = s.service_id
                JOIN business_services bs ON bs.service_level_id = sl.service_level_id
                JOIN batches b ON b.business_services_id = bs.business_services_id
                JOIN student st ON st.batch_id = b.batch_id
        GROUP BY s.name, student_birth_year
        ORDER BY service_name, student_birth_year, total_students DESC
)

SELECT *
FROM birthyearwise_student_services ;
```



Query without partitioned tables:

```
WITH student AS (
        SELECT
                s.*,
                EXTRACT(YEAR FROM s.date_of_birth) AS student_birth_year
        FROM PURPLE.STUDENTS s
        WHERE DATE_OF_BIRTH BETWEEN TO_DATE('01-01-2006','DD-MM-YYYY') AND TO_DATE('01-01-
2010','DD-MM-YYYY')
),
```

```
birthyearwise_student_services AS (
        SELECT
                s.name AS service_name,
                student_birth_year,
                count(*) AS total_students
        FROM services s
                JOIN service_levels sl ON sl.service_id = s.service_id
                JOIN business_services bs ON bs.service_level_id = sl.service_level_id
                JOIN batches b ON b.business_services_id = bs.business_services_id
                JOIN student st ON st.batch_id = b.batch_id
        GROUP BY s.name, student_birth_year
        ORDER BY service_name, student_birth_year, total_students DESC
)

SELECT *
FROM birthyearwise_student_services ;
```



Assess Query performance:

When comparing the execution plans for the partitioned and non-partitioned versions of the query, it is evident that the partitioned version offers significant performance improvements.

Partitioned Query Execution Plan:

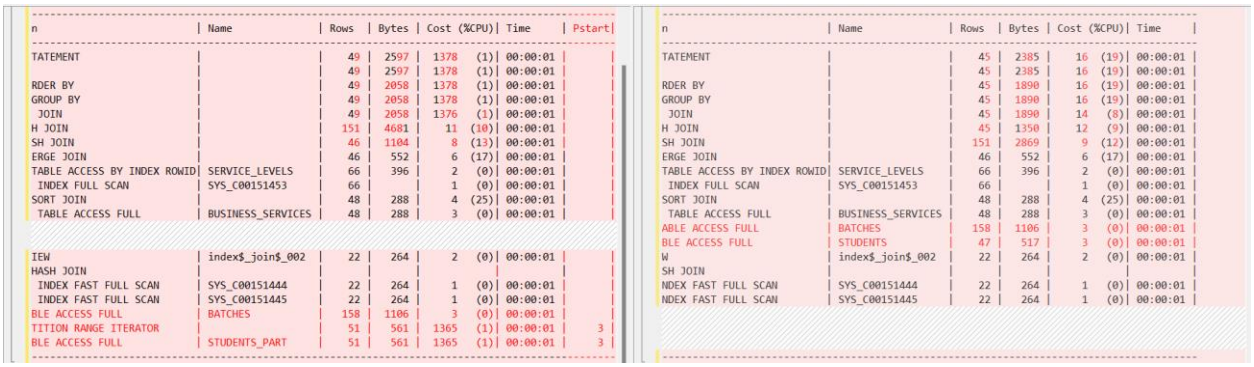
The query involves a **HASH JOIN** operation, which has a cost of 1376.HASH JOIN is used to combine results from multiple tables efficiently. A TABLE ACCESS FULL operation on the partitioned table "STUDENTS_PART" is executed with a cost of 1365.The partition range iterator accesses specific partitions (Pstart to Pstop) which helps in narrowing down the data, reducing the cost significantly. The filtering on "DATE_OF_BIRTH" is done more efficiently with a cost of 1.

Non-Partitioned Query Execution Plan:

The query also involves a HASH JOIN operation, but it has a higher cost of 14.

The TABLE ACCESS FULL operation on the non-partitioned table "STUDENTS" has a higher cost of 3.The filtering on "DATE_OF_BIRTH" in this case has a higher cost due to a broader scan range.The overall cost is higher compared to the partitioned version.

Comparison:



n	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart
TATEMENT		49	2597	1378 (1)	00:00:01	
		49	2597	1378 (1)	00:00:01	
RDER BY		49	2058	1378 (1)	00:00:01	
GROUP BY		49	2058	1378 (1)	00:00:01	
JOIN		49	2058	1376 (1)	00:00:01	
H JOIN		151	4681	11 (10)	00:00:01	
SH JOIN		46	1104	8 (13)	00:00:01	
ERGE JOIN		46	552	6 (17)	00:00:01	
TABLE ACCESS BY INDEX ROWID	SERVICE_LEVELS	66	396	2 (0)	00:00:01	
INDEX FULL SCAN	SYS_C00151453	66		1 (0)	00:00:01	
SORT JOIN		48	288	4 (25)	00:00:01	
TABLE ACCESS FULL	BUSINESS_SERVICES	48	288	3 (0)	00:00:01	
IEW	index$_join$_002	22	264	2 (0)	00:00:01	
HASH JOIN						
INDEX FAST FULL SCAN	SYS_C00151444	22	264	1 (0)	00:00:01	
INDEX FAST FULL SCAN	SYS_C00151445	22	264	1 (0)	00:00:01	
BLE ACCESS FULL	BATCHES	158	1106	3 (0)	00:00:01	
TITION RANGE ITERATOR		51	561	1365 (1)	00:00:01	3
BLE ACCESS FULL	STUDENTS_PART	51	561	1365 (1)	00:00:01	3

n	Name	Rows	Bytes	Cost (%CPU)	Time
TATEMENT		45	2385	16 (19)	00:00:01
		45	2385	16 (19)	00:00:01
RDER BY		45	1890	16 (19)	00:00:01
GROUP BY		45	1890	16 (19)	00:00:01
JOIN		45	1890	14 (8)	00:00:01
H JOIN		45	1350	12 (9)	00:00:01
SH JOIN		151	2869	9 (12)	00:00:01
ERGE JOIN		46	552	6 (17)	00:00:01
TABLE ACCESS BY INDEX ROWID	SERVICE_LEVELS	66	396	2 (0)	00:00:01
INDEX FULL SCAN	SYS_C00151453	66		1 (0)	00:00:01
SORT JOIN		48	288	4 (25)	00:00:01
TABLE ACCESS FULL	BUSINESS_SERVICES	48	288	3 (0)	00:00:01
ABLE ACCESS FULL	BATCHES	158	1106	3 (0)	00:00:01
BLE ACCESS FULL	STUDENTS	47	517	3 (0)	00:00:01
W	index$_join$_002	22	264	2 (0)	00:00:01
SH JOIN					
NDEX FAST FULL SCAN	SYS_C00151444	22	264	1 (0)	00:00:01
NDEX FAST FULL SCAN	SYS_C00151445	22	264	1 (0)	00:00:01

The partitioned version takes advantage of **partition pruning**, allowing it to access specific partitions that contain relevant data. This results in significantly reduced data access and lower costs. On the other hand, the non-partitioned version needs to perform a broader scan, leading to a higher cost.

