ABC Health's Product Management System

Rachel Armington
Towson University
Advisor: [redacted]
AIT 735: Case Study in Database Management Systems
Section 101
Fall 2017

Table of Contents

DELIVERABLE 1:

Project Proposal/Statement of Work

Proposal Date: August 31, 2017

AIT Case Study Cover Sheet

Student Name: Rachel Armington

Semester: Fall 2017

Case Study Title: ABC Health's Product Management System

Advisor Name: [redacted]

Course
- ☐ AIT 710
- ☐ AIT 715
- ☐ AIT 720
- ☐ AIT 725
- ☐ AIT 730
- [✓ AIT 735]

Abstract:

ABC Health (ABC), formerly ABC Services, Inc., is an independent, business-to-business health information publishing company that provides relevant news, data, and analysis to customers throughout the United States. The company's researchers, writers, reporters, editors, customer relations representatives, and sales representatives provide products such as newsletters, online subscriptions, directories, and databases to hundreds of executives at health care industry organizations such as health insurers, providers, pharmaceutical companies, and purchasers. ABC has approximately 30 employees working from its sole office location in downtown Washington, D.C.

In late 2016, ABC was acquired by xxx (XXX). Since the acquisition, the company has undergone several transitions – including changes in its product offerings. However, ABC does not have an efficient and organized system to manage its products. Currently, the company utilizes Excel spreadsheets to track each company product, customer, and order. Although ABC is a small company, this management method does not prove very useful or effective. There are frequently duplicate entries, missing entries, and inaccurate data entries. Furthermore, the Excel spreadsheets can only be edited by one user at a time. This project proposes designing and implementing a database system that addresses ABC's need for an effective product management system.

Past Courses: Please list all course completed and currently taking. Remember that you must complete all prerequisites in order to register for a case study.

Number	*Name*	*Semester*	*Grade*
AIT 600	Information Technology Infrastructure	Summer 2016	A
AIT 610	Systems Development Process	Fall 2016	A
AIT 612	Information Systems Vulnerability and Risk Analysis	Fall 2016	A
AIT 632	Database Management Systems	Fall 2016	A
AIT 614	Network Security	Spring 2017	A
AIT 616	Fundamentals of Web Technologies and Development	Spring 2017	A
AIT 732	Advanced Database Management Systems	Spring 2017	A

Client Description

Brief description of the company you've chosen including an overview of the company and what they do, the size of the company (number of employees, customers, locations, etc.), where they operate (local or international) and any other details that you find important for your case study.

ABC Health (ABC), formerly ABC Services, Inc., is an independent health information publishing company located in downtown Washington, D.C. ABC is a business-to-business company that has been operating and serving the health care industry for over 30 years. Its small team of approximately 30 researchers, writers, reporters, editors, customer relations representatives, and sales representatives provides relevant news, data, and analysis to hundreds of customers throughout the United States. The company's customer base primarily consists of executives at health care industry organizations such as health insurers, providers, pharmaceutical companies, and purchasers. Currently, ABC offers about 23 products including online subscriptions, newsletters, directories, and databases.

Company Websites:
https://abchealth.com (main)
https://abchealth.com/marketplace (site to purchase all products)
https://abchealthdata.com (site to access data for subscriber-based products only)

Proposed Project Overview

Describe what you plan on accomplishing throughout the semester.

In late 2016, ABC was acquired by xxx (XXX). Although ABC maintains journalistic independence from XXX, it has undergone several changes since the acquisition. In addition to structural changes and job shifts, ABC has altered its product offerings. However, the company lacks an efficient and organized system to manage its products. Currently, ABC does not track any details or statistics of each of its products. Furthermore, the company utilizes Excel spreadsheets to track each of ABC's customers and orders.

Although ABC is a small company, this management method does not prove very useful or effective. With this method, minimal data is captured, there are frequently duplicate and inaccurate entries, and the Excel spreadsheets can only be edited by one authorized user at a time. This project proposes designing and implementing a database system that addresses ABC's need for an effective product management system.

This project will enable me to apply database management knowledge that I have learned throughout my graduate classes, as well as research and gain knowledge on additional advanced topics. Throughout the semester, I will submit five deliverables, all focused on developing and implementing a database system based on ABC's business needs, goals, and objectives.

Goals

Describe the business need that the company is looking to fulfill with this project.

This project aims to design and implement a database system that addresses ABC's need for an effective product management system. The system will accurately track specifications of each ABC product, information about each customer, and details of each order. It will prevent duplicate entries, conduct data validation checks, and ensure that all required data is recorded.

Additionally, the data will be accessible by multiple users at one time. Employing an effective real time product management system will improve the productivity and organization of the company, as well as strengthen ABC's client relationships by ensuring that each order is entered correctly in the system and packed with the correct products. Overall, implementing such a system could result in increased profits for the company.

Objectives

Describe project milestones that need to be completed in order to call this database system implementation a success.

This project proposes designing and implementing a database system to manage ABC's products. There are five deliverables to this case study. Deliverable 1 – this project proposal/statement of work – contains a description of the client, the goals and objectives of the project, an overview of the proposed project, a proposed timeline detailing the project tasks and deliverables, and a justification of how the project demonstrates graduate coursework.

Deliverable 2 – the database design documents – consists of a system requirements specification (at least 25 detailed system requirements and a list and description of key database entities and attributes); an entity relationship diagram (ERD) that identifies the database entities, attributes, relationships, and primary and foreign keys; and a data dictionary that describes the entities and attributes in the data model.

Deliverable 3 – the database architecture document – consists of the hardware, operating system, and database requirements; a server architecture diagram; a disaster recovery strategy; and a security strategy.

Deliverable 4 – the implementation document – is a working prototype of the database management system. It will contain the tables and relationships depicted in the ERD, configured backups described in the disaster recovery strategy, sample data, and sample users and roles.

Deliverable 5 – the final project submission – is a professional report containing a detailed compilation of the entire project's deliverables. In addition, it involves testing the prototype for each requirement. The final deliverable includes an explanation of how each requirement was solved and documentation proving that the prototype was created.

All five of these deliverables should be completed and submitted for a successful database system implementation.

Proposed Timeline

When you plan on accomplishing the project tasks in conjunction with the project deliverable dates. Your total time spent on this project must total 140 hours minimum.

Week #	Dates	Activities	Estimated Time (hrs)
1	8/28 – 9/1	Complete project proposal	15
2	9/4 – 9/8		
	Friday, Sept 8	Submit Deliverable 1 (project proposal)	-
3	9/11 – 9/15	Complete system requirements specification	15
4	9/18 – 9/22	Complete entity relationship diagram (ERD)	15

5	9/25 – 9/29	Complete data dictionary	10
	Friday, Sept 29	Submit Deliverable 2 (database design documents)	-
6	10/2 – 10/6	Complete hardware, operating system, and database requirements	10
7	10/9 – 10/13	Complete server architecture diagram	15
8	10/16 – 10/20	Complete disaster recovery strategy	20
9	10/23 – 10/27	Complete security strategy	20
	Friday, Oct 27	Submit Deliverable 3 (database architecture document)	-
10	10/30 – 11/3	Complete prototype of database system	40
11	11/6 – 11/10		
12	11/13 – 11/17		
13	11/20 – 11/24		
	Friday, Nov 24	Submit Deliverable 4 (implementation document)	-
14	11/27 – 12/1	Complete final project	15
15	12/4 – 12/8		
16	12/11 – 12/12		
	Tuesday, Dec 12	Submit Deliverable 5 (final project submission)	-
Total Project Hours			**175**

Project Justification

How this project will demonstrate your graduate coursework.

