MET CS 669 Database Design and Implementation for Business
Health Term Project: Multi-Specialty Ambulatory Health Center

Melinda Speckmann
08/06/17

Table of Contents

Introduction & Proposal

The purpose of this project is to design an initial database schema for BUAHC which is a Multi-Specialty Ambulatory Health Center. This paper will deliver a proposed design that will display a conceptual and logical entity relationship diagram (ERD), explain the business rules, and then walk through the creation of the database and various required use cases that the design was required to meet.

At a high-level, this design **must** cover the following four use cases:

- Physician Schedules
- Patient Appointments/Visits with their Physicians
- Physician Waitlists for Patients
- Patient Insurance Plans

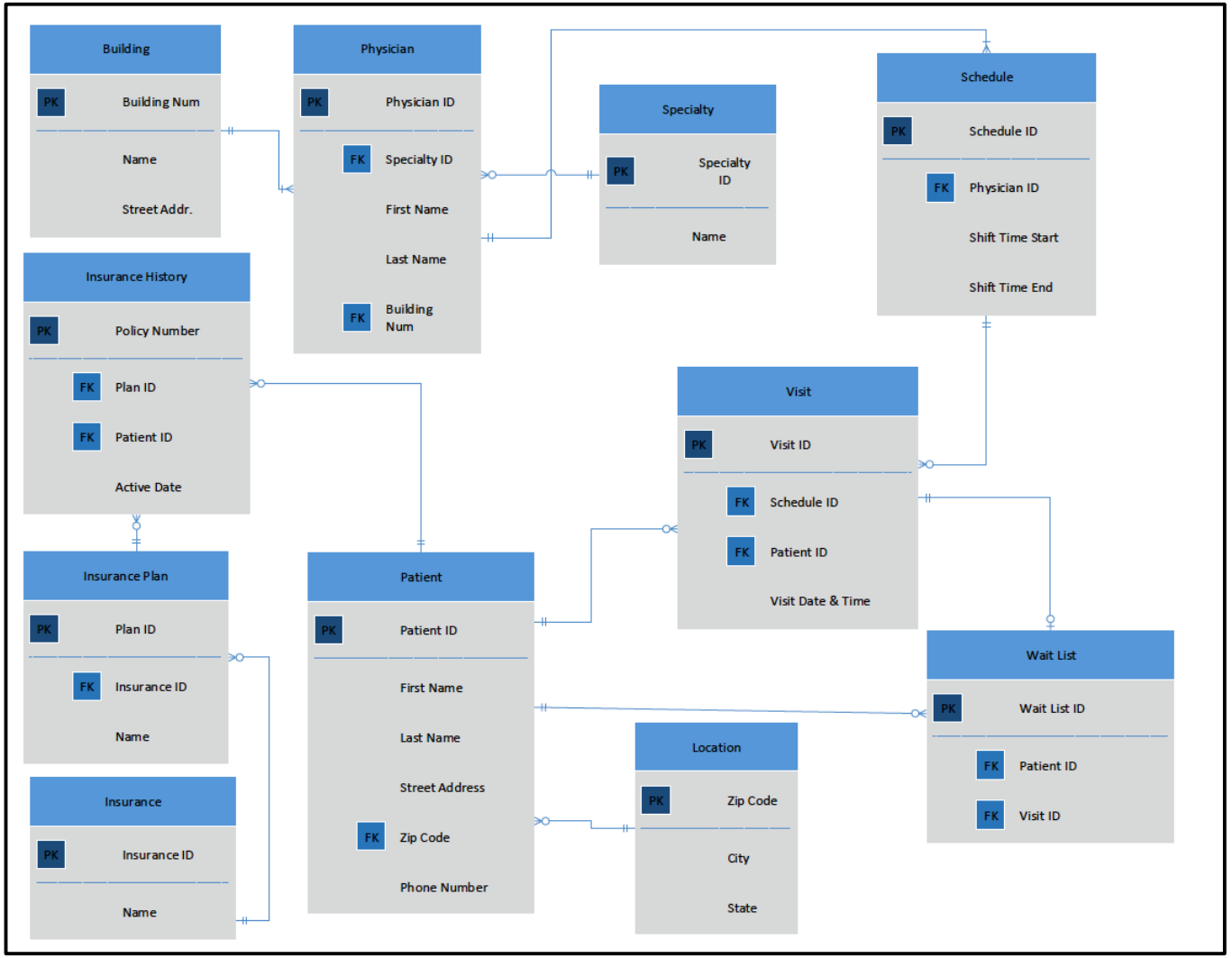
As an addition for this proposal, the design also includes the following two **additional** use cases:

- Waitlist checker when a Physician Schedule Entry is added to the Schedule
- Appointment to Specialty reporting to help make budget decisions when a new Physician can be hired.

Please find the following attached with this proposal:

- Create Table Scripts
- Insert Table Scripts for test data
- Use Case queries
- Index query
- PDF of Conceptual ERD
- PDF of Logical ERD

Conceptual Entity Relationship Diagram



Business Rules

Entities: Building, Physician, Specialty, Patient, Insurance Plan, Insurance, Insurance History, Visit, Waiting List, Schedule, Location

Each **visit** is with one **physician**. (singular, mandatory)
Each **physician** will have zero to many **visits**. (plural, optional)

A **physician** can only see two patients per hour.

Each **physician** works in one **building**. (singular, mandatory)
A **building** has one to many **physicians** working there. (plural, mandatory)

Each **physician** has zero to many **waiting listings**. (plural, optional)
Each **waiting listing** is for one **physician**. (singular, mandatory)

Each **patient** can be on zero to many **wait listings**. (plural, optional)
Each **wait listing** can only have one **patient**. (singular, mandatory)

Each **patient** can be scheduled for zero to many **visits**. (plural, optional)
Each **visit** only has one **patient**. (singular, mandatory)

Each **patient** has zero to many insurance histories. (plural, optional)
Each **insurance history** has one **patient**. (singular, mandatory)

Each **patient** has one **zip code**. (singular, mandatory)
A **zip code** can belong to zero or more **patients**. (plural, optional)

Each **insurance** will have zero to many **insurance plans**. (plural, optional)
Each **insurance plan** has one type of **insurance**. (singular, mandatory)

Each **Insurance plan** will have zero to many **insurance histories**. (plural, optional)
Each **insurance history** will have one **insurance plan**. (singular, mandatory)

Each **physician** will have one to many **schedules**. (plural, mandatory)
Each **schedule** entry will be for one **physician**. (singular, mandatory)

Each **physician** will have a **specialty**. (singular, mandatory)
Each **specialty** will belong to zero or more **physicians**. (plural, optional)

Each **schedule** entry will be in zero or more **visits**. (plural, optional)
Each **visit** will be for one **schedule entry**. (singular, mandatory)

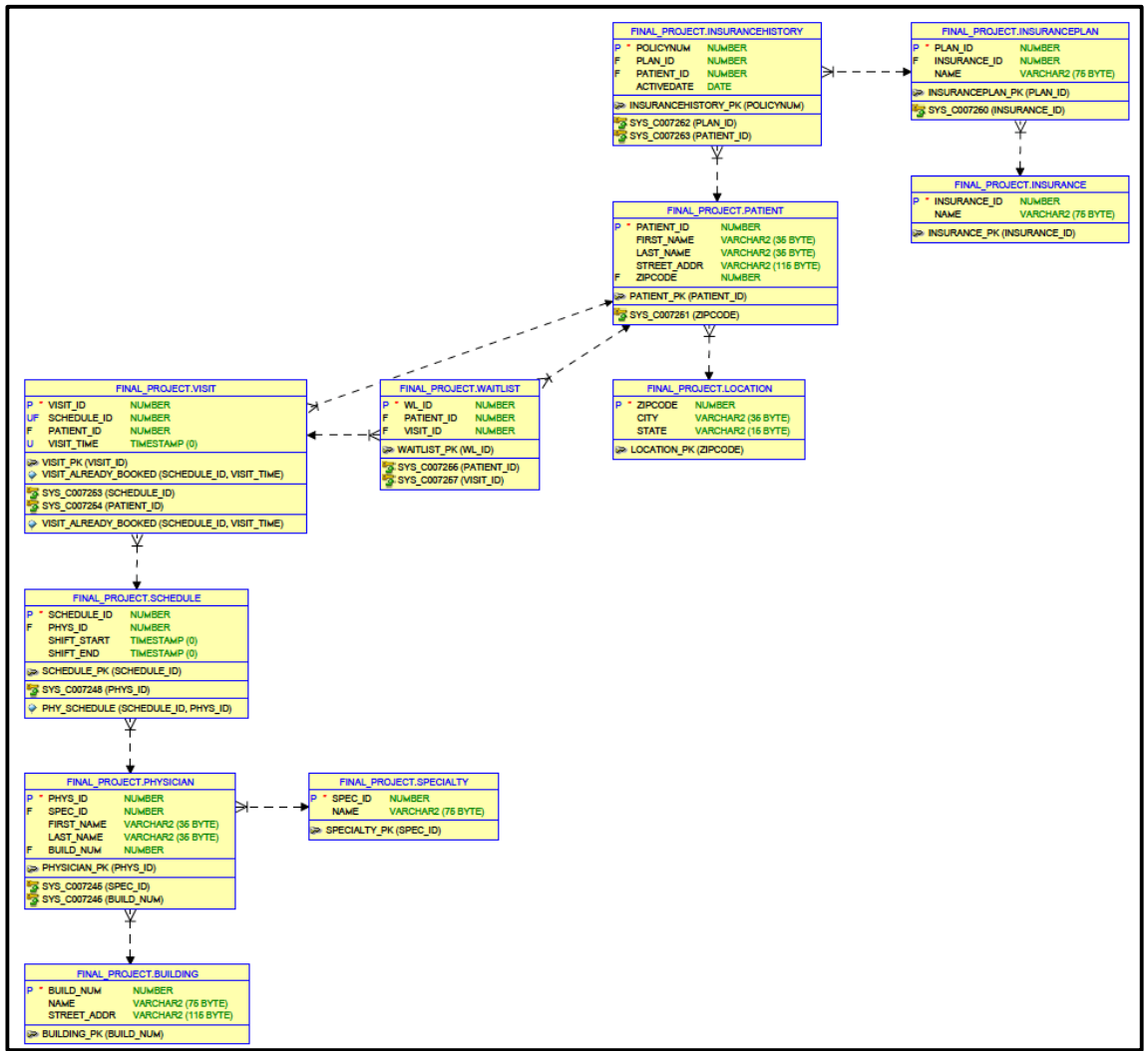
Each **wait list entry** lists one **visit**. (singular, mandatory)
A **visit** may or may not be listed in a **wait list entry**. (singular, optional)