In summary, the partitioned query's performance is much better, as it efficiently utilizes partitioning to minimize the amount of data accessed and processed, resulting in improved query execution speed and resource usage.

Others

DBA Scripts

We present a collection of Database Administrator (DBA) scripts designed to explore and report on the inner workings of an Oracle database. These scripts offer valuable insights into the database's structure, object relationships, storage, and more. Each script is accompanied by explanations of their utility, potential use cases, and illustrative sample results.

1.Script for Checking Constraints on a Table:

Below script retrieves information about constraints on a specific table, such as primary keys, unique constraints, and foreign keys. This can be used for documentation or ensuring data integrity by verifying the constraints on a table.

```
SELECT acc.table_name, acc.column_name, acc.position, cons.constraint_name, cons.constraint_type
FROM user_cons_columns acc
JOIN user_constraints cons
ON acc.constraint_name = cons.constraint_name
WHERE acc.table_name IN
('SERVICE_CATEGORY','SERVICES','BUSINESS','SERVICE_LEVELS','EMPLOYEE_TYPE','EMPLOYEE','BATCHES','CUSTOMER','STUDENTS','MEMBERSHIP_TYPE');
```

Output:

	TABLE_NAME	COLUMN_NAME	POSITION	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	MEMBERSHIP_TYPE	MEMBERSHIP_TYPE_ID	1	SYS_C00151480	P
2	STUDENTS	CUSTOMER_ID	(null)	SYS_C00151472	C
3	STUDENTS	BATCH_ID	(null)	SYS_C00151473	C
4	STUDENTS	STUDENT_ID	1	SYS_C00151474	P
5	STUDENTS	CUSTOMER_ID	1	FK_STUDENTS_CUSTOMER_ID	R
6	STUDENTS	BATCH_ID	1	FK_STUDENTS_BATCH_ID	R
7	STUDENTS	MEMBERSHIP_TYPE_ID	1	FK_STUDENT_MEM_TYPE_ID	R
8	CUSTOMER	CONTACT	(null)	SYS_C00151467	C
9	CUSTOMER	EMAIL_ID	(null)	SYS_C00151468	C
10	CUSTOMER	EMAIL_ID	(null)	CHK_CUSTOMER_EMAIL_ID	C
11	CUSTOMER	CUSTOMER_ID	1	SYS_C00151470	P
12	BATCHES	BATCH_ID	1	SYS_C00151464	P
13	BATCHES	BUSINESS_SERVICES_ID	1	FK_BATCHES_BUS_SERVICES_ID	R
14	BATCHES	INSTRUCTOR_ID	1	FK_BATCHES_INSTRUCTOR_ID	R
15	EMPLOYEE	EMAIL_ID	(null)	SYS_C00151459	C
16	EMPLOYEE	EMAIL_ID	(null)	CHK_EMPLOYEE_EMAIL_ID	C
17	EMPLOYEE	EMPLOYEE_ID	1	SYS_C00151461	P
18	EMPLOYEE	BUSINESS_ID	1	FK_EMPLOYEE_BUSINESS_ID	R
19	EMPLOYEE	EMPLOYEE_TYPE_ID	1	FK_EMPLOYEE_TYPE_ID	R
20	EMPLOYEE_TYPE	EMPLOYEE_TYPE_ID	1	SYS_C00151458	P
21	SERVICE_LEVELS	SERVICE_LEVEL_ID	1	SYS_C00151453	P
22	SERVICE_LEVELS	SERVICE_ID	1	FK_SERVICE_ID	R
23	BUSINESS	NAME	(null)	SYS_C00151447	C

2. Script for Monitoring Table Storage:

Below script provides the size of a table in megabytes and helpful for monitoring table growth and managing storage space.

```
SELECT table_name, blocks*8/1024 AS size_mb
FROM user_tables
WHERE table_name IN
('SERVICE_CATEGORY','SERVICES','BUSINESS','SERVICE_LEVELS','EMPLOYEE_TYPE','EMPLOYEE','BATCHES','CUSTOMER','STUDENTS','MEMBERSHIP_TYPE');
```

Output:

	TABLE_NAME	SIZE_MB
1	STUDENTS	0.0390625
2	SERVICE_LEVELS	0.0390625
3	SERVICE_CATEGORY	0.0390625
4	SERVICES	0.0390625
5	MEMBERSHIP_TYPE	0.0390625
6	EMPLOYEE_TYPE	0.0390625
7	EMPLOYEE	0.0390625
8	CUSTOMER	0.0390625
9	BUSINESS	0.0390625
10	BATCHES	0.0390625

3. **Script for Listing All Tables in a Schema:**

This script is useful for quickly retrieving a list of all tables within a schema.It can be used for documentation, auditing, or simply to get an overview of the database's structure.