Towson University’s Applied Information Technology: Database Management Systems post-baccalaureate certificate program focuses on designing and developing database systems, and learning how to manage large database systems. The learning objectives presented by this project are extremely relevant to this graduate program, as well as applicable to business needs faced by companies today.

This capstone project will prepare me for future projects that may be assigned to me as a database manager. It incorporates the process of assessing a real-world company’s business needs and designing and implementing a database system based on the company’s goals and objectives. More specifically, it integrates the concepts of the systems development lifecycle (SDLC): database planning, requirements collection and analysis, database design, prototyping and implementation, data conversation and loading, testing, and operational maintenance. These concepts were learned in previous classes such as AIT 610 (System Development Process), AIT 632 (Database Management Systems), and AIT 732 (Advanced Database Management Systems). Not only does this project require me to apply topics learned in my graduate coursework, but it also requires me to research and understand additional advanced topics such as disaster recovery and security strategies. Having a solid (and complete) background knowledge of database management systems will help me in the future when I am designing and implementing databases.

DELIVERABLE 2:

Database Design Documents

Overview

ABC Health (ABC), formerly ABC Services, Inc., is an independent, business-to-business health information publishing company that provides relevant news, data, and analysis to customers throughout the United States. ABC has less than 30 employees working from its sole office location in downtown Washington, D.C. The company's team of researchers, writers, reporters, editors, customer relations representatives, and sales representatives provides products to hundreds of executives at health care industry organizations such as health insurers, providers, pharmaceutical companies, and purchasers. Currently, ABC offers about 23 products including webinars, newsletters, reports, and directories and databases.

In late 2016, ABC was acquired by xxx (XXX). ABC maintains journalistic independence from XXX, but it has undergone several changes since the acquisition. In addition to structural changes and job shifts, ABC has altered its product offerings. However, the company lacks an efficient and organized system to manage its products. Currently, ABC does not track any details or specifications of each of its products. Furthermore, the department directors utilize Excel spreadsheets to track each of the company's customers and orders.

Although ABC is a small organization, this management method does not prove useful or effective. With this method, minimal data is captured, there are frequently duplicate and inaccurate entries, and the Excel spreadsheets can only be edited by one director at a time. This project aims to design and implement a database system that addresses ABC's need for an effective product management system.

Company Websites:
https://abchealth.com (main)
https://abchealth.com/marketplace (all products – purchase via site)
https://abchealthdata.com (subscriber-based products only – purchase and access data via site)

Key Business Needs

The key business needs of ABC Health's Product Management System project are as follows:

1) Provide a system to keep track of ABC's products and the information contained in each.
2) Provide a system to record ABC employee information.
3) Provide a system to house information about each customer.
4) Provide a system to track details of each order.
5) Improve data accuracy by implementing a mechanism that checks data entered.
6) Reduce employee workload by eliminating mistakes that need to be researched and/or corrected.

Project Goals

By implementing ABC Health's Product Management System, the company aims to improve productivity and organization, strengthen client relationships, and increase profits. By implementing a barcode scanner into the system, ordered products will be packed successfully, eliminating wasted company resources.

System Requirements

Functional Requirements

The following table lists the function requirements for ABC Health's Product Management System. These requirements define the various inputs, outputs, and behaviors or functions of the system. Note: additional rows have been added for the final project to provide evidence of implementation (EOI).

Requirement #	Requirement Statement
FR-01	System must store data for the following entities: Department, Employee, Customer, ABCOrder, Publication, Product, OrderProduct
EOI: Each entity has a table in the database consisting of columns to define the data (see Create Tables, Columns, and Relationships). Each table contains sample data (see Insert Sample Data Into Tables). The screenshots after each block of `INSERT` script include a `SELECT * FROM <table name>` statement, which returns all data stored in the respective table.	
FR-02	System must verify all employees before granting any access to the system
EOI: Each employee at ABC Health has a user account (see Users). All users are password-protected, implemented by the `IDENTIFIED BY <password>` parameter. Each user is assigned a role specific to his/her job function at ABC Health (see Roles). All roles are granted the `CREATE SESSION` privilege, which enables users to connect to the database. Therefore, when an employee attempts to login to the system, the user must enter his/her case-sensitive password before s/he can connect to the system.	
FR-03	System must lock user account after a certain number of unsuccessful login attempts
EOI: All users are assigned a profile (`abc_user` for local users and `c##abc_user` for the common user) (see Profiles). The parameters `FAILED_LOGIN_ATTEMPTS` and `PASSWORD_LOCK_TIME` enforce this system requirement. According to the profiles, if a user has 5 unsuccessful login attempts, the user will be locked out of his/her account for 1/144 seconds, or 10 minutes. After this time period is up, the user may attempt to connect to the system again.	
FR-04	System must restrict all employees from logging in to the system outside of normal business hours (9 a.m. to 5 p.m.), with the exception of the employee who conducts system backup and recovery tasks
EOI: The trigger `trg_enforce_normal_business_hours` satisfies this requirement (see Procedures/Triggers/Views FR-04). It sets `restricted_user` as the current user attempting to login and `restricted_time` as any hour before 9 a.m. or after 5 p.m. When a user attempts to connect to the system, the trigger fires and instructs the system to compare the current hour (hour of the system timestamp) to `restricted_time`. If the login attempt occurs during a restricted time, the trigger displays an error message and the user is unable to connect to the system.	
FR-05	System must provide access based on employee's job function (i.e. a manager must have more access than a sales representative)
EOI: The created roles grant privileges to ABC Health employees based on job function (see Roles). For example, an employee assigned the manager role can select from and update the Product table, which stores data such as product price. A manager is granted these privileges	

because it is his/her job duty at ABC Health to rabce and lower product prices according to customer demand (after obtaining approval from top executives). On the other hand, other employees, such as a sales representative, are not granted select and update privileges on the Product table. Although sales representatives use data stored in this table to perform their job functions (e.g. need product prices to calculate customers' order totals), they can perform all duties by executing stored procedures. Creating stored procedures for sales representatives increases the confidentiality of the Product table, as well as helps to enforce the principle of least privilege.	
FR-06	System must allow input from authorized users
EOI: The stored procedures for FR-10, FR-12, FR-13, and FR-14 contain input parameters (see Procedures/Triggers/Views FR-10, Procedures/Triggers/Views FR-12, Procedures/Triggers/Views FR-13, and Procedures/Triggers/Views FR-14). Execute privileges are granted to roles appropriately based on the function of each procedure. For example, the procedure for FR-10 enables employees to create an order product by inputting an order ID, product ID, and order product quantity. This function is performed by sales representatives so the execute privilege for this procedure is granted only to the sales role. Thus, the system will allow input from users authorized to execute these stored procedures. See the screenshots associated with each stored procedure for examples.	
FR-07	System must process and perform validation on user input data
EOI: The stored procedures for FR-10, FR-12, FR-13, and FR-14 process and validate user input (see Procedures/Triggers/Views FR-10, Procedures/Triggers/Views FR-12, Procedures/Triggers/Views FR-13, and Procedures/Triggers/Views FR-14). For example, the procedure for FR-10 enables authorized employees (users assigned the sales role) to create an order product by inputting an order ID, product ID, and order product quantity. The procedure validates the input order ID by ensuring that it is equal to one of the order IDs in the ABCOrder table. If the input does not equal an existing order ID, the system displays the message "`Invalid orderId. Insert failed.`" The procedure validates the input product ID by counting the number of product IDs in the Product table that equal the input product ID. If the count does not equal 1, the system displays the message "`Invalid productId. Insert failed.`" Finally, the trigger `trg_orderProductQty_validate` validates the input order product quantity by ensuring that the input is greater than 0. If the input is less than or equal to 0, the system displays the message "`Invalid orderProductQty. Insert failed.`" All validation code is clearly noted in the SQL scripts.	
FR-08	System must return a feedback message to the user with the success or failure of each action
EOI: The stored procedures for FR-10, FR-12, FR-13, and FR-14 display feedback messages with the success or failure of each action (see Procedures/Triggers/Views FR-10, Procedures/Triggers/Views FR-12, Procedures/Triggers/Views FR-13, and Procedures/Triggers/Views FR-14). Note that the user must first enable system feedback by passing the `set serveroutput on` statement. For example, the procedure for FR-10 displays the following failure messages when an invalid order ID, product ID, or order product quantity, respectively, are input: "`Invalid orderId. Insert failed`," "`Invalid productId. Insert failed`," and "`Invalid orderProductQuantity. Insert failed.`" It also returns the following message when an order product has been added successfully: "`Inserted <row count> rows.`"	