When a **Visit** is cancelled, the **Wait List** must be checked to see if a patient can schedule a visit sooner with the doctor. Thus the wait list entry should be deleted, their backup visit cancelled and patient moved to the original cancelled visit if they are available.

In order to reduce redundancy, the Physician entity only has a relationship with the Specialty, Building and Schedule entities. This means that in order to look up the attending Physician for a Visit, Waitlist or Patient instance, a join must be done with the Schedule entity that will act as a middle man.

Logical Entity Relationship Diagram

***Please note that a clearer PDF copy is attached with this report.*



Use Cases: Screenshots and Explanations

Use Case 1: Appointments

a. A patient books an appointment with physician “Kathlin Jones” and a different appointment with physician “Scottie Marr”. Develop a parameterized stored procedure that supports a patient booking an appointment, then invoke the stored procedure two times (to satisfy the use case) for a patient of your choosing.

The first two queries needed before the ADD_APPOINTMENT procedure are the following:

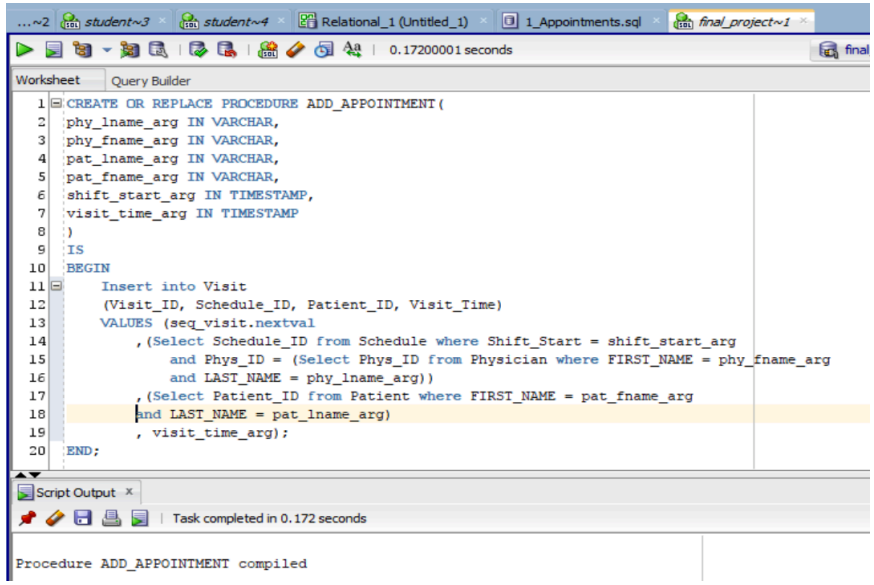
1. A “Create Sequence” query to automate the generation of the Visit_ID Primary Key from the Visit table. This is added because it does not make sense to have a staff member manually add an uninformative primary key when generating a new appointment.
2. A query to check the schedule of Kathlin Jones. This will show the Physician’s name and their schedule.

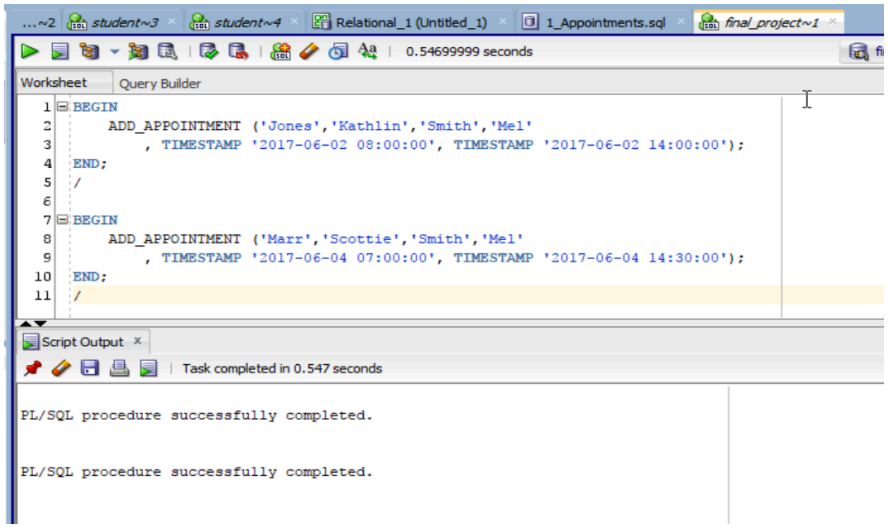
Both of the above queries can be referenced in the 1_Appointment.sql attachment in the zip along with this report.

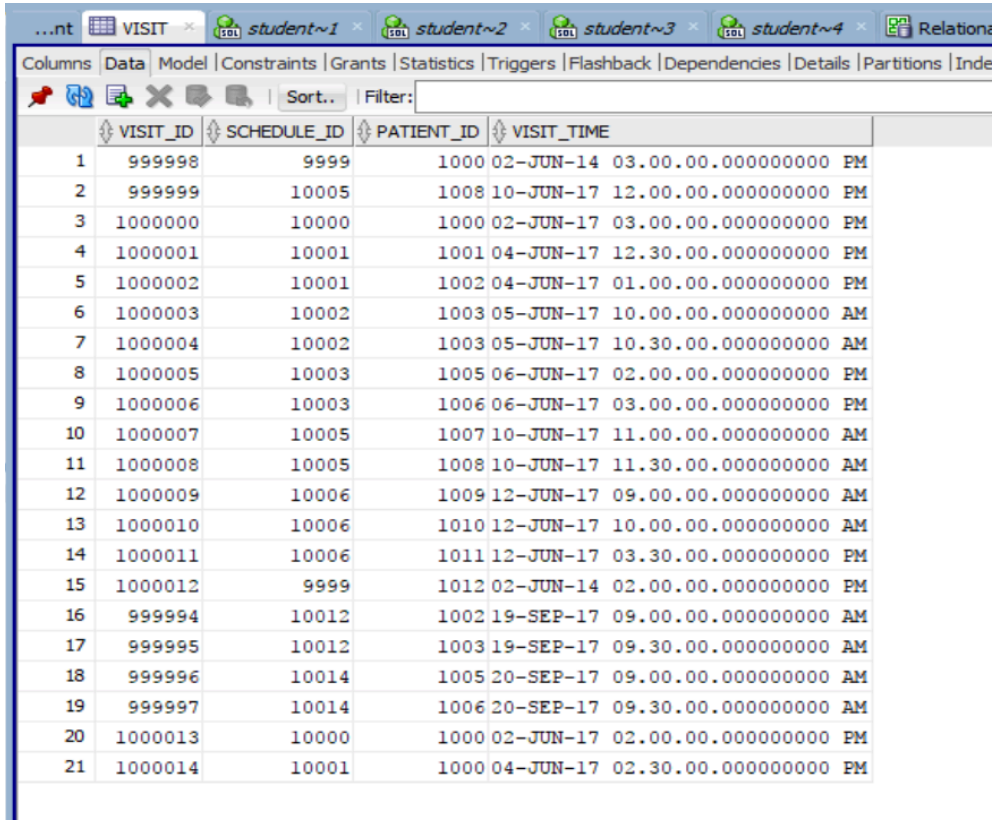
Logic of stored procedure:
The below procedure requires that the following be entered to book an appointment:

- Name of the Physician
- Name of the Patient
- Shift_Start from the Physician Schedule. (This is to specify which schedule entry we are using from the Physician).
- Visit_Time which is the date and time of the visit that the patient would like to book.

Please see below for a screenshot of the procedure generation and the booking of two appointments for Mel Smith with two different Physicians, one being Kathlin Jones.