```
SELECT table_name
FROM user_tables;
```

Output:

	TABLE_NAME
1	TEMP_STUDENTS
2	TEMP_BATCHES
3	STUDENTS
4	SERVICE_LEVELS
5	SERVICE_CATEGORY
6	SERVICES
7	MEMBERSHIP_TYPE
8	EMPLOYEE_TYPE
9	EMPLOYEE
10	CUSTOMER
11	BUSINESS_SERVICES
12	BUSINESS
13	BATCHES
14	STUDENTS_PART

4. Script to display SQL statements for the current database sessions.

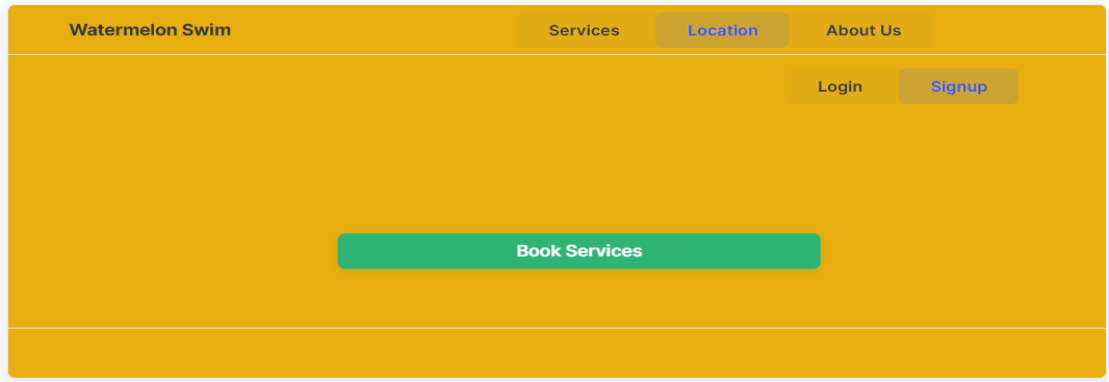
```
SELECT vs.sid,vs.status,vs.process,vs.schemaname,vs.osuser,va.sql_text
FROM v$session vs, v$sqlarea va
WHERE vs.sql_hash_value = va.hash_value
AND vs.sql_address = va.address;
```

Output:

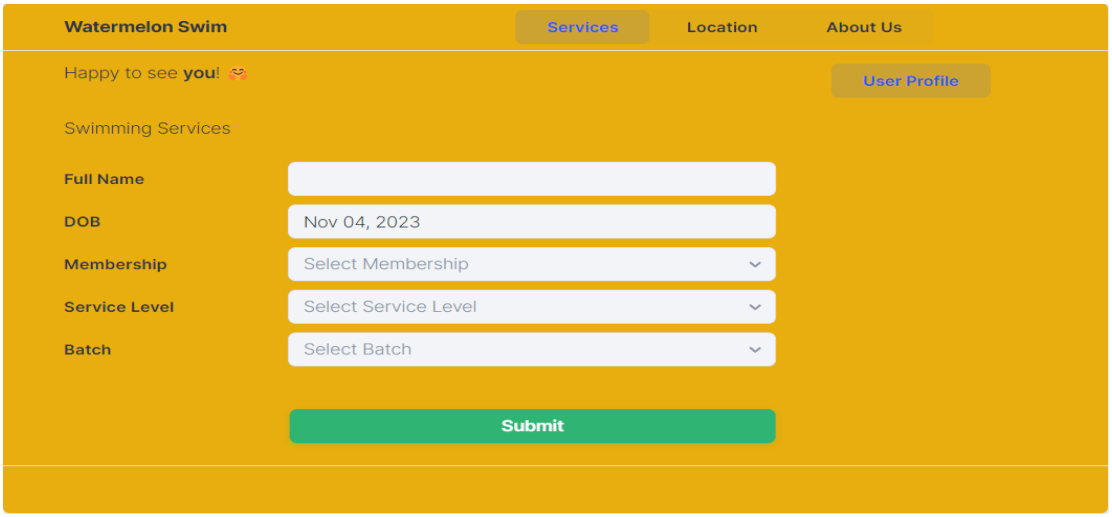
	SID	STATUS	PROCESS	SCHEMANAME	OSUSER	SQL_TEXT
1	44	ACTIVE	2660	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
2	51	ACTIVE	3424	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
3	55	ACTIVE	2672	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
4	59	ACTIVE	2652	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
5	64	ACTIVE	2668	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
6	65	ACTIVE	29964	PURPLE	Acer	SELECT vs.sid, vs.status, vs.process, vs.schemaname, vs.osuser, va.sql_text FROM v$session vs, v$sqlarea va WHERE vs.sql_hash_
7	70	ACTIVE	2952	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
8	78	ACTIVE	3448	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
9	82	INACTIVE	30912	DW328	Bhargav Lakkireddy	select * from blue.soundtrack
10	83	ACTIVE	2880	DW328	Bhargav Lakkireddy	select * from blue.soundtrack

User Interface Mockups

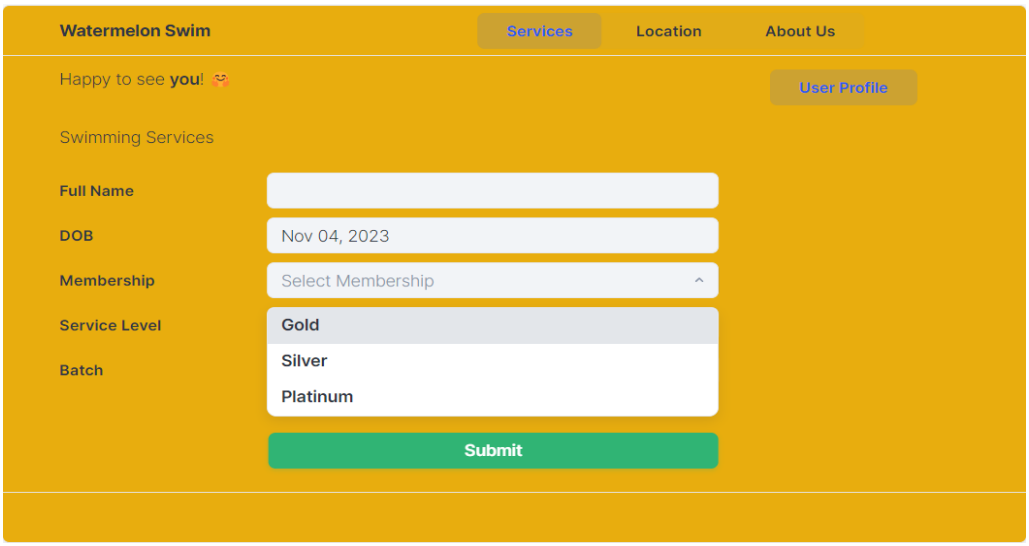
This is the Watermelon Swim website Booking Page for Streamlined Service Reservation.