FR-09	System must be able to scan barcodes and handle barcode data
EOI: The barcode scanners are integrated with the wireless mobile devices. The system's database processes data input with a barcode scanner in the same way that it processes data input with a keyboard. Here is an example to explain: an employee in the Production department is packing an order for a customer. S/he enters an order ID then use the wireless device to scan a 2-dimensional barcode on a product before packing it. The barcode decoder software in the device converts the 13-character barcode into numerical text. The text is sent from the device to the database (via MSCA – see NFR-06 under Non-Functional Requirements) and the database performs validation on the product barcode (see FR-12). The system provides feedback to the employee indicating whether the scanned product barcode matches the product ID of an ordered product in the specified order. If so, the scanned product should be packed. In conclusion, the barcode scanning software in the device converts barcode images to text for the database to process and store like it does with all other data. All barcode data is stored in the Product table in the productBarcode column. It has a datatype of `CHAR(13)`.	
FR-10	Given a valid order ID, product ID, and order product quantity, system must be able to create an order product, adding the product price to the order total (via trigger), and adding the order product quantity to the correct product's total quantity sold (via trigger)
EOI: This requirement is satisfied by the procedure `create_orderProduct` and the triggers `trg_orderProductQty_validate`, `trg_insert_orderProduct_validate`, `trg_update_orderTotal`, and `trg_update_productTotalQtySold` (see Procedures/Triggers/Views FR-10). This procedure is only needed by employees in the Sales department. Thus, the privilege to execute this procedure is only granted to the sales role.	
FR-11	System must support a view which lists all orders that have not been completely packed, accessible by employees in the Production department
EOI: This requirement is satisfied by the view `view_unpacked_orders` (see Procedures/Triggers/Views FR-11). This view displays the order ID, order date, order method, order total, and number of products ordered for all unpacked orders. It is granted to the production role so that employees in the Production department can use it to quickly look up orders that need to be packed.	
FR-12	Given a valid order ID and product barcode, system must be able to update the quantity of the order product packed
EOI: This requirement is satisfied by the procedure `update_orderProductQtyPacked` (see Procedures/Triggers/Views FR-12). This procedure is only needed by employees in the Production department to use when packing orders. Thus, the privilege to execute this procedure is only granted to the production role.	
FR-13	Given a valid customer ID, system must be able to list all products purchased and the dates that each was purchased
EOI: This requirement is satisfied by the procedure `products_purchased` (see Procedures/Triggers/Views FR-13). This procedure is only needed by employees in the Sales and Customer Relations departments. Thus, the privilege to execute this procedure is only granted to the sales and custrelations roles.	
FR-14	Given a valid sales representative's employee ID, system must be able to list all products sold and the date that each product was sold
EOI: This requirement is satisfied by the procedure `products_sold` (see	

Procedures/Triggers/Views FR-14). This procedure is only needed by managers to check the sales made by each sales representative. Thus, the privilege to execute this procedure is only granted to the manager role.	
FR-15	System must support a view which lists sales statistics per month, accessible by managers
EOI: This requirement is satisfied by the view `view_sales_stats` (see Procedures/Triggers/Views FR-15). This view displays the year, month, number of orders, number of products ordered, number of products packed, number of customers, the most commonly ordered publication, and the most commonly ordered product. It is grouped so that statistics can be viewed per month. The view is also in descending order with the most recent year and month at the top. This data is used by managers to make business decisions. Thus, this view is only granted to the manager role.	

Non-Functional Requirements

The following table lists the non-functional requirements for ABC Health's Product Management System. These requirements elaborate on the various performance characteristics of the system, such as technical considerations, reliability, usability, maintainability, and security. Note: additional rows have been added for the final project to provide evidence of implementation (EOI).

Requirement #	**Requirement Statement**
NFR-01	System must be available 24/7/365 with scheduled maintenance after normal business hours. In the event of a disaster, system must be recovered and restored in less than 1 hour
EOI: ABC Health's Product Management System is a high availability system due to its clustered database design with shared storage and data redundancy (see High Availability Solution). Additionally, per the backup strategy (see Database Backup Strategy), the database redo logs will be backed up every 30 minutes to increase availability. This is implemented by the `ARCHIVE_LAG_TARGET` parameter, which forces the database to archive redo logs every 1800 seconds, or 30 minutes (see Perform Database Backup). This backup will help to recover the database in the event of a disaster. Furthermore, cumulative incremental backups (`BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE`) will be performed every night at 6 p.m., after normal business hours. Cumulative incremental backups record all changed data blocks since the last level 0 backup. Although differential incremental backups are the default type of incremental backup, if a disaster happens, recovery time is faster with cumulative incremental backups. Thus, performing these backups will help meet the system's Recovery Time Objective (RTO) of 1 hour (see Recovery Time Objective (RTO)).	
NFR-02	System must allow backups to be performed while the database is open
EOI: Running the database in `ARCHIVELOG` mode enables backups to be performed while the database is open (see Enable Archive Mode). Although incremental backups will be performed after normal business hours (to prevent slowing down the system during the day), it is still possible to perform them while the database is open and running in this mode. Furthermore, the system must archive redo logs every 30 minutes to meet the RTO (see Perform Database Backup). In order for this to be done while the database is open, `ARCHIVELOG` mode must be enabled.	