...~2
student~3
student~4
Relational_1 (Untitled_1)
1_Appointments.sql
final_project~1
0.17200001 seconds
Worksheet
Query Builder
CREATE OR REPLACE PROCEDURE ADD_APPOINTMENT(
phy_lname_arg IN VARCHAR,
phy_fname_arg IN VARCHAR,
pat_lname_arg IN VARCHAR,
pat_fname_arg IN VARCHAR,
shift_start_arg IN TIMESTAMP,
visit_time_arg IN TIMESTAMP
)
IS
BEGIN
Insert into Visit
(Visit_ID, Schedule_ID, Patient_ID, Visit_Time)
VALUES (seq_visit.nextval
,(Select Schedule_ID from Schedule where Shift_Start = shift_start_arg
and Phys_ID = (Select Phys_ID from Physician where FIRST_NAME = phy_fname_arg
and LAST_NAME = phy_lname_arg))
,(Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
and LAST_NAME = pat_lname_arg)
, visit_time_arg);
END;
Script Output
Task completed in 0.172 seconds
Procedure ADD_APPOINTMENT compiled


...~2
student~3
student~4
Relational_1 (Untitled_1)
1_Appointments.sql
final_project~1
0.54699999 seconds
Worksheet
Query Builder
BEGIN
ADD_APPOINTMENT ('Jones','Kathlin','Smith','Mel'
, TIMESTAMP '2017-06-02 08:00:00', TIMESTAMP '2017-06-02 14:00:00');
END;
/
BEGIN
ADD_APPOINTMENT ('Marr','Scottie','Smith','Mel'
, TIMESTAMP '2017-06-04 07:00:00', TIMESTAMP '2017-06-04 14:30:00');
END;
/
Script Output
Task completed in 0.547 seconds
PL/SQL procedure successfully completed.
PL/SQL procedure successfully completed.



...nt | VISIT | student~1 | student~2 | student~3 | student~4 | Relationa

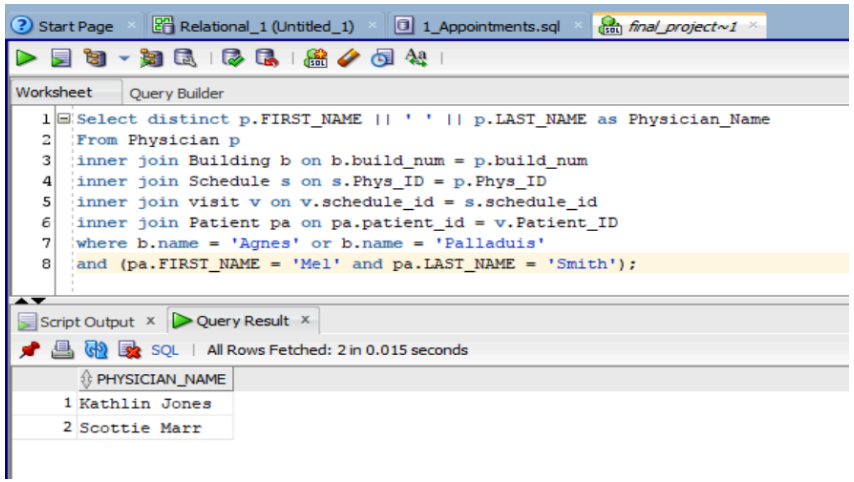
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Inde

Sort.. | Filter:

	VISIT_ID	SCHEDULE_ID	PATIENT_ID	VISIT_TIME
1	999998	9999	1000	02-JUN-14 03.00.00.000000000 PM
2	999999	10005	1008	10-JUN-17 12.00.00.000000000 PM
3	1000000	10000	1000	02-JUN-17 03.00.00.000000000 PM
4	1000001	10001	1001	04-JUN-17 12.30.00.000000000 PM
5	1000002	10001	1002	04-JUN-17 01.00.00.000000000 PM
6	1000003	10002	1003	05-JUN-17 10.00.00.000000000 AM
7	1000004	10002	1003	05-JUN-17 10.30.00.000000000 AM
8	1000005	10003	1005	06-JUN-17 02.00.00.000000000 PM
9	1000006	10003	1006	06-JUN-17 03.00.00.000000000 PM
10	1000007	10005	1007	10-JUN-17 11.00.00.000000000 AM
11	1000008	10005	1008	10-JUN-17 11.30.00.000000000 AM
12	1000009	10006	1009	12-JUN-17 09.00.00.000000000 AM
13	1000010	10006	1010	12-JUN-17 10.00.00.000000000 AM
14	1000011	10006	1011	12-JUN-17 03.30.00.000000000 PM
15	1000012	9999	1012	02-JUN-14 02.00.00.000000000 PM
16	999994	10012	1002	19-SEP-17 09.00.00.000000000 AM
17	999995	10012	1003	19-SEP-17 09.30.00.000000000 AM
18	999996	10014	1005	20-SEP-17 09.00.00.000000000 AM
19	999997	10014	1006	20-SEP-17 09.30.00.000000000 AM
20	1000013	10000	1000	02-JUN-17 02.00.00.000000000 PM
21	1000014	10001	1000	04-JUN-17 02.30.00.000000000 PM

b. Management requests the names of all physicians who work in the “Agnes” or “Palladuis” buildings who have attended appointments with a particular patient. Write a single query that retrieves this information for a patient of your choosing.

***The Patient that was chosen is Mel Smith.*



Use Case 2: Waiting List

a. A patient phones a physician's administrative assistant asking for an appointment, and the assistant adds the patient to the waiting list so that the patient will be the next one to be scheduled for a canceled appointment. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a patient and physician of your choosing.

There are two queries added before we run the creation of the ADD_Waitlist stored procedure:

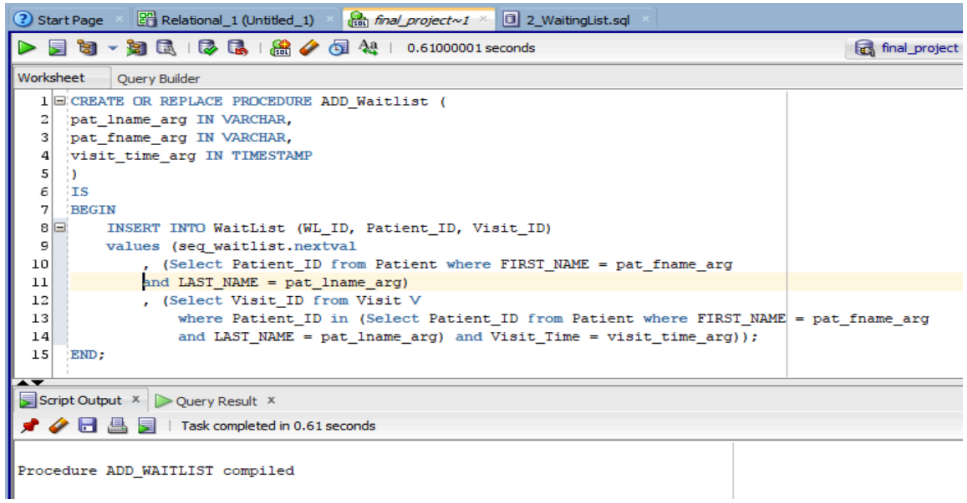
1. I have added a constraint that ensures that visits can't be double booked. This is done by ensuring that the combination of Schedule_ID and Visit_Time are unique to the Visit table. If a double booking attempts to occur, an error with the informative constraint name "Visit_Already_Booked" will pop up.
2. A "Create Sequence" query to automate the generation of the WL_ID Primary Key from the WaitList table. This is added because it does not make sense to have a staff member manually add an uninformative primary key when generating a new appointment.

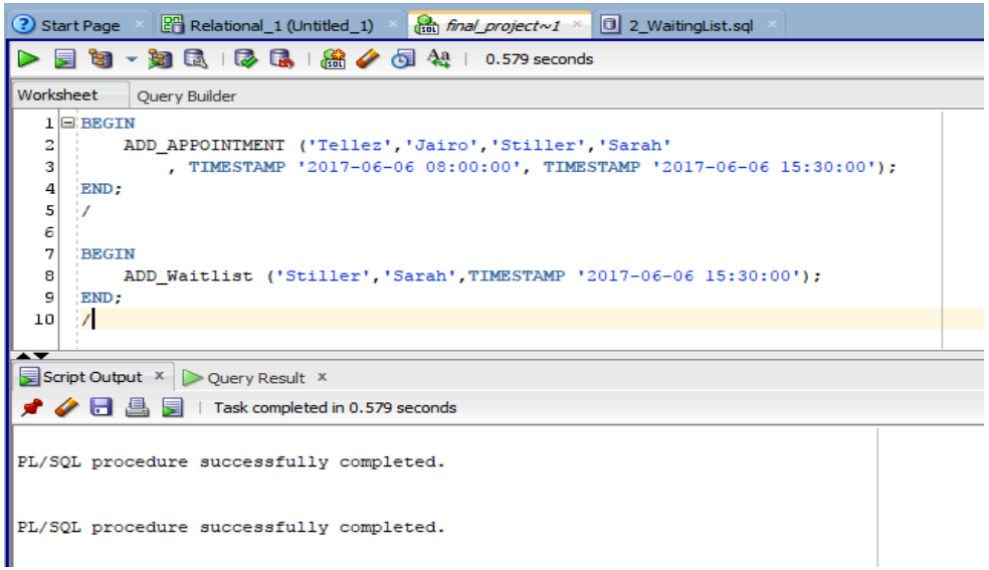
Both of the above queries can be referenced in the 2_WaitList.sql attachment in the zip along with this report.