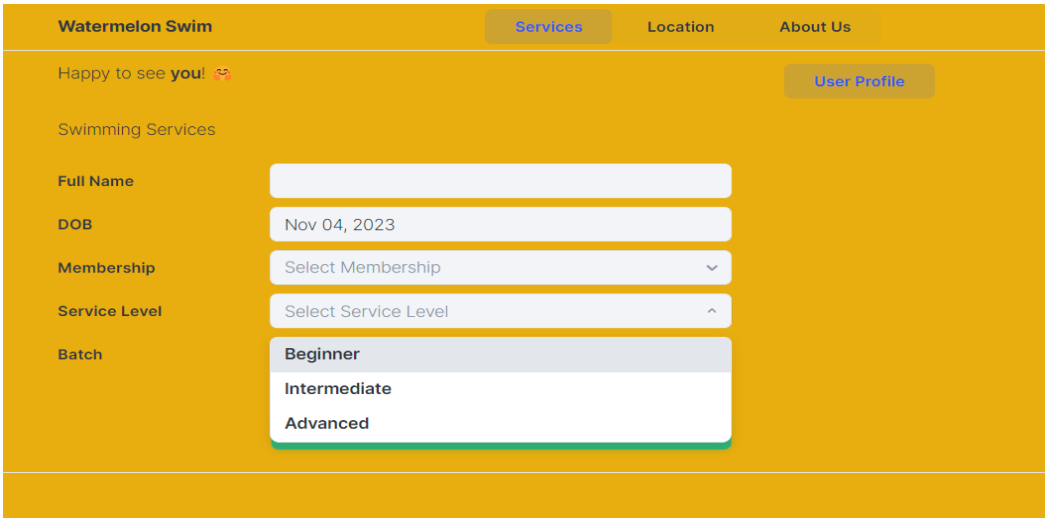
Watermelon Swim offers an intuitive service reservation page on its website, guiding customers through a quick booking process. Users select services, fill in personal details (full name, DOB, membership type), and choose a service level from a dropdown. The system dynamically presents relevant batch options for scheduling. This ensures tailored swimming lessons and a personalized experience for each customer, optimizing resource allocation and customer satisfaction.



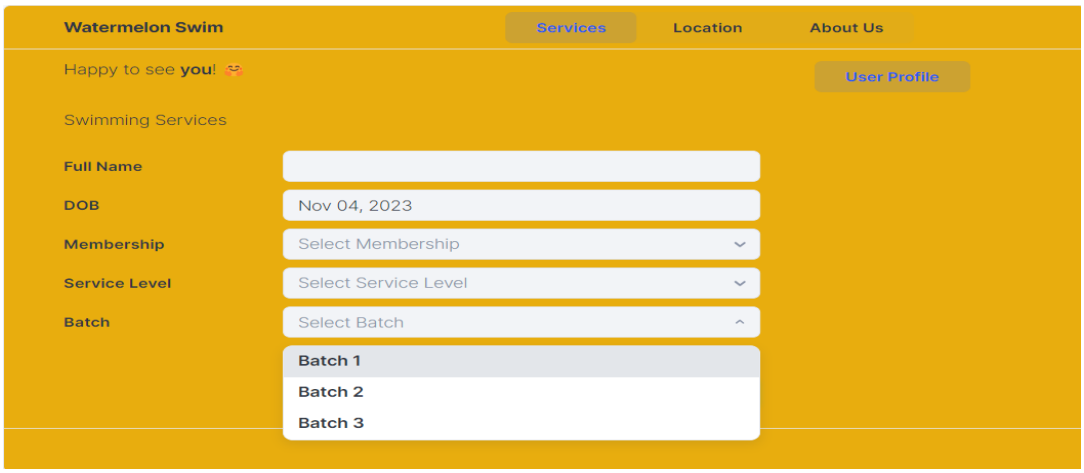
Customers have the option to choose memberships.



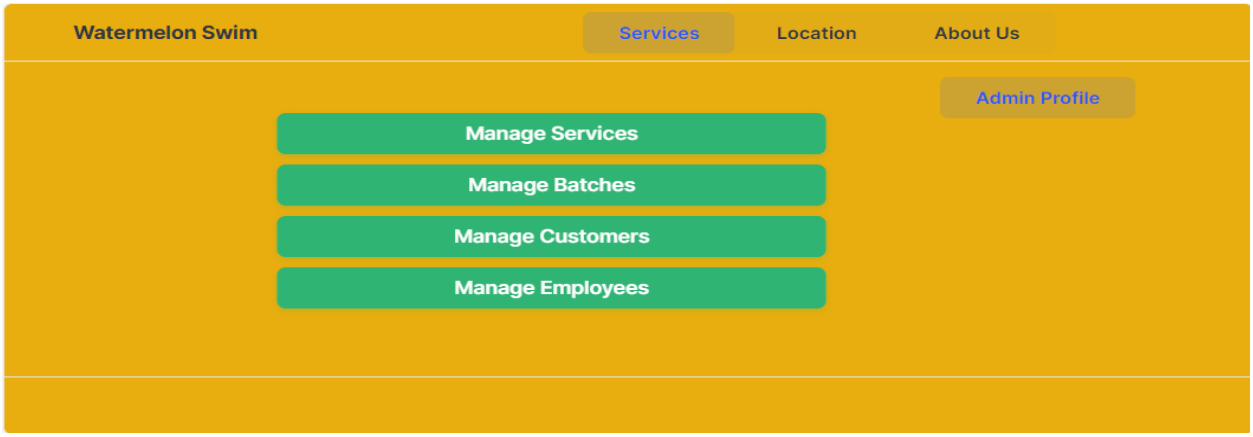
Customers have the option to choose from a variety of service levels.