NFR-03	System must ensure that system backups can only be performed by an authorized employee in the IT department
EOI: In order to perform system backup and recovery tasks such as `STARTUP` and `SHUTDOWN`, a user must be granted the `SYSBACKUP` privilege. The user must also be a common user to perform backup and recovery operations on the container database. `c##bgartrell` is the only user that meets both of these criteria. This account belongs to Bob Gartrell, an employee in the IT department. Thus, he will be the only employee authorized to perform all system backup and recovery tasks (see Perform Database Backup).	
NFR-04	System must accommodate 30 users and support 30 concurrent connections
EOI: There are no more than 30 employees at ABC Health and each employee has a user account (see Users). Each user is assigned a profile (`abc_user` for the local users and `c##abc_user` for the common user) (see Profiles). The `SESSIONS_PER_USER` parameter in the profiles enforces this system requirement. According to the profiles, each user can only have 1 concurrent session. Thus, based on this resource limitation, the system can support 30 concurrent connections.	
NFR-05	System must be designed to work on Linux operating systems on the server side. System must be designed to work on Windows operating systems on the client side
EOI: The system utilizes Oracle Database Standard Edition 12c as its database. Oracle is compatible with Linux operating systems. For guidance on installing Oracle 12c onto a Linux server, visit Oracle Help Center's Database Installation Guide (https://docs.oracle.com/database/122/LADBI/toc.htm). Oracle client software such as SQL Plus and SQL Developer are compatible with Windows operating systems. For guidance on installing Oracle 12c clients onto Windows machines, visit Oracle Help Center's Database Client Installation Guide (https://docs.oracle.com/database/122/NTCLI/toc.htm).	
NFR-06	System must enable authorized users to access the database via wireless mobile devices
EOI: ABC Health's Product Management System will incorporate Oracle Mobile Supply Chain Application (MSCA) to enable employees to access the database in real-time from company authorized mobile devices. MSCA requires a java-based mobile server – Mobile Web Application (MWA) server – that supports the Telnet protocol. The MWA server must be installed on both Apache web servers and configured along with the mobile devices. However, due to resource limitations, configuring MSCA is outside the scope of this project. More about configuring MSCA can be found in the Oracle Mobile Supply Chain Applications Implementation Guide (https://docs.oracle.com/cd/E18727_01/doc.121/e13467/title.htm). Once MSCA is properly configured, ABC Health users will be able to access the database via wireless mobile devices. To connect to the database, users must enter their username and password on the device, similar to logging in from a computer terminal.	
NFR-07	System views must display monetary amounts in USD and system must store all monetary amounts to two decimal places to ensure accurate amounts are captured

EOI: The datatype for all currency columns is `NUMBER(7,2)`, which means that the columns can store numbers up to 7 digits long and rounded to 2 decimal places (see empPayRate, orderTotal, and productPrice in the Data Dictionary). Furthermore, the view `view_unpacked_orders` is designed to display the dollar sign for the order total field to clearly illustrate that amounts are in USD (see Procedures/Triggers/Views FR-11).	
NFR-08	System must ensure that the Employee table can only be accessed by employees in the Human Resources department
EOI: The `humanres` role grants object privileges to employees in the Human Resources department (see Roles). Users granted this role may select, insert, or delete from the Employee table. On the other hand, when users without this role attempt to perform any action on the Employee table, the system displays a message stating that the table does not exist, for security reasons. The screenshots under the Roles section prove that only users granted the `humanres` role can access the Employee table.	
NFR-09	System must ensure that the Product table can only be directly updated by managers
EOI: The `manager` role grants object privileges to employees with the manager status (see Roles). Users granted this role may select or delete from the Product table. On the other hand, when users without this role attempt to perform any action on the Product table, the system displays a message stating that the table does not exist, for security reasons.	
NFR-10	System must employ an auditing function that monitors actions performed on secure data objects
EOI: The system will use unified auditing to keep all audit records in one centralized location and to improve system security (see Enable Unified Auditing). The system also implements a unified auditing policy on the Employee object (see Create Object Action Unified Audit Policy). This policy, `all_actions_on_employee_table_pol`, audits all actions, made by all users, on the Employee table. This table contains employees' social security numbers and it is the only object in the database that stores confidential information. (All billing-related matters are conducted through a third party.) The screenshot in the Create Object Action Unified Audit Policy section proves that the unified auditing policy is active and working as intended.	
NFR-11	System must store all archives and backups on the SAN. Storage must be accessible by both database servers

EOI: All logs, archives, and backups will be stored on the SAN, and shared between both database servers (see Local or SAN Attached Storage and File Size, Placement, and Justification). In order for storage to be accessible by both database servers, the SAN must be mounted on both servers. Below is an example of a command to mount a NAS network file system from Oracle (2015):

```
mount nasfiler:/vol/vol1/u01/oracle /u01/oracle -t nfs -o
rw,bg,hard,nointr,tcp,vers=3,timeo=300,rsize=32768,wsize=32768
```

In this instance, the NAS (nasfiler), a network file share type (t), is mounted with the following options (o): read-write (rw), background (bg) retry behavior if the mount attempt fails, mount continues to retry until the server responds (hard), NFS request may not be manually interrupted (nointr), mount using TCP protocol (tcp), mount using NFS protocol version 3 (vers=3), 300 tenths of a second timeout for NFS requests (timeo=300), NFS client requests 32768 bytes from the NFS server in a single read request (rsize=32768), and NFS client sends 32768 bytes to NFS server in a single write request (wsize=32768).

Due to resource limitations, the prototype of this system does not implement the SAN. However, the command above should be modified appropriately to mount the SAN. Note: the directory where the file system is mounted must already exist!
Once the SAN is successfully mounted, all archives and backups should be stored on the SAN. See Set Archive Destination and Perform Database Backup for SQL commands to set the destination.

Entity Relationship Diagram (ERD)

ABC Health has several departments such as Customer Relations, Sales, Directories & Databases, Production, etc. The Department entity will hold the unique ID, name, and code of each department. There can be one or more employees in each department. The relationship between Department and Employee is one to many and the modality is not null, as each department must have an employee.

The Employee entity will store information about each employee at ABC Health. Each employee will have a unique employee ID. The relationship between Employee and Department is one to one because an employee can only work in one (and only one) department. The modality is not null because each employee must be assigned to one department. Furthermore, employees in the Sales department can take zero or more orders from customers. The relationship between Employee and ABCOrder is zero to many because not all sales employees may have taken an order (if those employees have not been let go for making zero sales) and not all employees are in the Sales department. Thus, the modality is null because not all employees take orders.

