

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

<u>TABLE OF CONTENTS</u>	2
<u>INTRODUCTION & PROPOSAL</u>	3
<u>CONCEPTUAL ENTITY RELATIONSHIP DIAGRAM</u>	4
<u>BUSINESS RULES</u>	5
<u>LOGICAL ENTITY RELATIONSHIP DIAGRAM</u>	6
<u>USE CASES: SCREENSHOTS AND EXPLANATIONS</u>	7
 USE CASE 1: APPOINTMENTS	7
 USE CASE 2: WAITING LIST	11
 USE CASE 3: INSURANCE PLANS	14
 USE CASE 4: PHYSICIAN SCHEDULES	17
 USE CASE 5: WAIT LIST CHECKER AND REPORTING	20
<u>CREATION OF AN INDEX</u>	25

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

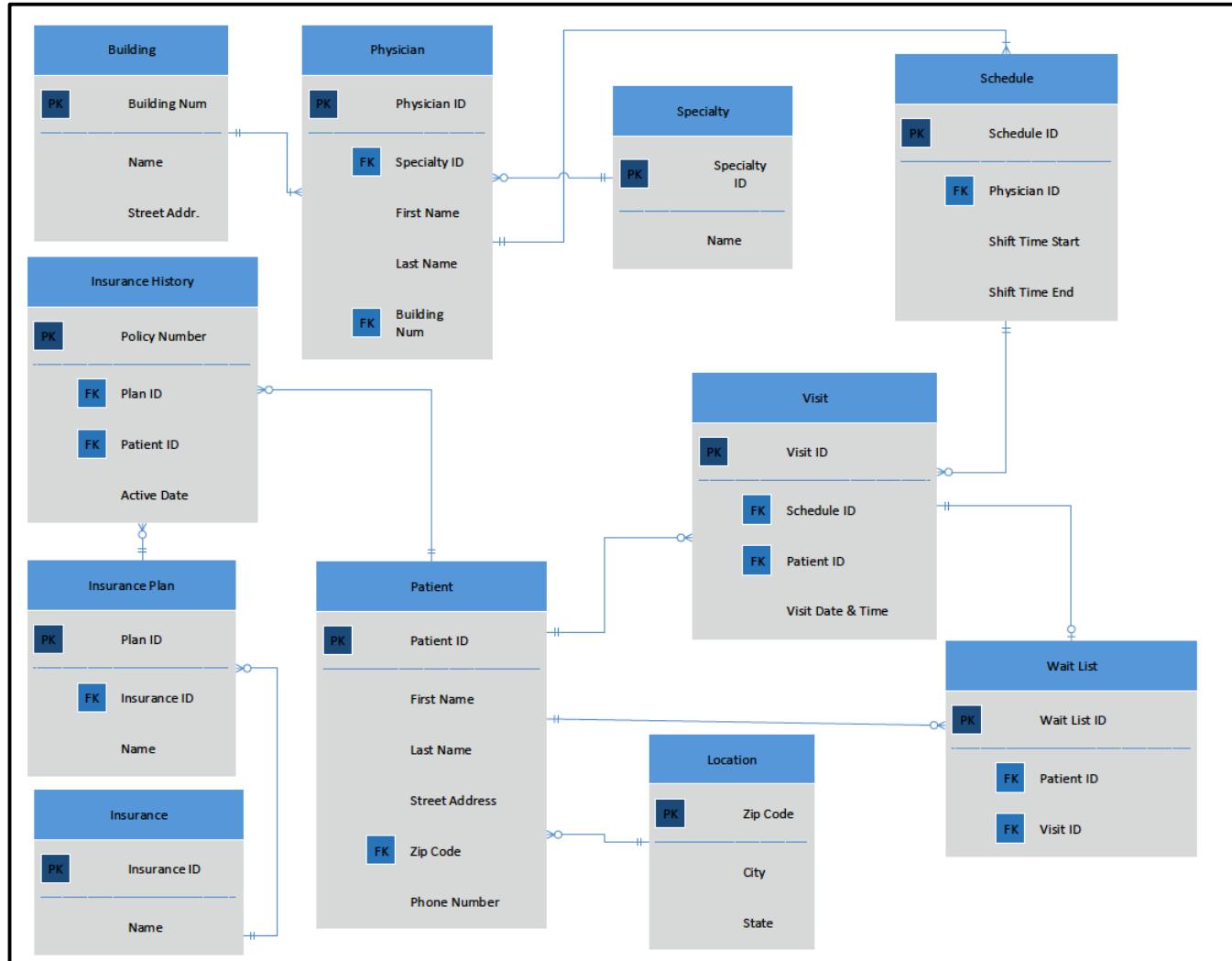
As an addition for this proposal, the design also includes the following two **additional** use cases:

- 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)

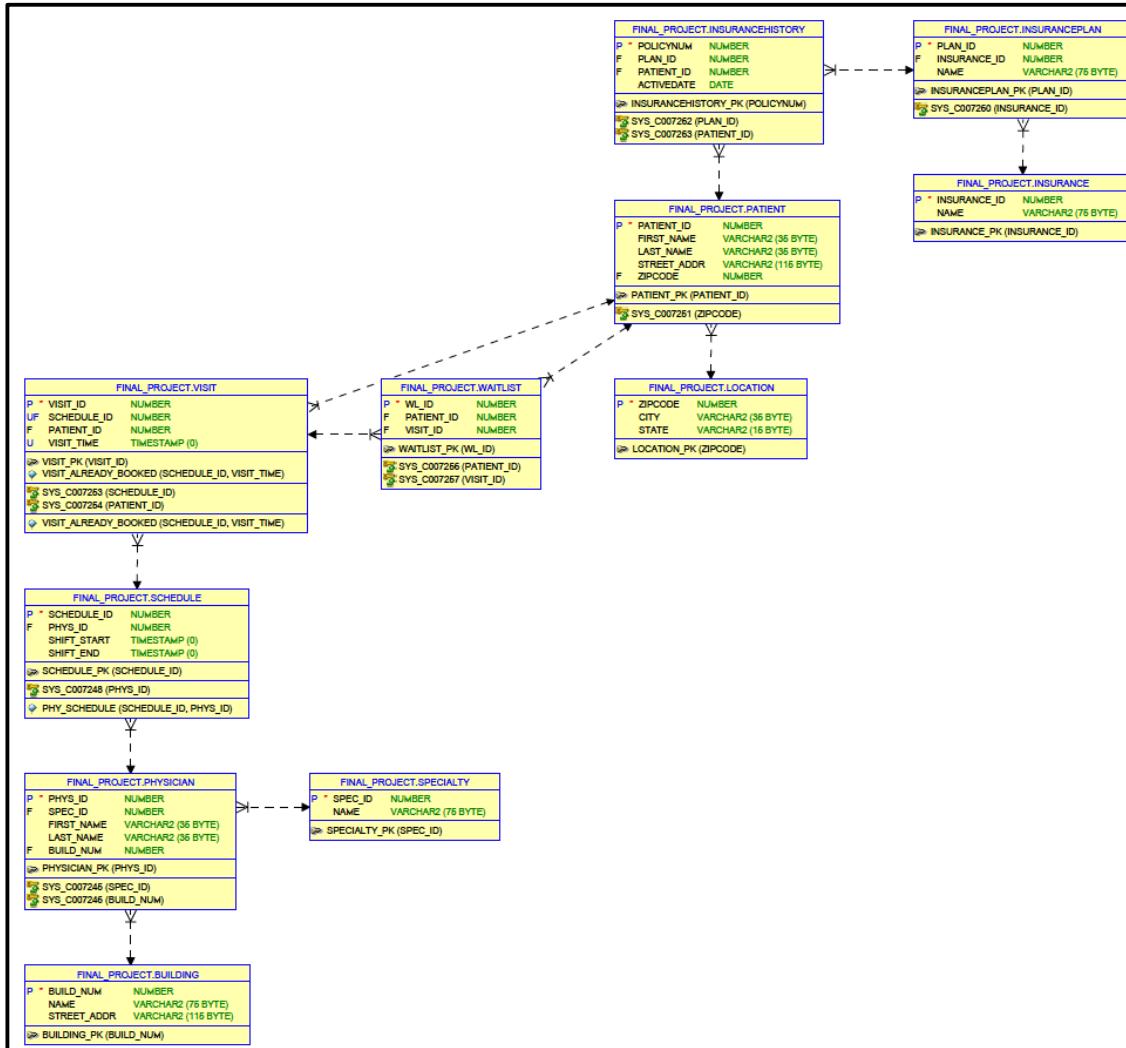
A **visit** may or may not be listed in a **wait list entry**. (singular, optional)