Customers have the option to choose from a variety of Batches as per availability.



The admin page serves as the centralized management hub for Watermelon Swim, providing administrators with robust tools to oversee and control essential aspects of the business. Administrators can efficiently manage services, customers, employees, and batches, ensuring a smooth and well-coordinated operation.



Key Functions:

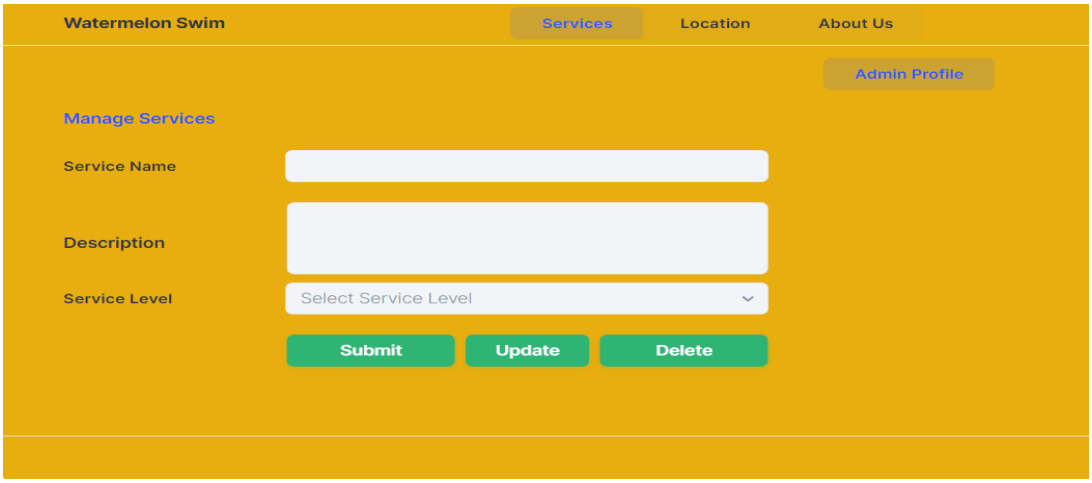
Service Management: Admins have the authority to add, update, or remove various service offerings, ensuring that the business stays aligned with customer needs and preferences.

Customer Administration: The admin page enables administrators to maintain customer records, add new customers, update existing profiles, and, if necessary, delete customer information.

Employee Oversight: Administrators can manage employee details, oversee their records, make updates, and, if needed, remove employee data.

Batch Control: The system facilitates the creation, modification, or deletion of batches for swim classes, ensuring that scheduling and resource allocation are effectively managed.

Batches Management: Admins can oversee batches, adjusting as required to ensure smooth operation and accommodating changing needs.



Data Visualization

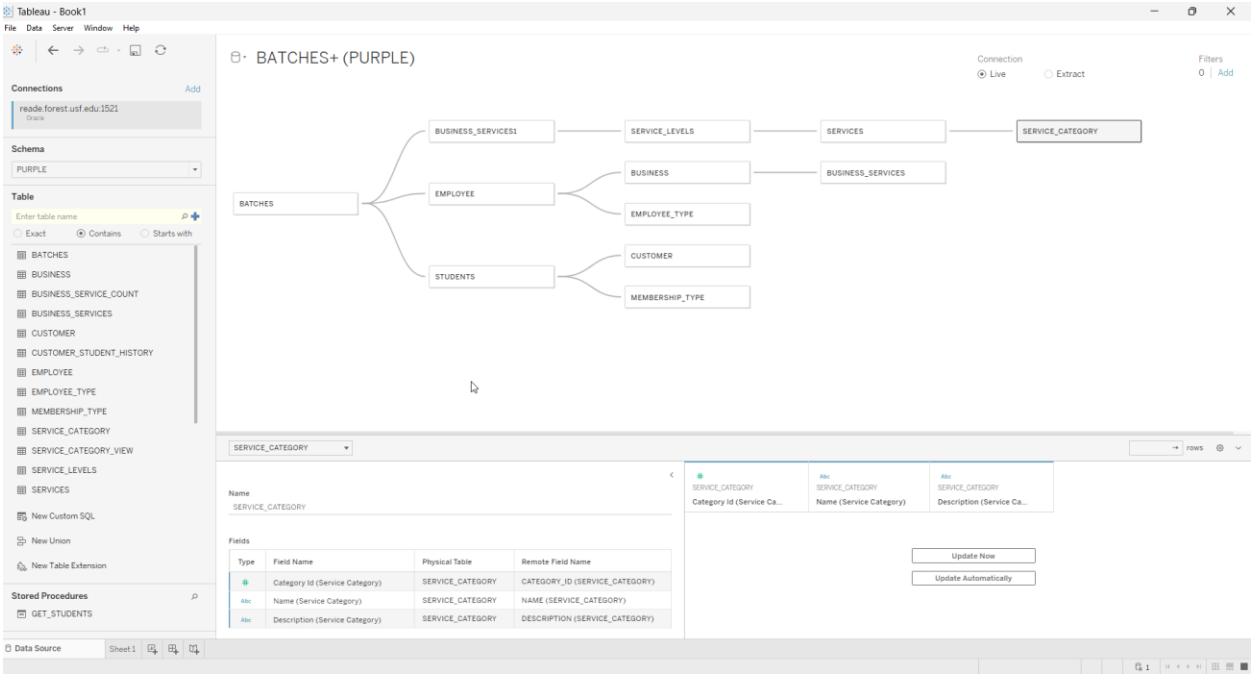
Data source for visualization

In our Database project, I designed a visualization that revolves around our single data source and corresponding datasets. Our goal was to create a unified and streamlined approach to visualization by consolidating all data into a single source. This made it easier to get insights and business metrics.