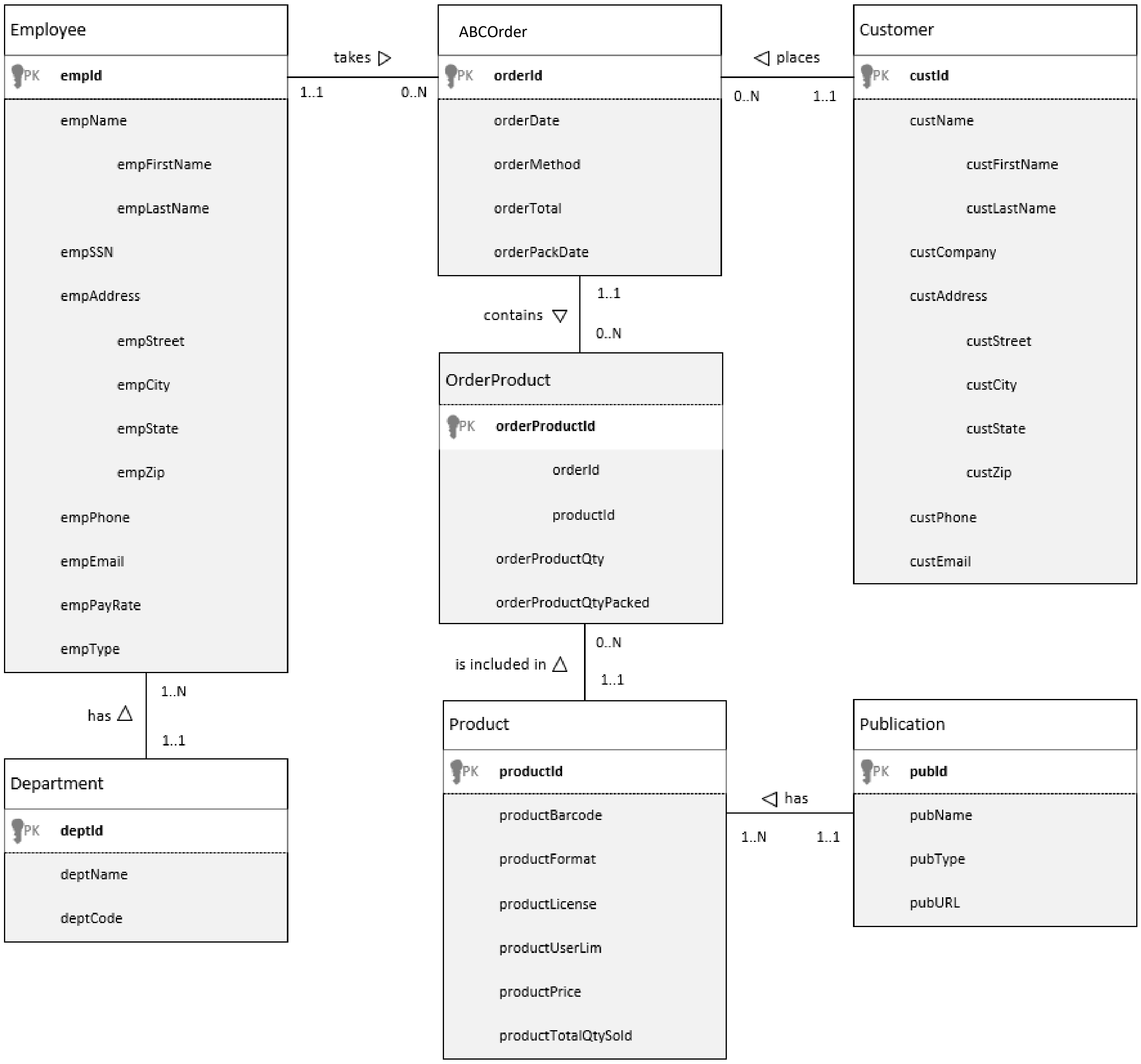
The Customer entity will hold information for each customer who ordered products from ABC Health or who are prospective customers. Each one of these customers can place zero or more orders. The relationship between the Customer and ABCOrder entities is zero to many and the modality is null, as each customer does not have to place an order to be recorded in the database.

The ABCOrder entity will store the information required to keep track of each order placed. Each order will have an order ID, along with an associated employee ID and customer ID. This enables ABC to track which employee worked with which customer. The relationship between the ABCOrder and Employee entities is one to one and the modality is not null, as each order must be taken by one employee. The relationship between the ABCOrder and Customer entities is one to one and the modality is not null, as each order must be placed by one customer. Additionally, each order contains one or more of ABC’s products (ordered products). The relationship between ABCOrder and OrderProduct is zero to many and the modality is null because an order may be started but not yet contain any ordered products.

ABC has several publications such as *ABC’s Directory of Health Plans*. Information about each publication will be stored in the Publication entity. Each publication has one or more products. The relationship between the Publication and Product entities is one to many and the modality is not null because each publication must have at least one product.

The Product entity will store information such as the barcode, format, license type, number of allowed users, price, and total quantity sold of each product. For example, *ABC’s Directory of Health Plans* is published in USB, print, and online subscription formats, along with single or multi-user license types. Each one of these products is assigned a unique product ID. The relationship between the Product and Publication entities is one to one and the modality is not null because each product must be a derivative of one publication. In addition, each product can be ordered by zero or more customers and, thus, can be included in zero or more orders (as an ordered product). The relationship between the Product and OrderProduct entities is zero to many and the modality is null because not every product may be included in an order as an ordered product.

The OrderProduct entity will consist of products contained in each order. There can be multiple products contained in each order. Thus, each ordered product will be uniquely identified by a combination of order ID and product ID, both of which are required attributes; this is a composite primary key. The relationship between the OrderProduct and ABCOrder entities is one to one and the modality is not null because each order product must belong to one order. The relationship between the OrderProduct and Product entities is one to one and the modality is not null because each order product must be one of ABC’s products.


Employee
PK empId
empName
empFirstName
empLastName
empSSN
empAddress
empStreet
empCity
empState
empZip
empPhone
empEmail
empPayRate
empType
takes ▷
1..1
0..N
ABCOrder
PK orderId
orderDate
orderMethod
orderTotal
orderPackDate
◁ places
0..N
1..1
Customer
PK custId
custName
custFirstName
custLastName
custCompany
custAddress
custStreet
custCity
custState
custZip
custPhone
custEmail
contains ▽
1..1
0..N
OrderProduct
PK orderProductId
orderId
productId
orderProductQty
orderProductQtyPacked
is included in △
0..N
1..1
has △
1..N
1..1
Department
PK deptId
deptName
deptCode
Product
PK productId
productBarcode
productFormat
productLicense
productUserLim
productPrice
productTotalQtySold
◁ has
1..N
1..1
Publication
PK pubId
pubName
pubType
pubURL

Data Dictionary

Table/Column Name	Data Type	Data Length	Primary Key	Foreign Key	Identity/ Sequence	Not Null	Unique	Default Value	Valid Values/ Check	Description	Sample
Department											
deptId	int		TRUE		TRUE	TRUE				unique ID of each department	1
deptName	varchar2	200				TRUE	TRUE			department name	Accounting
deptCode	char	4				TRUE			upper(deptCode)	department code	ACCT
Employee											
empId	int		TRUE		TRUE	TRUE				unique ID of each employee	18
empFirstName	varchar2	30				TRUE				employee's first name	Jane
empLastName	varchar2	50				TRUE				employee's last name	Doe
empSSN	char	11				TRUE	TRUE			employee’s social security number (SSN), used for tax purposes	999-99-9999
empStreet	varchar2	150								employee's street address of primary residence	8000 Eastern Dr.
empCity	varchar2	70								employee's city of primary residence	Silver Spring
empState	char	2								employee's state of primary residence	MD
empZip	varchar2	10								employee's zip code of primary residence	20910
empPhone	char	12								employee's main contact phone number	410-775-9208
empEmail	varchar2	250							LIKE '%@%'	employee's main contact email address	jane.doe@gmail.com
empPayRate	number	7,2				TRUE				employee’s pay rate	46500
empType	char	4				TRUE			"full", "part", "temp"	employee's job type	full
deptId	int			TRUE		TRUE				reference Department (deptId)	1
Customer											
custId	int		TRUE		TRUE	TRUE				unique ID of each customer	9999

Table/Column Name	**Data Type**	**Data Length**	**Primary Key**	**Foreign Key**	**Identity/ Sequence**	**Not Null**	**Unique**	**Default Value**	**Valid Values/ Check**	**Description**	**Sample**
custFirstName	varchar2	30				TRUE				customer's first name	Josh
custLastName	varchar2	50				TRUE				customer's last name	Bennett
custCompany	varchar2	200								customer's company	CareFirst BlueCross BlueShield
custStreet	varchar2	150								customer's mailing street address	10455 Mill Run Circle
custCity	varchar2	70								customer's city of mailing address	Owings Mills
custState	char	2								customer's state of mailing address	MD
custZip	varchar2	10								customer's zip code of mailing address	21117
custPhone	char	12								customer's main contact phone number	800-544-8703
custEmail	varchar2	250							LIKE '%@%'	customer's main contact email address	joshbennett@carefirst. com
ABCOrder											
orderId	int		TRUE		TRUE	TRUE				unique ID of each order	28753
orderDate	date					TRUE		sysdate		date order placed	12/15/2016
orderMethod	varchar2	6							"phone", "email", "person", "site"	method of order taken	phone
orderTotal	number	10,2						0	>=0	total amount of customer's order	765.00
orderPackDate	date									date order completely packed	12/16/2016
empId	int			TRUE		TRUE				reference Employee (empId)	18
custId	int			TRUE		TRUE				reference Customer (custId)	9999
Publication											
pubId	int		TRUE		TRUE	TRUE				unique ID of each publication	1
pubName	varchar2	200				TRUE	TRUE			publication name	ABC's Directory of Health Plans
pubType	varchar2	50								publication type (e.g. "webinar", "newsletter",	directory