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.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for student~3, student~4, Relational_1 (Untitled_1), 1_Appointments.sql, and final_project~1. The main window title is "final". The "Worksheet" tab is selected, showing a PL/SQL code editor. The code defines a procedure ADD_APPOINTMENT with parameters phy_lname_arg, phy_fname_arg, pat_lname_arg, pat_fname_arg, shift_start_arg, and visit_time_arg. It inserts a new row into the Visit table, selecting 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). It also selects Patient_ID from Patient where FIRST_NAME = pat_fname_arg and LAST_NAME = pat_lname_arg. The procedure ends with an END; statement. Below the code is a "Script Output" window showing the message "Procedure ADD_APPOINTMENT compiled" and a completion time of 0.172 seconds.

```
1 CREATE OR REPLACE PROCEDURE ADD_APPOINTMENT(
2   phy_lname_arg IN VARCHAR,
3   phy_fname_arg IN VARCHAR,
4   pat_lname_arg IN VARCHAR,
5   pat_fname_arg IN VARCHAR,
6   shift_start_arg IN TIMESTAMP,
7   visit_time_arg IN TIMESTAMP
8 )
9 IS
10 BEGIN
11   Insert into Visit
12     (Visit_ID, Schedule_ID, Patient_ID, Visit_Time)
13   VALUES (seq_visit.nextval
14           ,(Select Schedule_ID from Schedule where Shift_Start = shift_start_arg
15             and Phys_ID = (Select Phys_ID from Physician where FIRST_NAME = phy_fname_arg
16                           and LAST_NAME = phy_lname_arg))
17           ,(Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
18             and LAST_NAME = pat_lname_arg)
19           ,visit_time_arg);
20 END;
```

Script Output x | Task completed in 0.172 seconds

Procedure ADD_APPOINTMENT compiled

The screenshot shows the Oracle SQL Developer interface with the same session tabs and title bar as the first screenshot. The "Worksheet" tab is selected, showing a PL/SQL code editor. The code consists of two BEGIN...END blocks. The first block calls ADD_APPOINTMENT with parameters ('Jones', 'Kathlin', 'Smith', 'Mel', '2017-06-02 08:00:00', '2017-06-02 14:00:00'). The second block calls ADD_APPOINTMENT with parameters ('Marr', 'Scottie', 'Smith', 'Mel', '2017-06-04 07:00:00', '2017-06-04 14:30:00'). Both blocks end with a slash (/). Below the code is a "Script Output" window showing two messages: "PL/SQL procedure successfully completed." for each call, and a completion time of 0.547 seconds.

```
1 BEGIN
2   ADD_APPOINTMENT ('Jones','Kathlin','Smith','Mel'
3                     , TIMESTAMP '2017-06-02 08:00:00', TIMESTAMP '2017-06-02 14:00:00');
4 END;
5 /
6 BEGIN
7   ADD_APPOINTMENT ('Marr','Scottie','Smith','Mel'
8                     , TIMESTAMP '2017-06-04 07:00:00', TIMESTAMP '2017-06-04 14:30:00');
9 END;
10 /
11 /
```

Script Output x | 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 Relations

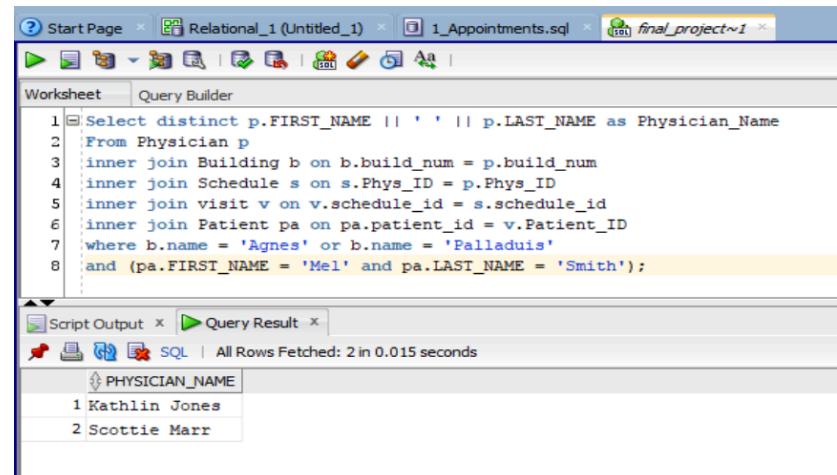
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes

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.*



The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Start Page', 'Relational_1 (Untitled_1)', '1_Appointments.sql', and 'final_project~1'. Below the menu is a toolbar with various icons. The main area is divided into two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the following SQL code:

```
1 Select distinct p.FIRST_NAME || ' ' || p.LAST_NAME as Physician_Name
2 From Physician p
3 inner join Building b on b.build_num = p.build_num
4 inner join Schedule s on s.Phys_ID = p.Phys_ID
5 inner join visit v on v.schedule_id = s.schedule_id
6 inner join Patient pa on pa.patient_id = v.Patient_ID
7 where b.name = 'Agnes' or b.name = 'Palladuis'
8 and (pa.FIRST_NAME = 'Mel' and pa.LAST_NAME = 'Smith');
```

The 'Query Result' tab is selected at the bottom, showing the output:

PHYSICIAN_NAME
1 Kathlin Jones
2 Scottie Marr

Below the table, it says 'All Rows Fetched: 2 in 0.015 seconds'.

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.

The screenshot shows the Oracle SQL Developer interface. The title bar includes tabs for 'Start Page', 'Relational_1 (Untitled_1)', 'final_project~1', and '2_WaitingList.sql'. The status bar at the top right shows '0.61000001 seconds'. The main area is a 'Worksheet' tab showing the following PL/SQL code:

```
1 CREATE OR REPLACE PROCEDURE ADD_Waitlist (
2     pat_lname_arg IN VARCHAR,
3     pat_fname_arg IN VARCHAR,
4     visit_time_arg IN TIMESTAMP
5 )
6 IS
7 BEGIN
8     INSERT INTO WaitList (WL_ID, Patient_ID, Visit_ID)
9         values (seq_waitlist.nextval
10            , (Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
11                and LAST_NAME = pat_lname_arg)
12            , (Select Visit_ID from Visit V
13                where Patient_ID in (Select Patient_ID from Patient where FIRST_NAME = pat_fname_arg
14                    and LAST_NAME = pat_lname_arg) and Visit_Time = visit_time_arg));
15 END;
```

The code is highlighted with syntax coloring. Below the worksheet, the 'Script Output' tab shows the message 'Procedure ADD_WAITLIST compiled'.

The screenshot shows the Oracle SQL Developer interface. The title bar includes tabs for 'Start Page', 'Relational_1 (Untitled_1)', 'final_project~1', and '2_WaitingList.sql'. The status bar at the top right shows '0.579 seconds'. The main area is a 'Worksheet' tab showing the following PL/SQL code:

```
1 BEGIN
2     ADD_APPOINTMENT ('Tellez','Jairo','Stiller','Sarah'
3                      , TIMESTAMP '2017-06-06 08:00:00', TIMESTAMP '2017-06-06 15:30:00');
4 END;
5 /
6
7 BEGIN
8     ADD_Waitlist ('Stiller','Sarah',TIMESTAMP '2017-06-06 15:30:00');
9 END;
10 /
```

The code is highlighted with syntax coloring. Below the worksheet, the 'Script Output' tab shows two messages: 'PL/SQL procedure successfully completed.' followed by another '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.

```

1 Select p.FIRST_NAME || ' ' || p.LAST_NAME as Patient_Name
2 ,rank() OVER (partition by ph.phys_id order by wl.wl_id) as wl_rank
3 From WaitList wl
4 left join Patient p on p.patient_id = wl.Patient_ID
5 left join Visit v on v.visit_id = wl.visit_id
6 left join schedule s on s.Schedule_ID = v.Schedule_ID
7 left join physician ph on ph.phys_ID = s.phys_ID
8 where ph.first_name = 'Jay' and ph.last_name = 'Morris'
9 and ROWNUM <= 3;