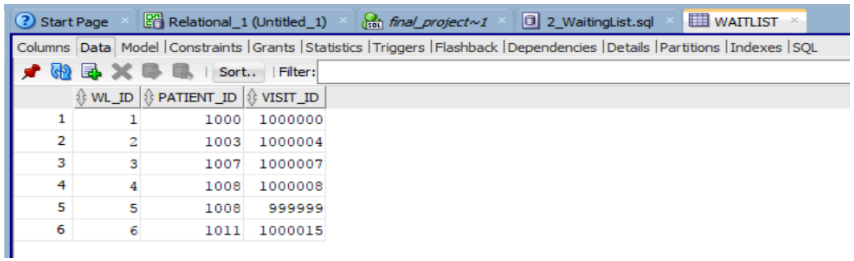
Since the WL_ID is generated in chronological order, the patient with the lowest WL_ID from the filtered entries by Physician should be the first patient to get off the wait list. For example, if WL_ID 3, 7, 20 are all waiting for the same doctor, then WL_ID = 3 is the first one to be contacted if there is an availability.

After booking a backup appointment (see Use Case 1), staff can then use the procedure below to create a wait list entry. All that is needed is the Patient Name, and the date and time of their backup appointment.

The below shows Sarah Stiller (Patient) booking a backup appointment with Dr. Tellez and adding herself to the Wait List.


Start Page
Relational_1 (Untitled_1)
final_project~1
2_WaitingList.sql
0.61000001 seconds
final_project
Worksheet
Query Builder
CREATE OR REPLACE PROCEDURE ADD_Waitlist (
pat_lname_arg IN VARCHAR,
pat_fname_arg IN VARCHAR,
visit_time_arg IN TIMESTAMP
)
IS
BEGIN
INSERT INTO WaitList (WL_ID, Patient_ID, Visit_ID)
values (seq_waitlist.nextval
, (Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
and LAST_NAME = pat_lname_arg)
, (Select Visit_ID from Visit V
where Patient_ID in (Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
and LAST_NAME = pat_lname_arg) and Visit_Time = visit_time_arg));
END;
Script Output
Query Result
Task completed in 0.61 seconds
Procedure ADD_WAITLIST compiled

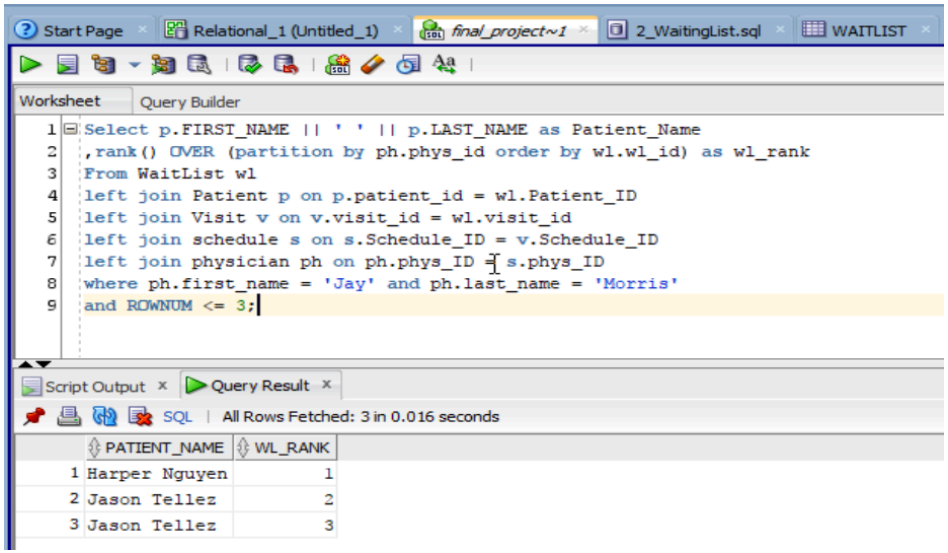

Start Page
Relational_1 (Untitled_1)
final_project~1
2_WaitingList.sql
0.579 seconds
Worksheet
Query Builder
BEGIN
ADD_APPOINTMENT ('Tellez','Jairo','Stiller','Sarah'
, TIMESTAMP '2017-06-06 08:00:00', TIMESTAMP '2017-06-06 15:30:00');
END;
/
BEGIN
ADD_Waitlist ('Stiller','Sarah',TIMESTAMP '2017-06-06 15:30:00');
END;
/
Script Output
Query Result
Task completed in 0.579 seconds
PL/SQL procedure successfully completed.
PL/SQL procedure successfully completed.



	WL_ID	PATIENT_ID	VISIT_ID
1	1	1000	1000000
2	2	1003	1000004
3	3	1007	1000007
4	4	1008	1000008
5	5	1008	999999
6	6	1011	1000015

b. A physician has had three appointment cancellations today, so requests the names of the first three patients on their waiting list. Write a single query that retrieves this information for a physician of your choosing.

The query below shows the first three patients on the wait list for Dr. Jay Morris. This is done with a rank to clearly rank positions 1, 2 and 3 and then ROWNUM <= 3 allows us to minimize to the top 3 results.



	PATIENT_NAME	WL_RANK
1	Harper Nguyen	1
2	Jason Tellez	2
3	Jason Tellez	3

Use Case 3: Insurance Plans

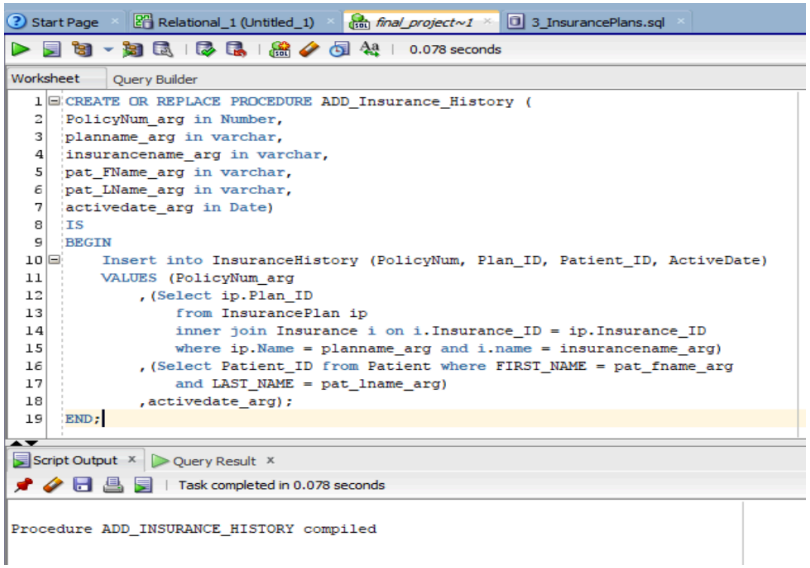
a. A new patient comes into the BUAHC, and a nurse enters the patient's insurance plan enrollment into the system. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a patient and insurance plan of your choosing.

There are two additions before the stored procedure to ADD_Insurance_History:

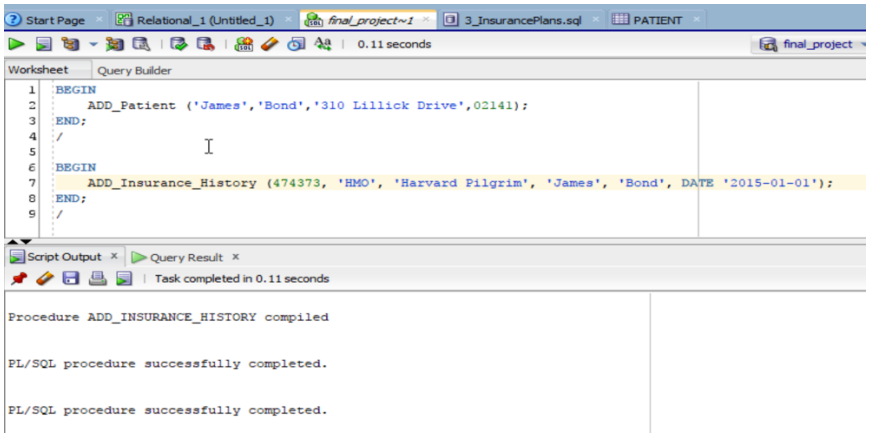
1. A "Create Sequence" query to automate the generation of the Patient_ID Primary Key from the Patient table. This is added because it does not make sense to have a staff member manually add an uninformative primary key when generating a new appointment.
2. A stored procedure to add a new Patient to the Patient table. This is done by collecting the Patient's Name and Address.

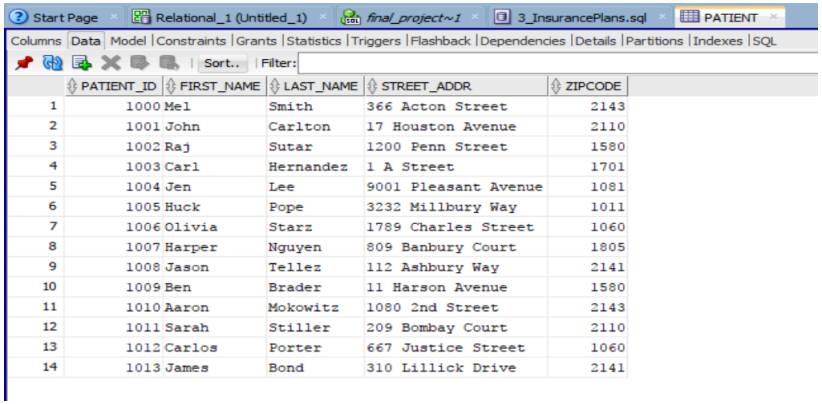
Both of the above queries can be referenced in the 3_InsurancePlans.sql attachment in the zip along with this report.