Table/Column Name	**Data Type**	**Data Length**	**Primary Key**	**Foreign Key**	**Identity/ Sequence**	**Not Null**	**Unique**	**Default Value**	**Valid Values/ Check**	**Description**	**Sample**
										"report", "directory", "database")	
pubURL	varchar2	250								publication URL	https://abchealth.com/ marketplace/abcs-directory-health-plans
Product											
productId	int		TRUE		TRUE	TRUE				unique ID of each product (format-specific publication)	1
productBarcode	char	13				TRUE	TRUE			text contained in the barcode of each product	9781222223334
productFormat	varchar2	9				TRUE			"USB", "print", "onlineSub", "PDF"	product format	print
productLicense	char	1				TRUE			"S", "M". upper(product License)	single or multi-user product license type	S
productUserLim	int					TRUE			>0	number of allowed users per product	1
productPrice	number	20,2								product price	765.00
productTotalQty Sold	int							0		total quantity of each product sold	77
pubId	int			TRUE		TRUE				reference Publication (pubId)	1
OrderProduct											
orderId	int		TRUE	TRUE		TRUE				composite primary key; reference ABCOrder (orderId)	28753
productId	int		TRUE	TRUE		TRUE				composite primary key; reference Product (productId)	1
orderProductQty	int					TRUE				quantity of product ordered	1
orderProductQty Packed	int							0		quantity of product packed	1

References

Oracle. (2015). 3 Using shared storage. Retrieved from https://docs.oracle.com/middleware/1212/core/ASHIA/storage.htm#ASHIA168

DELIVERABLE 3:

Database Architecture Documents

Purpose

The purpose of this document is to provide an architectural overview of ABC Health's Product Management System. It contains information on the database system's hardware, operating system, and database requirements; a server architecture diagram; and the disaster recovery and security strategies.

Hardware Requirements

The following hardware requirements are needed to implement the database for ABC Health's Product Management System:

CPU and Memory

The system will need two processors for each server to add redundancy, with dual cores for each CPU to increase speed and performance. Although a quad-core processor could further improve the speed and performance of the database, it is not necessary for ABC's system. ABC's data environment is relatively small so incorporating a higher core processor is not worth the cost. Furthermore, each database server should have 4 GB of memory. With this amount of memory, the processors will be able to access data faster. Again, since ABC is a small company and its databases are not extremely large, a higher amount of memory is unnecessary and not worth the extra cost. The web servers do not need as much memory as the database servers; they only need about 1 GB of RAM to effectively manage the site traffic.

Disk Space

Since disk space is relatively inexpensive, ABC should have a minimum of 5 TB of disk space. The system requires about 3.5 TB of disk space for the SAN, 600 GB of disk space per database server, and 2 GB of disk space per web server. As an information company in an industry that is continuously changing, ABC's databases are constantly expanding. Thus, the company needs lots of space for its databases, other software, and backups.

RAID level 10 will be implemented on the Oracle servers and on the SAN. This configuration combines disk mirroring and disk striping to provide redundancy and performance, both of which help to protect the data. This level is used because it has the highest read-write performance.

RAID level 5 will be implemented on the web servers. This configuration combines data parity and disk striping to provide fault tolerance and performance. This level does not have as high of a write performance as the RAID level 10 configuration, but it is less expensive than RAID level 10. Only a select number of employees will have write permissions via the web servers so RAID level 10 is not necessary on these servers.

Number of Servers

ABC's Product Management System will employ a database failover cluster, with two servers in the cluster. This database failover cluster will increase availability of the database, which is important to ABC's disaster recovery strategy. In addition, the system requires two web servers (one is a failover) because ABC needs to host its websites (one site for corporate information and one site for publishing its subscriber-based databases). ABC's Product Management System also requires a java-based mobile web application that supports the Telnet protocol – the Oracle

Mobile Web Application (MWA) Server – to implement the mobile barcode scanning function. This must be installed on both web servers. Finally, the system requires one SAN server for the SAN attached storage. Therefore, the system will have a total of five servers.

Physical or Virtual

The system will use a physical server because ABC is currently a small (but growing!) company. Although disaster recovery is a much simpler and faster process with virtual servers, ABC's system only has five servers. With a physical server, ABC would have to pay for bandwidth and software licenses per month.

Implementing a virtualization solution, such as VMware, for five servers may be slightly less costly in terms of the monthly application and hardware licenses, but the upfront costs would be too great for ABC. If ABC had ten or more servers, then the lower monthly cost of implementing a virtual server, as opposed to a physical server, may be worth the upfront cost.

Local or SAN Attached Storage

ABC Health's Product Management System will utilize storage area network (SAN) for several reasons. First, disaster recovery is faster and more reliable with SAN because the SAN will remain accessible if the system's servers go offline. Similarly, the SAN is accessible by both database servers so if the primary server fails, the secondary failover server can access the SAN.

Second, SAN has increased utilization of storage space. SAN allocates space depending on how much space each server needs. For example, if Server 1 is running out of storage space and Server 2 has unused storage space, the SAN can assign more storage space to Server 1. On the other hand, local disk space cannot be changed. With local disks, not only can disk space be wasted, but resources can also be wasted because electricity is required to power these unused disks. This concept does not apply to ABC at this time because one of the servers will be the primary server, while the second server will be a failover server. However, this is important to consider because if ABC expands its business and adds another database server, storage will be allocated accordingly.

Third, SAN attached storage is scalable. There is no limit to SAN storage size, whereas local storage size is limited. If ABC needs to add more storage, it can easily add new disks without even having the reboot the SAN system. Finally, performance is better with SAN attached storage because data is transmitted on its own network, away from all user traffic.

Operating System Requirements

ABC Health's Product Management System will utilize Oracle, which is supported on Windows, UNIX, and Linux operating systems (OS). ABC Health's Product Management System will use Oracle Linux 7.3 on its two database servers because this platform "delivers high transaction performance, enterprise-class reliability, and a cloud-ready infrastructure — all at an extremely low total cost of ownership (TCO)" (Oracle Corporation, 2013).

Red Hat Enterprise Linux 7 (RHEL) will be installed on the Apache web servers. Both Linux and Apache are common and open source. Also, Linux OS has the ability to run a variety of software.

The system will require Windows 7 Professional edition on the client machines. ABC's employees are familiar with the Windows OS and it is compatible with Oracle. Although

Windows 7 is still a widely used operating system among businesses today, it lacks some of the improved features and functionalities of Windows 10. Some of the most important upgrades offered by Windows 10 include VBS (Virtualization-Based Security), a security feature that prevents viruses or malware from modifying certain parts of the operating system, and Windows Hello, which utilizes fingerprint and facial biometric data along with a pin code to grant users access to the operating system and some applications. These added security features would be beneficial to ABC in terms of added security. However, Windows 10 Professional edition licenses for five or more users costs $187 per user, per year. ABC should upgrade to this improved operating system after its business expands.