```

Script Output | Query Result | All Rows Fetched: 3 in 0.016 seconds

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 screenshot shows a SQL query editor interface with a toolbar at the top and several tabs open. The main tab is 'Worksheet' and contains the following T-SQL code:

```
1 CREATE OR REPLACE PROCEDURE ADD_Insurance_History (
2     PolicyNum_arg int,
3     planname_arg varchar,
4     insurancename_arg varchar,
5     pat_FName_arg varchar,
6     pat_LName_arg varchar,
7     activedate_arg date)
8 IS
9 BEGIN
10    Insert into InsuranceHistory (PolicyNum, Plan_ID, Patient_ID, ActiveDate)
11        VALUES (PolicyNum_arg
12                  ,(Select ip.Plan_ID
13                      from InsurancePlan ip
14                      inner join Insurance i on i.Insurance_ID = ip.Insurance_ID
15                      where ip.Name = planname_arg and i.name = insurancename_arg)
16                  ,(Select Patient_ID from Patient where FIRST_NAME = pat_FName_arg
17                      and LAST_NAME = pat_LName_arg)
18                  ,activedate_arg);
19 END;
```

The 'Script Output' tab at the bottom shows the message: "Procedure ADD_INSURANCE_HISTORY compiled".

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.

The screenshot displays two windows from Oracle SQL Developer. The top window is a 'Worksheet' showing a PL/SQL script being run. The script contains two blocks: one for adding a patient ('James Bond') with address '310 Lillick Drive', ZIP code '02141', and another for adding insurance history for patient ID 474373 with plan 'HMO', provider 'Harvard Pilgrim', and date '2015-01-01'. The bottom window shows a table named 'PATIENT' with 14 rows of data, including the newly added patient 'James Bond' at the bottom.

```
1 BEGIN
2     ADD_Patient ('James','Bond','310 Lillick Drive',02141);
3 END;
4 /
5
6 BEGIN
7     ADD_Insurance_History (474373, 'HMO', 'Harvard Pilgrim', 'James', 'Bond', DATE '2015-01-01');
8 END;
9 /
```

Procedure ADD_INSURANCE_HISTORY compiled
PL/SQL procedure successfully completed.
PL/SQL procedure successfully completed.

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	808 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

The screenshot shows a database table named 'INSURANCEHISTORY' with four columns: POLICYNUM, PLAN_ID, PATIENT_ID, and ACTIVEDATE. The data consists of 16 rows, each representing an insurance plan enrollment record.

	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).

The screenshot shows the SQL Worksheet and Script Output panes of Oracle SQL Developer. The worksheet contains a query that joins multiple tables to find patients enrolled in the 'Empire Plan' who have had an appointment in the last 730 days. The script output shows the results, which are the names of five patients: Aaron Mokowitz, Mel Smith, Carl Hernandez, Harper Nguyen, and Olivia Starz.

```

1 Select distinct p.First_Name || ' ' || p.LAST_NAME as Patient_Name
2 From Patient p
3 inner join InsuranceHistory ih on ih.Patient_ID = p.Patient_ID
4 inner join InsurancePlan ip on ip.Plan_ID = ih.Plan_ID
5 left join Visit v on v.Patient_ID = p.Patient_ID
6 where ip.name = 'Empire Plan'
7 and (SYSTIMESTAMP - v.visit_time) <= '+730 00:00:00.000000';

```

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:

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the tabs are "Start Page", "Relational_1 (Untitled_1)", "final_project~1", "SCHEDULE", and "4_PhysicianSchedules.sql". The "4_PhysicianSchedules.sql" tab is active. Below the tabs, there are two panes: "Worksheet" and "Query Builder". The "Worksheet" pane contains the following PL/SQL code:

```

1 CREATE OR REPLACE PROCEDURE UPDATE_Shift_Start_Add_hour (
2     phys_fname_arg in varchar,
3     phys_lname_arg in varchar
4 )
5 IS
6 BEGIN
7     Update Schedule
8     Set Shift_Start = Shift_Start - INTERVAL '1' HOUR
9     Where Phys_ID in (Select Phys_ID from Physician where First_Name = phys_fname_arg
10                      and Last_Name = phys_lname_arg) and Shift_Start > Systimestamp;
11     --this ensures that schedule entries are only updated in the future and not in the past
12 END;
13
14 BEGIN
15     UPDATE_Shift_Start_Add_hour ('Jay', 'Morris');
16 END;
17 /

```

The line "UPDATE_Shift_Start_Add_hour ('Jay', 'Morris');" is highlighted in blue. Below the code, the status bar says "Task completed in 0.063 seconds". In the bottom left corner of the worksheet pane, it says "Procedure UPDATE_SHIFT_START_ADD_HOUR compiled". In the bottom right corner, it says "PL/SQL procedure successfully completed."