The below procedure allows staff to add the insurance information for a patient. This is done by entering in the Insurance Policy Number, the Plan Name, the Insurance Company, Patient Name and date that the Insurance began being active.

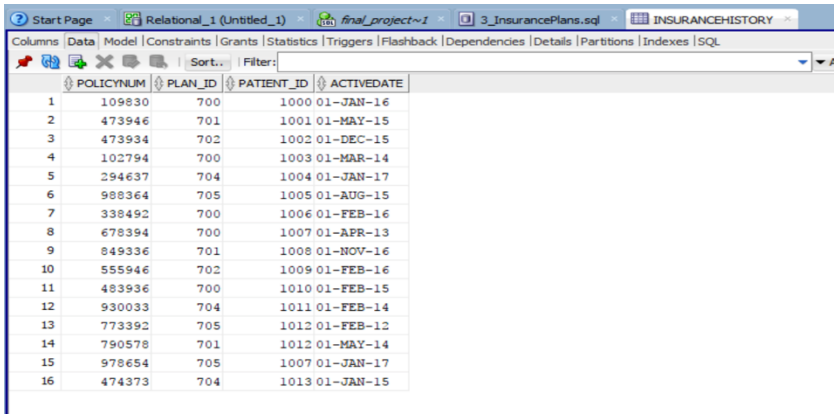


The below shows a test case for a new patient, James Bond who is added as a New Patient and then has his Insurance information added.





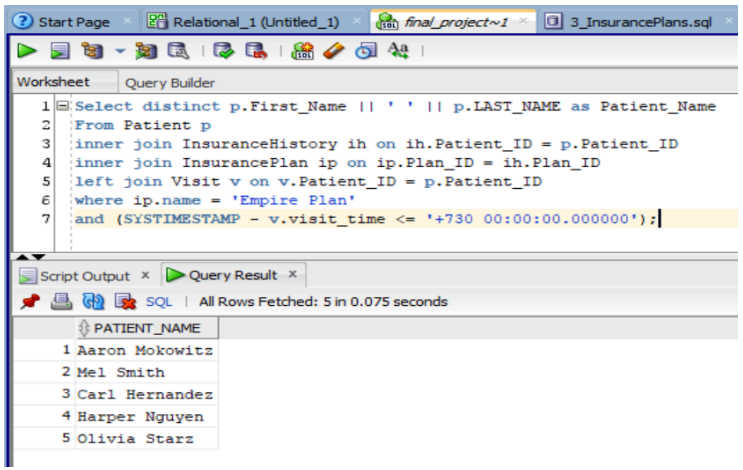
	PATIENT_ID	FIRST_NAME	LAST_NAME	STREET_ADDR	ZIPCODE
1	1000	Mel	Smith	366 Acton Street	2143
2	1001	John	Carlton	17 Houston Avenue	2110
3	1002	Raj	Sutar	1200 Penn Street	1580
4	1003	Carl	Hernandez	1 A Street	1701
5	1004	Jen	Lee	9001 Pleasant Avenue	1081
6	1005	Huck	Pope	3232 Millbury Way	1011
7	1006	Olivia	Starz	1789 Charles Street	1060
8	1007	Harper	Nguyen	809 Banbury Court	1805
9	1008	Jason	Tellez	112 Ashbury Way	2141
10	1009	Ben	Brader	11 Harson Avenue	1580
11	1010	Aaron	Mokowitz	1080 2nd Street	2143
12	1011	Sarah	Stiller	209 Bombay Court	2110
13	1012	Carlos	Porter	667 Justice Street	1060
14	1013	James	Bond	310 Lillick Drive	2141



	POLICYNUM	PLAN_ID	PATIENT_ID	ACTIVEDATE
1	109830	700	1000	01-JAN-16
2	473946	701	1001	01-MAY-15
3	473934	702	1002	01-DEC-15
4	102794	700	1003	01-MAR-14
5	294637	704	1004	01-JAN-17
6	988364	705	1005	01-AUG-15
7	338492	700	1006	01-FEB-16
8	678394	700	1007	01-APR-13
9	849336	701	1008	01-NOV-16
10	555946	702	1009	01-FEB-16
11	483936	700	1010	01-FEB-15
12	930033	704	1011	01-FEB-14
13	773392	705	1012	01-FEB-12
14	790578	701	1012	01-MAY-14
15	978654	705	1007	01-JAN-17
16	474373	704	1013	01-JAN-15

b. Management requests the names of all current patients who are enrolled in the “Empire Plan” insurance plan. Management defines a “current” patient as one who has had an appointment in the past two years. Write a single query that retrieves this information for a patient of your choosing.

The below is a query that returns all patients that have the Empire Plan who have had an Appointment in the last 2 years (or 730 days).



	PATIENT_NAME
1	Aaron Mokowitz
2	Mel Smith
3	Carl Hernandez
4	Harper Nguyen
5	Olivia Starz

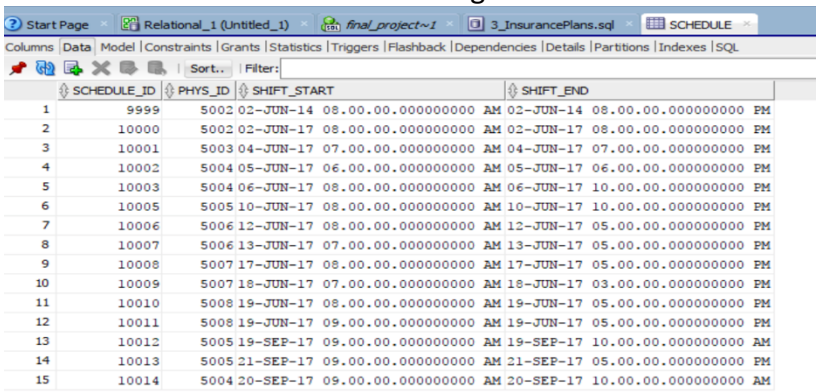
Use Case 4: Physician Schedules

a. A physician decides to work an extra hour each day, specifically by beginning work an hour earlier, in order to support two additional appointments per day. Develop a parameterized stored procedure to adjust the physician's schedule, then invoke the stored procedure for a physician of your choosing.

The below stored procedure adds an hour to all schedule shifts of a specified Physician in the future. Adding the Shift_Start > Systimestamp ensures that passed Schedule entries are not altered.

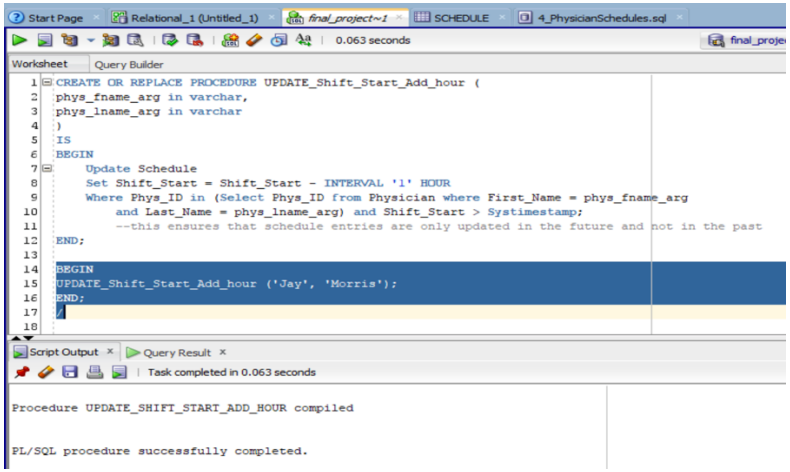
This is tested out for the Physician Jay Morris who has a Physician ID of 5005.