Database Requirements

DBMS Version

ABC Health's Product Management System will utilize Oracle Database 12c Standard Edition 2, an object-relational database management system. ABC should not implement any of the other editions for the following reasons: Enterprise edition is for large, data processing-intensive environments and it is more expensive than the Standard Edition 2; Express edition (Oracle Database XE) is limited to only using one CPU; and Personal edition supports single-user environments. Thus, ABC should use the Standard Edition 2, which is optimal for small to medium sized businesses. The latest version of Oracle Database Standard Edition 2 is 12.1.0.2.

DBMS Licensing Requirements

Oracle Database Standard Edition 2 can be licensed under the Named User Plus (NUP) metric. According to Oracle's "License Definitions and Rules" document (n.d.), a named user plus can be defined as "an individual authorized by you to use the programs which are installed on a single server or multiple servers, regardless of whether the individual is actively using the programs at any given time"; this can include human-operated devices and non-human operated devices. There is a minimum of 10 NUP licenses per server. Oracle can be licensed on servers with a maximum of 2 sockets, regardless of the number of cores.

Per Oracle Corporations' "Licensing Data Recovery Environments" document (2012), a disaster recovery server does not require a separate license if the following conditions are met: two servers (one primary and one secondary failover) are clustered and have access to a single storage or SAN; if the primary server fails, the secondary server takes over and acts as the primary server; when the primary server is repaired after failure, the primary server takes over from the secondary failover server; and the failover server cannot be used for more than 10 separate days per calendar year (if the failover server is used for maintenance purposes, it still counts toward the 10 day limit).

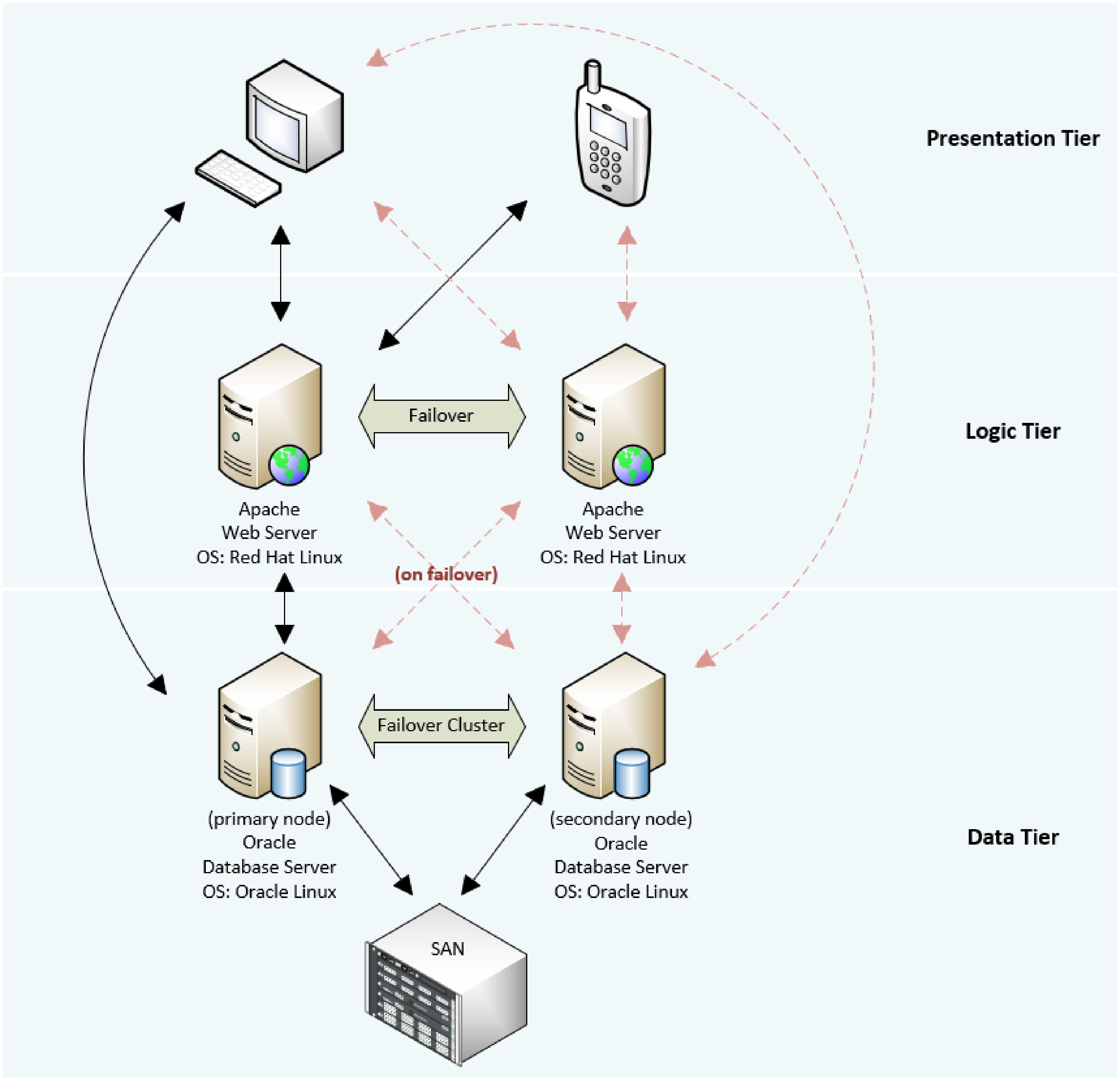
In order to implement ABC Health's Product Management System, ABC needs to obtain a minimum of 10 NUP licenses, per Oracle's licensing requirement. However, ABC should obtain 30 NUP licenses so that each employee has access to the database and/or barcode scanners used for packing orders.

File Size, Placement, and Justification

The system database (Oracle) should be installed on the two database servers, as one server is a failover. The Oracle Database Standard Edition 2 software requires a minimum of 7.5 GB of local disk storage space. Space on the database servers will be reserved for the database software.

Backups of the system database will be kept on the SAN to ensure that this data is available during recovery, if there is a server failure. Furthermore, all database logs will also be placed on the SAN, in a separate file from the actual data (and indexes), to ensure that a full recovery can be performed if there is a hard disk crash. User data will be stored in the SAN, in a separate tablespace from system data in order to increase flexibility in terms of database administration operations, as well as to reduce contention for the same data.

Server Architecture Diagram



Disaster Recovery Strategy

Recovery Point Objective (RPO)

Recovery Point Objective (RPO) is the acceptable point in time from which an organization can recover data. In other words, it defines the acceptable amount of data that an organization can afford to lose in the event of a disaster scenario. ABC's RPO is 30 minutes, which is crucial during normal business hours (between 9 a.m. and 5 p.m.). The system prevents non-management employees from working after normal business hours, unless granted prior authorization, to prevent employees from working overtime. Therefore, it is crucial that the RPO is met during business hours because ABC cannot afford to lose valuable customer information, research results, or order packing data.

Recovery Time Objective (RTO)

Recovery Time Objective (RTO) is the acceptable amount of time that an organization can be without a service. ABC's RTO is no longer than 1 hour. In the event of a disaster scenario, if employees are unable to access the servers, they can still maintain contact with customers via phone. Sales employees, for example, would be able to record phone orders via paper forms and input the information into the system once the server is up and running. Furthermore, employees such as those in the Directories & Databases department would be able to enter data on spreadsheets and import them into the database once the server is restored. Although this is inconvenient and not as productive as if the server were functioning correctly, ABC would still have some limited functionality. Thus, the RTO is less time-sensitive than the RPO since the company would still have some limited functionality if the servers failed.