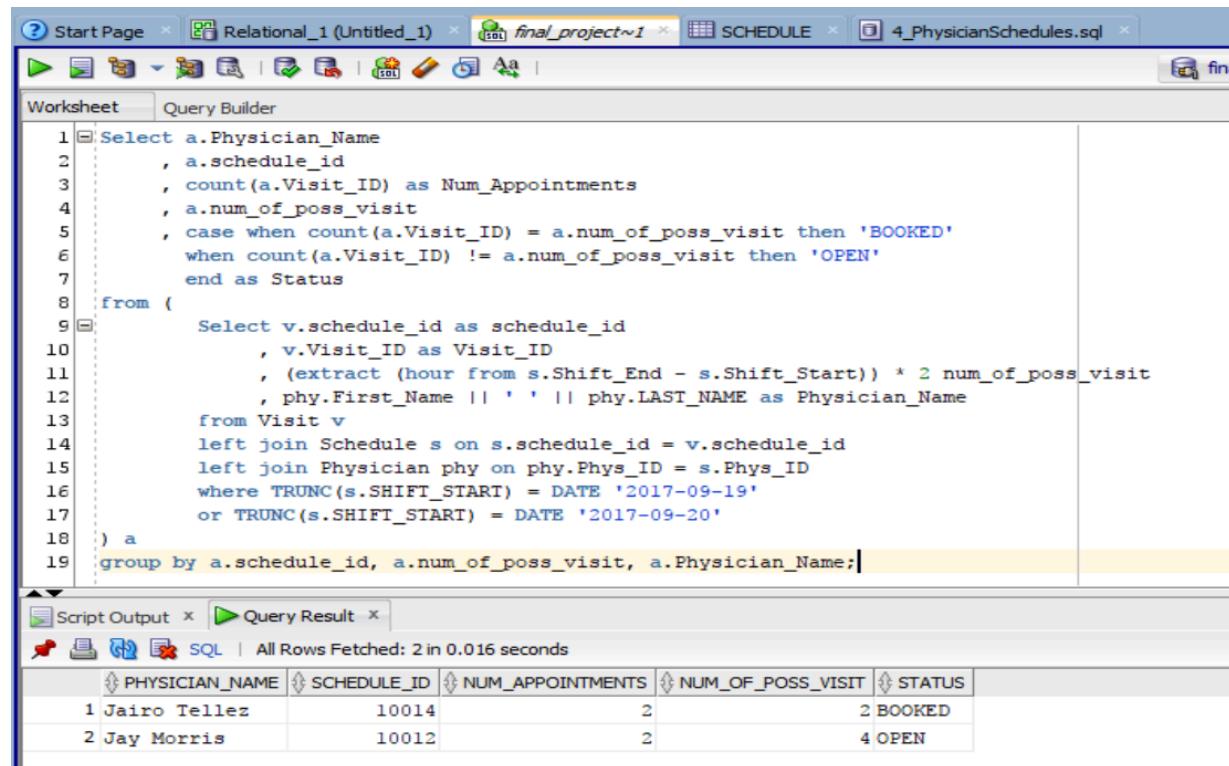
After:

The screenshot shows the Oracle SQL Developer interface with the "SCHEDULE" tab selected in the top navigation bar. Below the tabs, there is a toolbar with icons for columns, data, model, constraints, grants, statistics, triggers, flashback, dependencies, details, partitions, indexes, and SQL. The main area displays a table with the following data:

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.