Before Change:

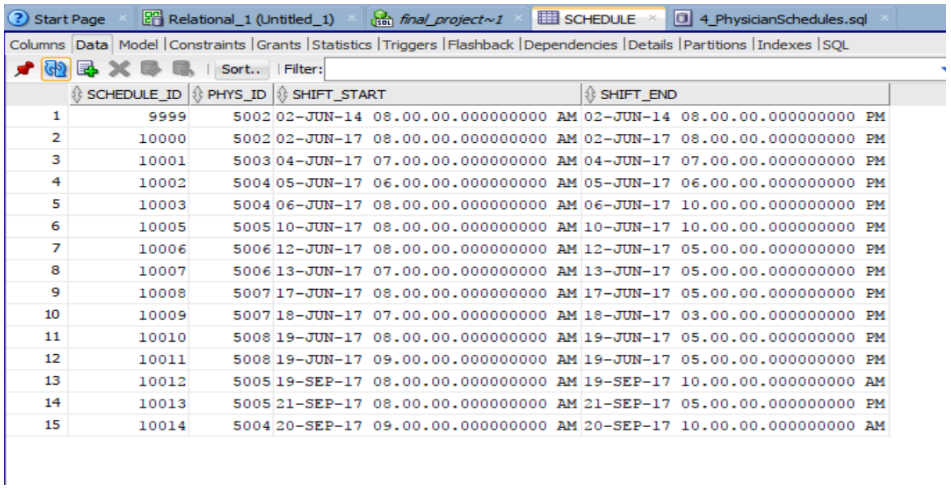


	SCHEDULE_ID	PHYS_ID	SHIFT_START	SHIFT_END
1	9999	5002	02-JUN-14 08.00.00.000000000 AM	02-JUN-14 08.00.00.000000000 PM
2	10000	5002	02-JUN-17 08.00.00.000000000 AM	02-JUN-17 08.00.00.000000000 PM
3	10001	5003	04-JUN-17 07.00.00.000000000 AM	04-JUN-17 07.00.00.000000000 PM
4	10002	5004	05-JUN-17 06.00.00.000000000 AM	05-JUN-17 06.00.00.000000000 PM
5	10003	5004	06-JUN-17 08.00.00.000000000 AM	06-JUN-17 10.00.00.000000000 PM
6	10005	5005	10-JUN-17 08.00.00.000000000 AM	10-JUN-17 10.00.00.000000000 PM
7	10006	5006	12-JUN-17 08.00.00.000000000 AM	12-JUN-17 05.00.00.000000000 PM
8	10007	5006	13-JUN-17 07.00.00.000000000 AM	13-JUN-17 05.00.00.000000000 PM
9	10008	5007	17-JUN-17 08.00.00.000000000 AM	17-JUN-17 05.00.00.000000000 PM
10	10009	5007	18-JUN-17 07.00.00.000000000 AM	18-JUN-17 03.00.00.000000000 PM
11	10010	5008	19-JUN-17 08.00.00.000000000 AM	19-JUN-17 05.00.00.000000000 PM
12	10011	5008	19-JUN-17 09.00.00.000000000 AM	19-JUN-17 05.00.00.000000000 PM
13	10012	5005	19-SEP-17 09.00.00.000000000 AM	19-SEP-17 10.00.00.000000000 AM
14	10013	5005	21-SEP-17 09.00.00.000000000 AM	21-SEP-17 05.00.00.000000000 PM
15	10014	5004	20-SEP-17 09.00.00.000000000 AM	20-SEP-17 10.00.00.000000000 AM

Procedure and Execution:



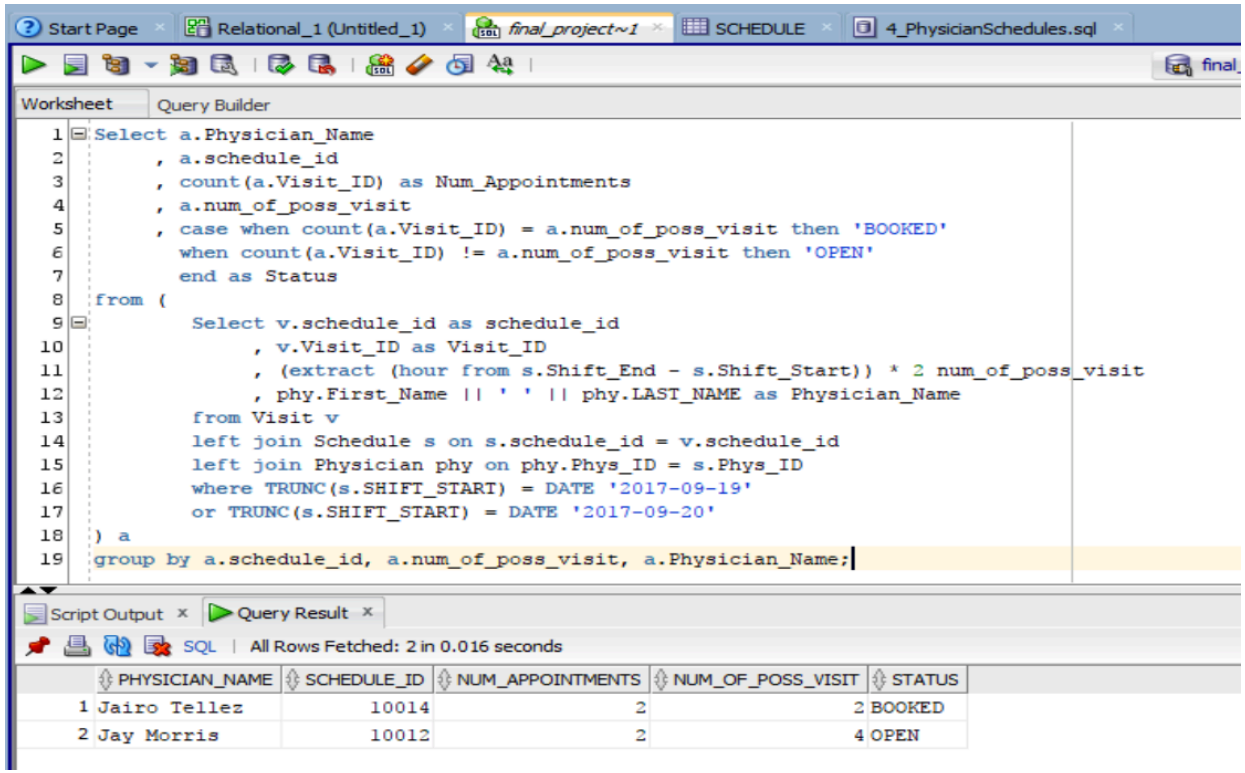
After:



	SCHEDULE_ID	PHYS_ID	SHIFT_START	SHIFT_END
1	9999	5002	02-JUN-14 08.00.00.000000000 AM	02-JUN-14 08.00.00.000000000 PM
2	10000	5002	02-JUN-17 08.00.00.000000000 AM	02-JUN-17 08.00.00.000000000 PM
3	10001	5003	04-JUN-17 07.00.00.000000000 AM	04-JUN-17 07.00.00.000000000 PM
4	10002	5004	05-JUN-17 06.00.00.000000000 AM	05-JUN-17 06.00.00.000000000 PM
5	10003	5004	06-JUN-17 08.00.00.000000000 AM	06-JUN-17 10.00.00.000000000 PM
6	10005	5005	10-JUN-17 08.00.00.000000000 AM	10-JUN-17 10.00.00.000000000 PM
7	10006	5006	12-JUN-17 08.00.00.000000000 AM	12-JUN-17 05.00.00.000000000 PM
8	10007	5006	13-JUN-17 07.00.00.000000000 AM	13-JUN-17 05.00.00.000000000 PM
9	10008	5007	17-JUN-17 08.00.00.000000000 AM	17-JUN-17 05.00.00.000000000 PM
10	10009	5007	18-JUN-17 07.00.00.000000000 AM	18-JUN-17 03.00.00.000000000 PM
11	10010	5008	19-JUN-17 08.00.00.000000000 AM	19-JUN-17 05.00.00.000000000 PM
12	10011	5008	19-JUN-17 09.00.00.000000000 AM	19-JUN-17 05.00.00.000000000 PM
13	10012	5005	19-SEP-17 08.00.00.000000000 AM	19-SEP-17 10.00.00.000000000 AM
14	10013	5005	21-SEP-17 08.00.00.000000000 AM	21-SEP-17 05.00.00.000000000 PM
15	10014	5004	20-SEP-17 09.00.00.000000000 AM	20-SEP-17 10.00.00.000000000 AM

b. A receptionist needs to know the names of all physicians that are booked for the next two days. Being booked means the physicians have no available appointments for either day. Write a single query that retrieves this information for the receptionist.

The query below assumes the next two days are 09/19/17 and 09/20/17. The receptionist can run the query against the Schedule of all Physicians over the next two days and get an easy to read status of “Booked” if they have no availability or “Open” if they do have availability.



Use Case 5: Wait List Checker and Reporting

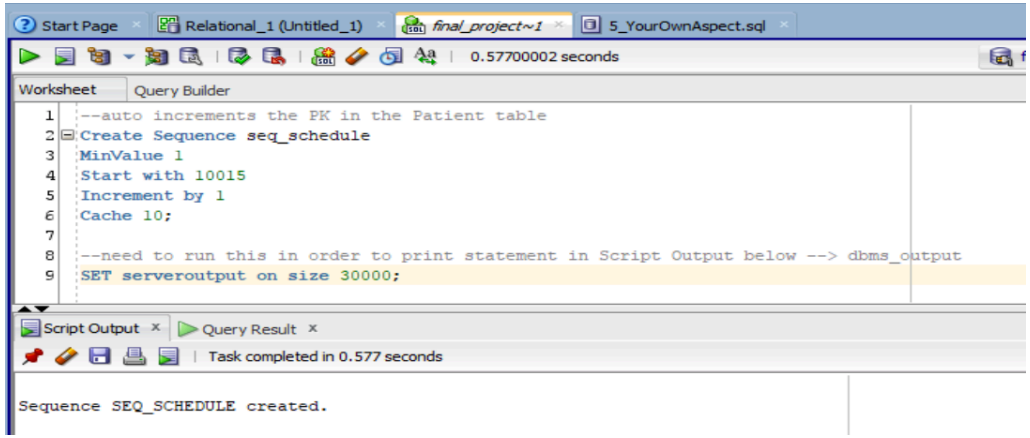
a. When a doctor decides to work on his/her day off (new last minute schedule entry), check the waitlist table on who should be called for an appointment first. Make sure that their back up appointment is a date after the doctor's new shift date.