The datasets were derived from the tables provided and served as the foundation for this visualization. They comprised a wide range of information such as customer details, membership types, student records, employee data, and much more. By consolidating these datasets, we obtained a comprehensive view of our project, allowing us to perform various calculations and analyses.

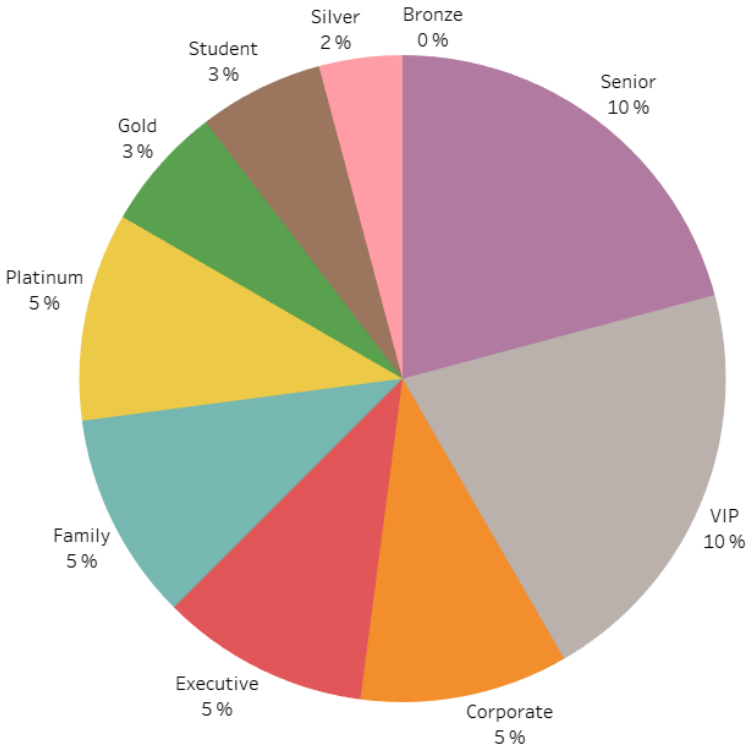
Using Tableau, we crafted a detailed and user-friendly dashboard that extracted meaningful insights and metrics. Each visualization was customized to highlight relevant data, such as mapping student distribution by city, calculating average ages, or visualizing membership-type discounts. Filters and parameters were thoughtfully integrated to offer users flexibility in interacting with the data based on their analytical needs.

Our approach to data source consolidation and visualization design streamlined the process and optimized the clarity and depth of insights. It allowed us to examine our project's data comprehensively, helping us to make informed decisions and gain a deeper understanding of our database.

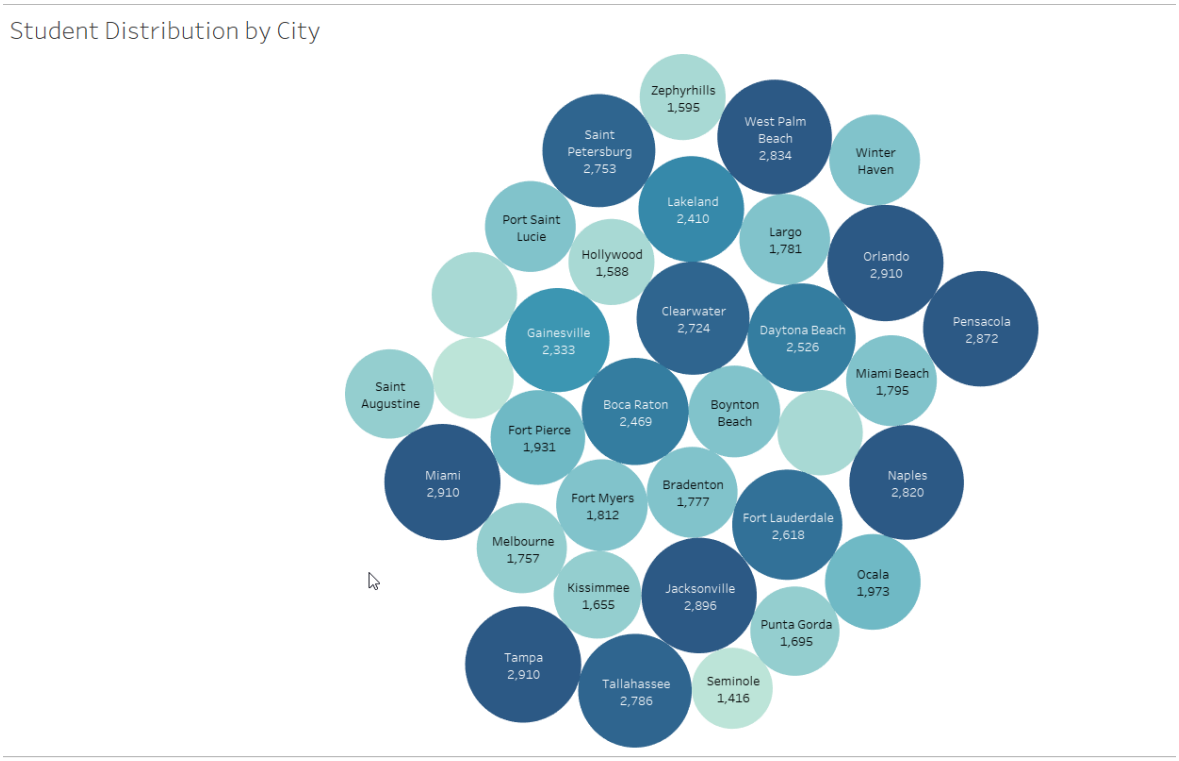


We designed a visualization to calculate the average discount per membership type. I created a donut chart, with "Membership Name" as the dimension and "Average Discount" as the measure. To enhance user interactivity, I added a parameter allowing users to control the threshold for displaying membership types based on the average discount. This visualization offers a clear and customizable view of membership discounts.