The screenshot shows a SQL query being run in Oracle SQL Developer. The query selects physician names, schedule IDs, appointment counts, and a status indicator ('BOOKED' or 'OPEN') based on the number of possible visits compared to actual visits. It filters for specific dates (09-19-17 and 09-20-17) and joins with Visit, Schedule, and Physician tables. The results show two physicians: Jairo Tellez and Jay Morris, both with 2 appointments, where Jairo is listed as 'BOOKED' and Jay as 'OPEN'.

```
1 Select a.Physician_Name
2   , a.schedule_id
3   , count(a.Visit_ID) as Num_Appointments
4   , a.num_of_poss_visit
5   , case when count(a.Visit_ID) = a.num_of_poss_visit then 'BOOKED'
6     when count(a.Visit_ID) != a.num_of_poss_visit then 'OPEN'
7   end as Status
8 from (
9   Select v.schedule_id as schedule_id
10  , v.Visit_ID as Visit_ID
11  , (extract (hour from s.Shift_End - s.Shift_Start)) * 2 num_of_poss_visit
12  , phy.First_Name || ' ' || phy.LAST_NAME as Physician_Name
13  from Visit v
14  left join Schedule s on s.schedule_id = v.schedule_id
15  left join Physician phy on phy.Phys_ID = s.Phys_ID
16  where TRUNC(s.SHIFT_START) = DATE '2017-09-19'
17  or TRUNC(s.SHIFT_START) = DATE '2017-09-20'
18 ) a
19 group by a.schedule_id, a.num_of_poss_visit, a.Physician_Name;
```

PHYSICIAN_NAME	SCHEDULE_ID	NUM_APPOINTMENTS	NUM_OF_POSS_VISIT	STATUS
Jairo Tellez	10014	2	2	BOOKED
Jay Morris	10012	2	4	OPEN

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.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Start Page', 'Relational_1 (Untitled_1)', 'final_project~1', and '5_YourOwnAspect.sql'. The status bar at the bottom indicates '0.57700002 seconds'. The main area is a 'Worksheet' tab showing the following SQL code:

```
1 --auto increments the PK in the Patient table
2 Create Sequence seq_schedule
3 MinValue 1
4 Start with 10015
5 Increment by 1
6 Cache 10;
7
8 --need to run this in order to print statement in Script Output below --> dbms_output
9 SET serveroutput on size 30000;
```

The 'Script Output' tab at the bottom shows the result: 'Sequence SEQ_SCHEDULE created.'

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Start Page', 'Relational_1 (Untitled_1)', 'final_project~1', and '5_YourOwnAspect.sql'. The status bar at the bottom indicates '0.078 seconds'. The main area is a 'Worksheet' tab showing the following PL/SQL code:

```
1 CREATE OR REPLACE PROCEDURE ADD_Schedule (
2     phy_lname_arg IN VARCHAR,
3     phy_fname_arg IN VARCHAR,
4     shift_start_arg IN TIMESTAMP,
5     shift_end_arg IN TIMESTAMP
6 )
7 IS
8
9 BEGIN
10    INSERT INTO Schedule (Schedule_ID, Phys_ID, Shift_Start, Shift_End)
11        VALUES (seq_schedule.nextval
12                ,(Select Phys_ID from Physician where Last_Name = phy_lname_arg
13                  and First_Name = phy_fname_arg)
14                ,shift_start_arg
15                ,shift_end_arg
16                );
17
18 END;
```

The 'Script Output' tab at the bottom shows the result: 'Procedure ADD_SCHEDULE compiled.'

The screenshot shows the Oracle SQL Developer interface with the following tabs at the top: Start Page, Relational_1 (Untitled_1), final_project~1, and 5_YourOwnAspect.sql. The main area is a Worksheet tab displaying the following PL/SQL code:

```
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;
```

The code is syntax-highlighted, with the output section (lines 22-23) highlighted in yellow. Below the worksheet, there are two tabs: Script Output and Query Result. The Script Output tab shows the message "Trigger CHECK_WAITLIST_TRG compiled".

The screenshot shows the Oracle SQL Developer interface. The top navigation bar has tabs for "Start Page", "Relational_1 (Untitled_1)", "final_project~1", and "5_YourOwnAspect.sql". The main area is divided into two panes. The left pane, titled "Worksheet", contains the following PL/SQL code:

```
1 --The trigger will alert to contact any patients who have an appointment
2 --after 6/9/17 with Dr. Morris
3 BEGIN
4     Add_Schedule ('Morris', 'Jay', TIMESTAMP '2017-06-09 08:00:00',TIMESTAMP '2017-06-09 15:00:00');
5 END;
6 /
7
8 --The trigger will alert that there isn't a need to check the waitlist as
9 --this doctor either doesn't have a waitlist or their appointments are before the new schedule entry.
10 BEGIN
11     Add_Schedule ('Jones', 'Kathlin', TIMESTAMP '2017-07-09 08:00:00',TIMESTAMP '2017-07-09 15:00:00'
12 END;
13 /
```

The right pane is currently empty. Below the worksheet is a toolbar with icons for running, saving, and zooming, followed by the text "0.18799999 seconds".