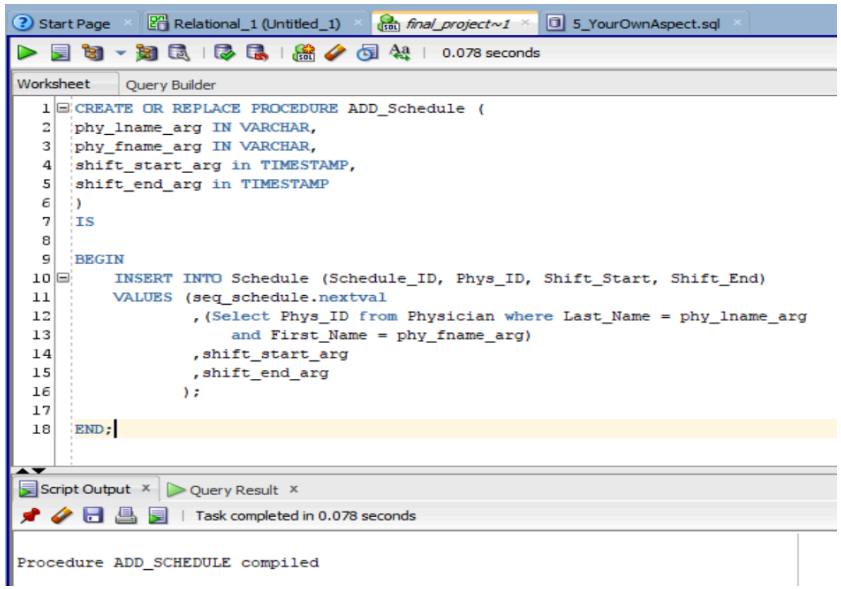
The first part of this Use Case was to create a “Create Sequence” statement to auto generate the Primary Key from the Schedule table. This then allowed me to create a stored Procedure to add a Schedule entry. This is done by entering the Physician’s Name, Start Date and Time and End Date and Time of their shift.

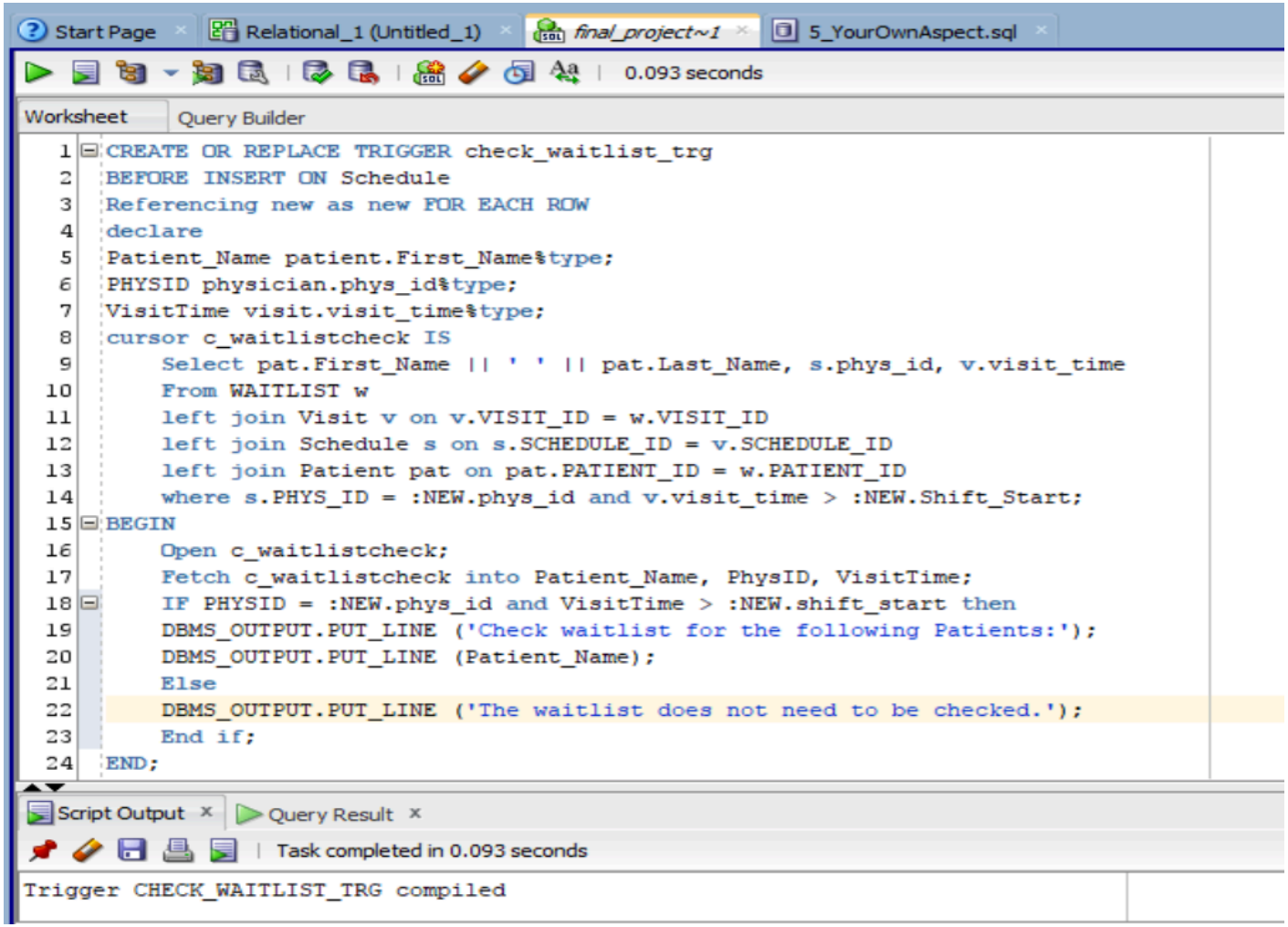
This procedure then sets off a trigger that will inform staff if they need to check the Wait List. If the Wait List does not need to be checked, this means that the Physician doesn’t have a wait list or the backup appointments are before the new schedule entry. This is important because if a Physician adds a shift on Thursday and a Patient has a backup appointment on Wednesday, they would not want to delay further. If the Wait List does need to be checked, it will not only print a statement but will also tell you the Patients to contact.

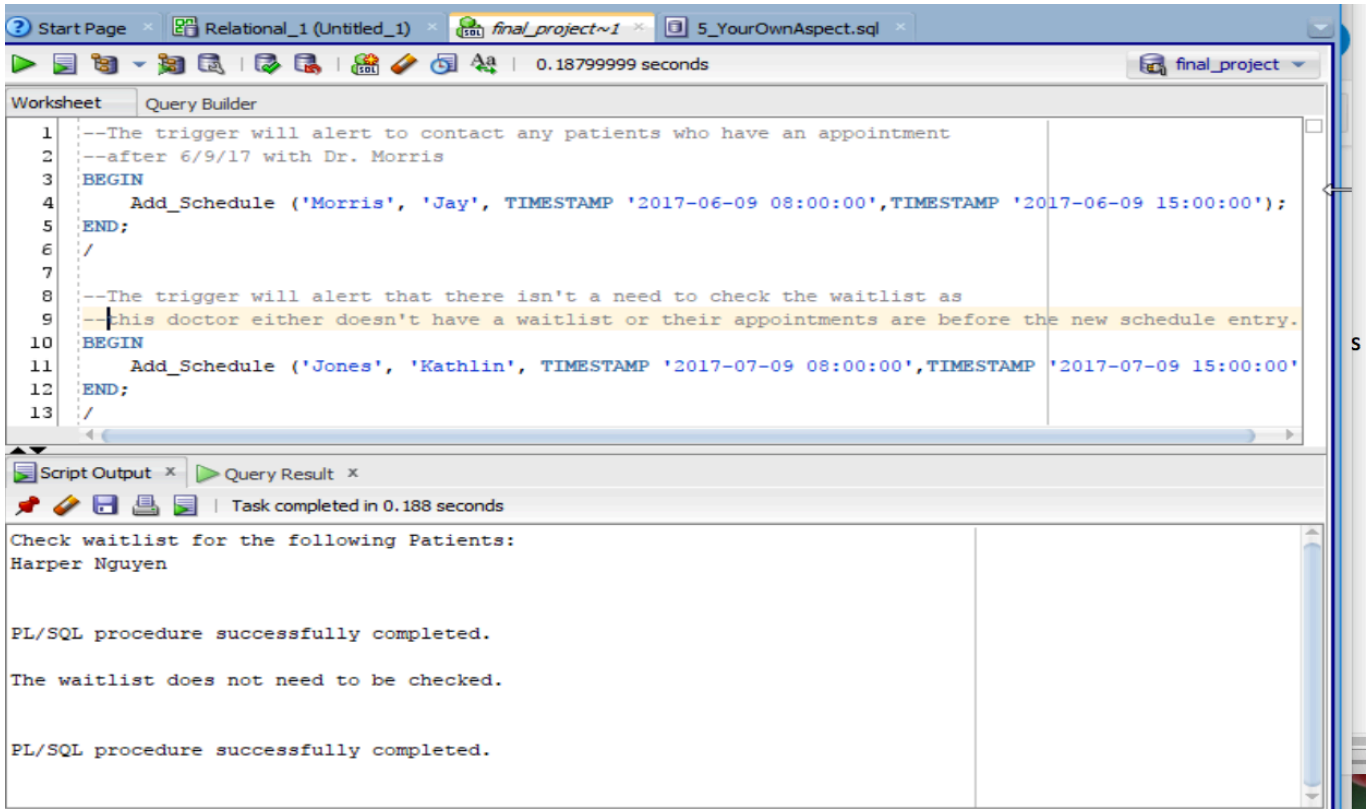
In order to receive the print statements, you must enable the server output and set a character limit.