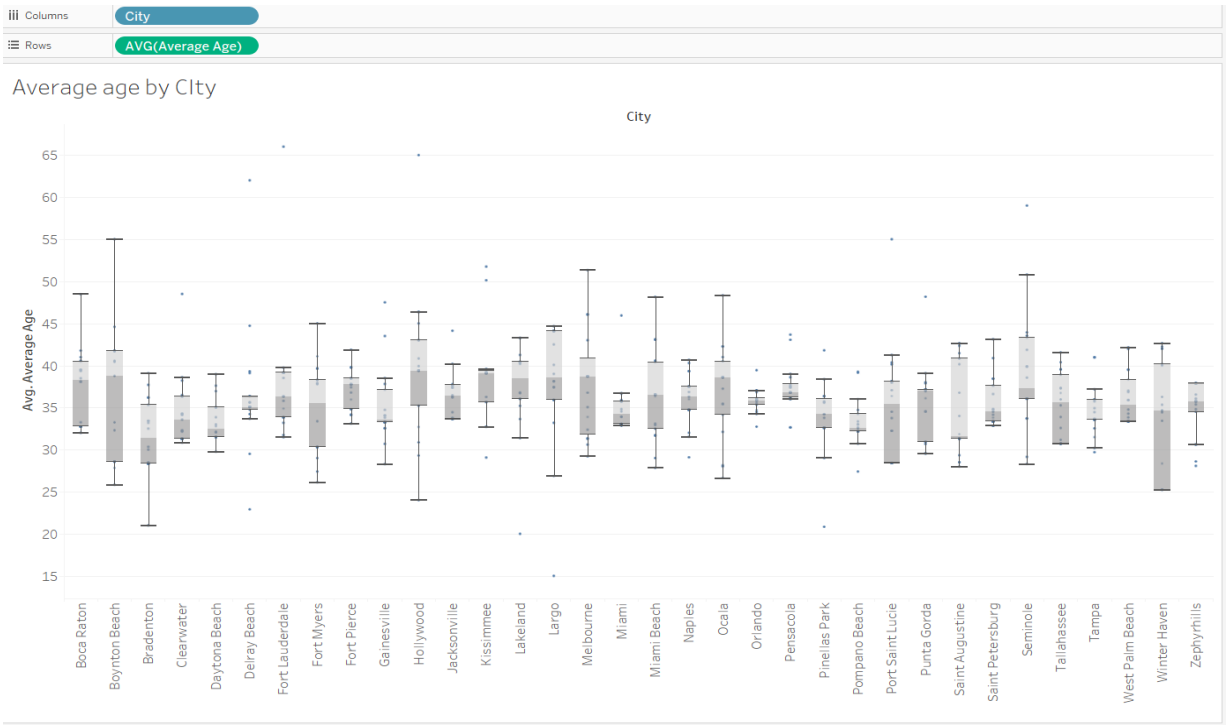
Membership Type Distribution with Avg Discounts



We designed a map visualization to display student distribution by city. I utilized the "City" as a geographic dimension and represented data using bubble chart markers sized according to the "Number of Students." To enhance interactivity, I integrated filters for date ranges and other relevant dimensions. Additionally, I incorporated a parameter to control the scaling of marker sizes, ensuring an informative and visually effective representation of the data.



We designed a visualization to calculate the average age of students in different cities. To achieve this, we created a calculated field, which computed the age of students based on their birthdates. We added filters for batch selection to provide flexibility in exploring the data. Additionally, we incorporated parameters to control the color scale and age thresholds, allowing for user customization. We used box and whisker plots to represent data distribution effectively, enhancing the visualization's interpretability and analytical depth.

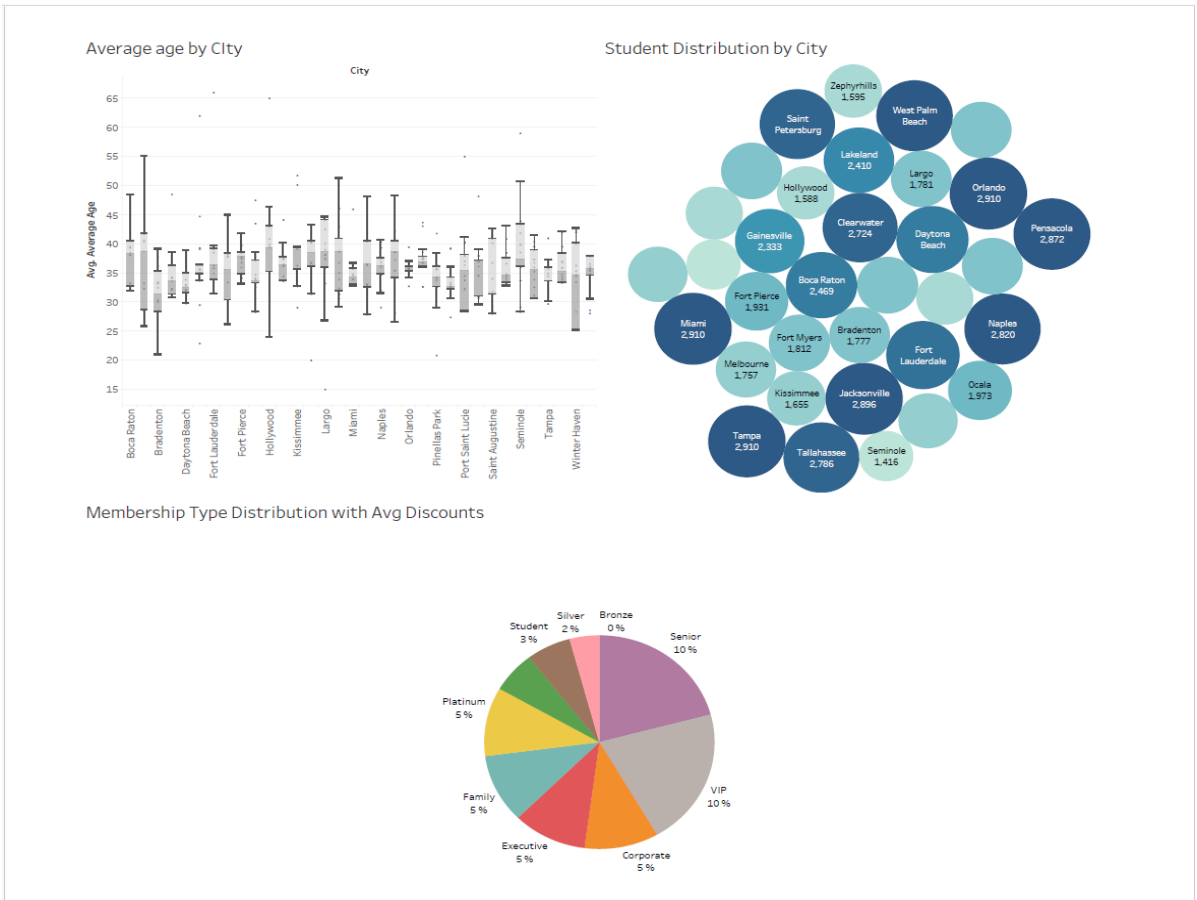


Dashboard

For our database project, I designed a visualization dashboard that combines three key elements to provide an informative view of our data. Firstly, we created a map visualization that highlights student distribution in different cities. Using a bubble chart marker technique, we sized markers according to the "Number of Students" and used the "City" dimension to achieve this. We also included date range filters and parameter controls to adjust marker scaling dynamically, making it an insightful representation of the data.