The bottom section of the interface is the "Script Output" tab, which displays the following log messages:

```
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.

The screenshot shows the Oracle SQL Developer interface. The Worksheet tab contains the following SQL code:

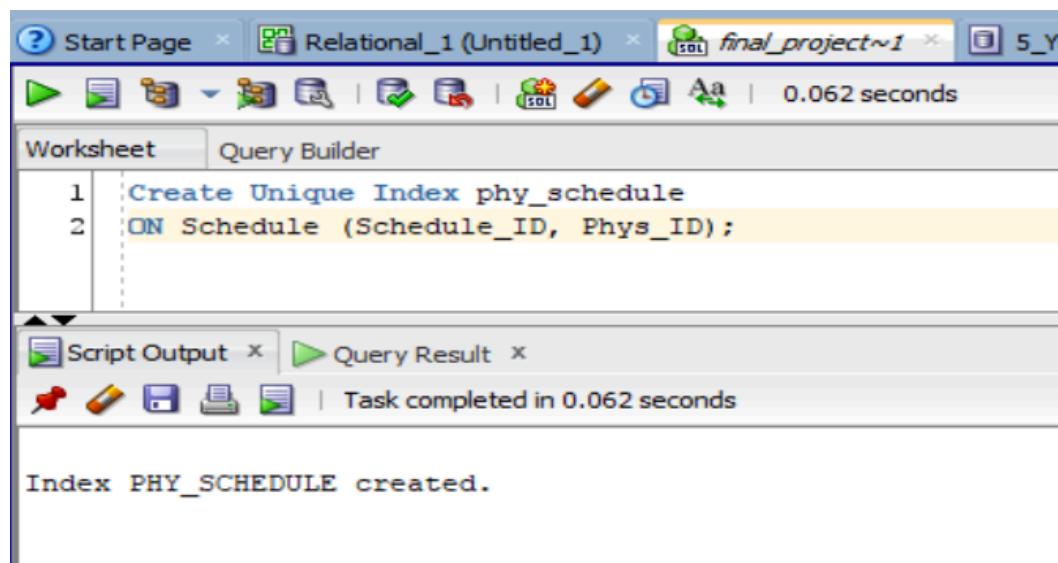
```
1 Select a.SpecialtyName, a.Num_Of_Visits
2 , b.Num_Of_Docs, a.Num_of_Visits/b.Num_of_Docs as Ratio
3 From (
4 Select sp.NAME as SpecialtyName
5 , count (v.visit_id) as Num_of_Visits
6 From Physician p
7 left join Specialty sp on sp.SPEC_ID = p.SPEC_ID
8 left join Schedule s on s.PHYS_ID = p.PHYS_ID
9 left join Visit v on v.SCHEDULE_ID = s.SCHEDULE_ID
10 group by sp.SPEC_ID, sp.NAME) a
11 Inner Join (Select sp.NAME as SpecialtyName, count (p.PHYS_ID) as Num_of_Docs
12 From Physician p
13 left join Specialty sp on sp.SPEC_ID = p.SPEC_ID
14 group by sp.SPEC_ID, sp.NAME
15 ) b ON b.SpecialtyName = a.SpecialtyName;
```

The Query Result tab displays the following data:

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.



The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'Start Page', 'Relational_1 (Untitled_1)', 'final_project~1', and '5_Yo'. Below the menu is a toolbar with various icons. The main workspace has tabs for 'Worksheet' and 'Query Builder'. In the 'Worksheet' tab, two lines of SQL code are visible:

```
1 Create Unique Index phy_schedule  
2 ON Schedule (Schedule_ID, Phys_ID);
```

The second line is highlighted with a yellow background. Below the worksheet is a 'Script Output' tab which displays the message:

```
Index PHY_SCHEDULE created.
```