The situation here is that both Physicians Dr. Morris and Dr. Jones are adding an extra day to their schedule. The receptionist will have an alert for Dr. Morris that the Wait List needs to be checked and that Harper Nguyen should be called to have her appointment rescheduled for sooner. Dr. Jones, however, will have an alert that the Wait List does not need to be checked as she either does not have a wait list or her patients have back up appointments before her new schedule date.


Start Page
Relational_1 (Untitled_1)
final_project~1
5_YourOwnAspect.sql
0.57700002 seconds
Worksheet
Query Builder
--auto increments the PK in the Patient table
Create Sequence seq_schedule
MinValue 1
Start with 10015
Increment by 1
Cache 10;
--need to run this in order to print statement in Script Output below --> dbms_output
SET serveroutput on size 30000;
Script Output
Query Result
Task completed in 0.577 seconds
Sequence SEQ_SCHEDULE created.

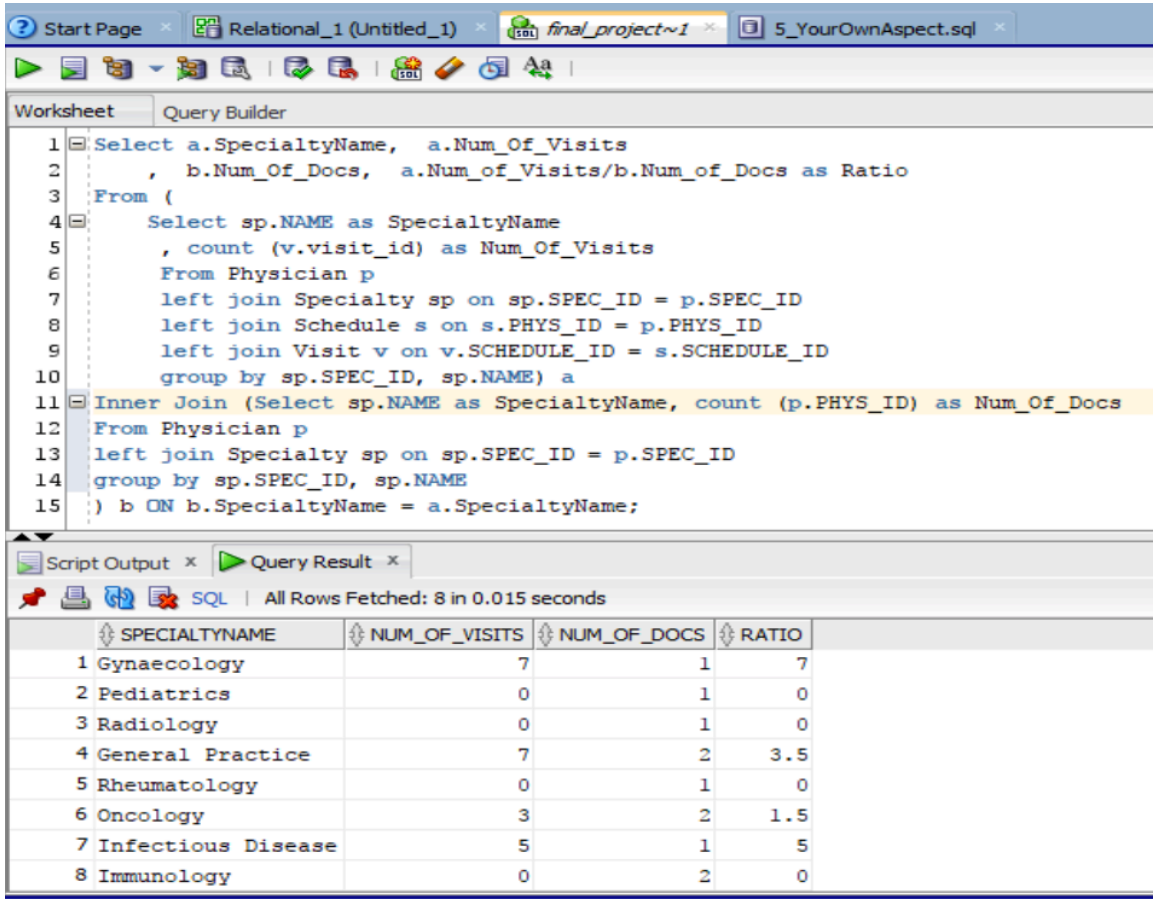

Start Page
Relational_1 (Untitled_1)
final_project~1
5_YourOwnAspect.sql
0.078 seconds
Worksheet
Query Builder
CREATE OR REPLACE PROCEDURE ADD_Schedule (
phy_lname_arg IN VARCHAR,
phy_fname_arg IN VARCHAR,
shift_start_arg in TIMESTAMP,
shift_end_arg in TIMESTAMP
)
IS
BEGIN
INSERT INTO Schedule (Schedule_ID, Phys_ID, Shift_Start, Shift_End)
VALUES (seq_schedule.nextval
,(Select Phys_ID from Physician where Last_Name = phy_lname_arg
and First_Name = phy_fname_arg)
,shift_start_arg
,shift_end_arg
);
END;
Script Output
Query Result
Task completed in 0.078 seconds
Procedure ADD_SCHEDULE compiled


Start Page
Relational_1 (Untitled_1)
final_project~1
5_YourOwnAspect.sql
0.093 seconds
Worksheet
Query Builder
1 CREATE OR REPLACE TRIGGER check_waitlist_trg
2 BEFORE INSERT ON Schedule
3 Referencing new as new FOR EACH ROW
4 declare
5 Patient_Name patient.First_Name%type;
6 PHYSID physician.phys_id%type;
7 VisitTime visit.visit_time%type;
8 cursor c_waitlistcheck IS
9 Select pat.First_Name || ' ' || pat.Last_Name, s.phys_id, v.visit_time
10 From WAITLIST w
11 left join Visit v on v.VISIT_ID = w.VISIT_ID
12 left join Schedule s on s.SCHEDULE_ID = v.SCHEDULE_ID
13 left join Patient pat on pat.PATIENT_ID = w.PATIENT_ID
14 where s.PHYS_ID = :NEW.phys_id and v.visit_time > :NEW.Shift_Start;
15 BEGIN
16 Open c_waitlistcheck;
17 Fetch c_waitlistcheck into Patient_Name, PhysID, VisitTime;
18 IF PHYSID = :NEW.phys_id and VisitTime > :NEW.shift_start then
19 DBMS_OUTPUT.PUT_LINE ('Check waitlist for the following Patients:');
20 DBMS_OUTPUT.PUT_LINE (Patient_Name);
21 Else
22 DBMS_OUTPUT.PUT_LINE ('The waitlist does not need to be checked.');
23 End if;
24 END;
Script Output
Query Result
Task completed in 0.093 seconds
Trigger CHECK_WAITLIST_TRG compiled


Start Page
Relational_1 (Untitled_1)
final_project~1
5_YourOwnAspect.sql
0.18799999 seconds
final_project
Worksheet
Query Builder
--The trigger will alert to contact any patients who have an appointment
--after 6/9/17 with Dr. Morris
BEGIN
Add_Schedule ('Morris', 'Jay', TIMESTAMP '2017-06-09 08:00:00',TIMESTAMP '2017-06-09 15:00:00');
END;
/
--The trigger will alert that there isn't a need to check the waitlist as
--this doctor either doesn't have a waitlist or their appointments are before the new schedule entry.
BEGIN
Add_Schedule ('Jones', 'Kathlin', TIMESTAMP '2017-07-09 08:00:00',TIMESTAMP '2017-07-09 15:00:00'
END;
/
Script Output
Query Result
Task completed in 0.188 seconds
Check waitlist for the following Patients:
Harper Nguyen
PL/SQL procedure successfully completed.
The waitlist does not need to be checked.
PL/SQL procedure successfully completed.

b. The hospital has received the budget to hire a new doctor but are unsure which specialty is in the highest demand. Find the specialty with the most appointments and divide it by the number of staffed doctors in that specialty to assess appointment to Physician Specialty ratio and decide what type of doctor should be hired.

For this query, we found that Gynecology is in the highest demand with 7 appointments and only one doctor, thus making the ratio 7.



	SPECIALTYNAME	NUM_OF_VISITS	NUM_OF_DOCS	RATIO
1	Gynaecology	7	1	7
2	Pediatrics	0	1	0
3	Radiology	0	1	0
4	General Practice	7	2	3.5
5	Rheumatology	0	1	0
6	Oncology	3	2	1.5
7	Infectious Disease	5	1	5
8	Immunology	0	2	0

Creation of an Index

I created the following Index for Schedule_ID and Phys_ID from the Schedule table. This was done to quickly retrieve information on a Physician from the Patient, Visit and Wait List entities. All of these entities must go through the Schedule table in order to create a relationship to the attending Physician. I chose to make a unique index as the schedule_id and phys_id combination is unique. The logic for this is that a specific schedule_id will never belong to another physician.