The second component of our dashboard focused on calculating the average age of students in various cities. We introduced a calculated field that computed student ages based on their birthdates. Batch selection filters were added to enable data exploration flexibility. We employed parameters to offer users control over the color scale and age thresholds, resulting in a versatile and interactive experience. We used box and whisker plots to visually represent data distribution, thereby enhancing the visualization's interpretability and analytical depth.

The final component of our dashboard involved calculating the average discount per membership type. We chose a donut chart format, leveraging "Membership Name" as the dimension and "Average Discount" as the measure. To boost user interactivity, we introduced a parameter that empowers users to adjust the threshold for displaying membership types based on average discount. This visualization offers a clear and customizable view of membership discounts, contributing to a well-rounded and insightful dashboard.



DB Security

In the Business Management System, the access privileges and security management are done by the system administrators. To access the website and the backend database, users must be authenticated.

The current scope of the system delineates two user categories:

Admin (representing the business owner): Admin users hold overarching responsibilities, managing all business operations, and making strategic decisions. This user will have admin access and control of the entire system. This user can create, edit, update, delete records on the website.

Employee: Distinct employee types, including Managers, Receptionists, House-keeping personnel, and Instructors. Managers supervise daily operations, ensuring efficiency and effective team management. Receptionists focus on customer service, welcoming and assisting visitors and managing inquiries. House-keeping staff maintain premises' cleanliness and hygiene. Instructors, specialized in various skills, conduct training sessions, and share expertise.

This is the future scope of our project where user hierarchy establishes role-specific access levels and privileges, optimizing the database's organizational structure and promoting efficient operations while reflecting the diverse roles within the business.

There are different access roles given to different types of users, as shown below:

Roles	Access and Privileges
Administrator-Owner	This user will have admin access and control of the entire system. This user can create, edit, update, delete records on the website.
Employee	This is the default access given to every user in clinic using this system. They will have role-specific access levels to various functionalities to operate on. Distinct employee types, including Managers, Receptionists, House-keeping personnel, and Instructors.

Archival Process

Efficient archival processes ensure the systematic capture and storage of historical data within the Business Management System, fostering compliance, analysis, and transparency.

The "**trg_customer_student_history**" trigger plays a critical role in the archival process within the Business Management System (BMS). Its primary function is to capture and archive historical data related to customer-student interactions. Here's an explanation of how it contributes to the archival process.

The trigger systematically logs historical data into the "customer_student_history" table, mirroring customer actions on student records. Each entry in the table includes crucial information, such as the date and time of the action, the action type (update or delete), the customer's details, and the corresponding student data.

	AUDIT_ID	ACTION_DATE	ACTION_TYPE	CUSTOMER_ID	CUSTOMER_FIRST_NAME	CUSTOMER_LAST_NAME	CUSTOMER_STREET_ADDRESS	CUSTOMER_COUNTY	CUSTOMER_CITY	CUSTOMER_STATE
1	1	05-NOV-23 11.48.19.487000000 AM	UPDATE	62	John	Doe	123 Main St	County	City	State
2	2	05-NOV-23 11.48.24.089000000 AM	UPDATE	62	John	Doe	123 Main St	County	City	State
3	3	05-NOV-23 11.49.29.094000000 AM	DELETE	62	John	Doe	123 Main St	County	City	State
4	4	05-NOV-23 11.49.32.152000000 AM	DELETE	62	John	Doe	123 Main St	County	City	State

Future Scope

1.**Scheduling and Appointment Management:**

Implement an advanced scheduling module that allows businesses to manage appointments and bookings efficiently. Features should include automated reminders, real-time availability, and the ability for customers to self-schedule.

2.**Online Payment System:**

Integrate a secure online payment gateway to facilitate online transactions. This can support various payment methods, including credit cards, digital wallets, and other online payment options, ensuring a seamless and secure payment experience.

3.**Inventory Management:**

Extend the system to include inventory management functionality for businesses that handle products. This feature will enable businesses to track stock levels, manage replenishment, and synchronize inventory data with sales and services.

4. **Role-specific access levels and privileges:** Expanding role-specific access levels and privileges will enhance security and streamline operations, enabling more fine-grained control over user actions within the system. This will provide an added layer of data protection and ensure that each user's permissions align precisely with their responsibilities.

References

https://uk.indeed.com/career-advice/career-development/business-management-system

https://www.capterra.com/business-management-software/

https://www.glofox.com/glofox-gym-management-software-revenue

https://www.w3schools.com/sql/

https://www.codecademy.com/article/sql-commands

https://data.world/

https://uibakery.io/

Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel

https://www.iclasspro.com/iclasspro-features