Database Backup Strategy

Level 0 database backups will be performed every Saturday night (weekly) at 6:00 p.m. Level 0 backups are identical to full backups in that they both back up all used data blocks in the database. However, level 0 backups are considered incremental backups and can be used as a parent for level 1 cumulative incremental backups. These level 0 backups will be performed outside of normal business hours because they use large amounts of resources. Performing backups of the database during peak hours (at 12 p.m., for example) would slow the system down significantly. Thus, these types of backups must be performed after normal business hours so that system performance does not suffer. Level 0 backups will be stored on the SAN.

Cumulative incremental (level 1) backups will be performed every night (daily) at 6:00 p.m. These backups can be performed when the database is open, but like level 0 backups, it is preferable to conduct them outside of normal business hours to avoid slowing down the system. Although cumulative incremental backups are not the default type of incremental backup (differential backups are the default) and they take up more disk space, cumulative backups record all changed data blocks since the last level 0 backup. Thus, recovery time is faster with cumulative incremental backups, which helps to meet the RTO goal. Level 1 backups will be stored on the SAN.

The system will use Oracle's Recovery Manager (RMAN) function to assist with backup and recovery tasks. Since the system must be available 24/7/365, the database must be run in ARCHIVELOG mode to archive redo logs while the database is open. These open data file backups will be performed frequently (every 30 minutes) to meet the RPO. Backing up the log every 30 minutes suffices for ABC because it is a small company; it does not have the high volumes of I/O activity that a large data enterprise would have, nor does it have the resources available to perform log backups every minute. Losing 30 minutes' worth of data would not destroy the company, but it would hinder its productivity levels. All database log backups will be saved to the SAN to ensure a full recovery if a disaster scenario occurs.

ABC's database administrator will also perform backups of the system database after each command that impacts disks, storage, databases, or segments. Backups containing system data will be stored on the SAN to ensure that servers have this information during recovery if a failure occurs.

Finally, full backups will be written to tape and an appropriate backup tape rotation scheme will be implemented. Weekly backups will be held for 12 weeks and then rotated and overwritten with new data. The last tape of every rotation will be permanently archived as a quarterly backup. These tapes will be stored in a secure off-site facility that is located within a two hour radius from ABC's office.

High Availability Solution

ABC Health's Product Management System achieves high availability (and meets its RPO goal) by incorporating a database failover cluster and a SAN. If the primary database server fails, the secondary failover database server will take over. The secondary failover server will be able to access all data as a result of the shared storage SAN.

The system also achieves high availability by incorporating data redundancy. This is provided through RAID configurations (RAID 10 for the database servers and SAN, and RAID 5 for the web servers). Data redundancy is further provided through the backup process. Data will consistently be backed up (every 30 minutes, daily, and weekly) and stored on the SAN, not on the server. Furthermore, there will be multiple copies of the backups (disk and tape).

The hardware and tapes must be tested continuously to ensure that all equipment is functioning correctly.

Security Strategy

Type of Authentication

There are several pre-existing authentication security measures in place to protect the data stored in ABC Health's Product Management System. First, there is a physical access restriction. Employees need a swipe card to enter the building and a key to enter ABC's front office door (the back door is always locked from the outside). Second, to access the operating system and ABC's network on any machine, users must log in with a registered username and password.

The system will implement several other authentication security functions. Oracle Database will authenticate users trying to access ABC Health's Product Management System by utilizing usernames and passwords stored in the data dictionary of the database. The system will also utilize Oracle's account locking feature to enhance security. After 5 consecutive unsuccessful login attempts, the system will lock the user's account for 10 minutes. This will provide an obstacle for attackers if they attempt to crack a user's password.

Furthermore, the system will utilize Oracle roles and system permissions to authenticate users. All employees will only be permitted to log into the system between normal business hours (9 a.m. to 5 p.m.), with the exception of managers and employees who have received prior approval to work outside of normal business hours. The system will use directory permissions to only allow employees in the Directories & Databases department to write to the database via the web browser application.

Additionally, the system will enforce object privileges to restrict table access to certain users. In other words, ABC employees will be granted access (or permissions) to certain information based on their job function. For example, employees in the Human Resources (HR) department will have access to the Employee data table, whereas employees in the Directories & Databases department will not. This authentication will be implemented to protect employees' confidential information such as address and pay rate.

Auditing Requirements

The Oracle database will use unified auditing to improve system security. Unified auditing will capture audit records from a variety of sources – including Oracle Recovery Manager audit records, SYS audit records, and audit records from unified audit policies – to improve the security of the database. The only confidential information stored in the system is the social security number of each of ABC's 30 employees. Therefore, an object action unified audit policy will be created to audit all actions on the Employee table by all employees in the HR department. (Employees in the HR department will be the only users with permissions to access this table.)

Encryption Requirements

Oracle Database uses the Advanced Encryption Standard (AES) to automatically encrypt passwords during network connections. This encryption function will help to secure users' credentials during the login process and improve ABC's confidentiality.

There are no other encryption requirements for ABC Health's Product Management system. ABC has a low risk of cyber-attack because the financial incentive is low. The organization is a small publishing company and most of its information is procured from publicly available sources. Furthermore, ABC does not store customers' payment information (e.g. credit card numbers) because it uses a third party to conduct its billing. The only confidential information stored in the system is the social security number of each of ABC's 30 employees. Employees' information will be kept secure by implementing permissions to authenticate users, along with auditing policies to view actions on the Employee table. Thus, there is no requirement to encrypt data stored in the database.

References

Oracle. (n.d.). License definitions and rules. Retrieved from http://www.oracle.com/us/corporate/pricing/olsadef-gr-v020703-070599.pdf

Oracle Corporation. (2012). Licensing data recovery environments. Retrieved from http://www.oracle.com/us/corporate/pricing/data-recovery-licensing-070587.pdf

Oracle Corporation. (2013). *What makes Oracle Linux the best platform for Oracle Database 12c*. [White paper]. Retrieved from http://www.oracle.com/us/technologies/linux/linux-for-oracle-database-wp-2068570.pdf

DELIVERABLE 4:

Implementation Documents

Purpose

The purpose of this deliverable is to document a working prototype of ABC Health's Product Management System database. The prototype includes all tables, columns, and relationships contained in the ERD from Deliverable 2; sample data; sample profiles, users, and roles; procedures/triggers (as stated in the System Requirements from Deliverable 2); configured backups (as described in the Database Backup Strategy from Deliverable 3); and configured audits (as described in the Auditing Requirements from Deliverable 3). This deliverable contains all of the SQL scripts used to create the prototype in Oracle Database Standard Edition 2, as well as screenshots to document the implementation process. The Oracle Help Center is a helpful source to reference while conducting database administration tasks in Oracle Database 12c (https://docs.oracle.com/database/122/ADMQS/introduction.htm#ADMQS001).

Overview

A multitenant architecture option was introduced with Oracle Database 12c. This option enables an Oracle multitenant container database (CDB) to host zero or more pluggable databases (PDBs), or user-created portable collections of schemas and objects (Oracle, 2017). A large company, for instance, may have a CDB consisting of multiple PDBs, one for each department-specific application.

ABC Health's Product Management System utilizes the multitenant architecture in single tenant configuration. In other words, this prototype has one PDB (orclpdb) per CDB (abcorcl). It is important to note that hosting more than one PDB requires a license. Thus, since ABC is a small company, there is currently no need for more than one PDB. However, this prototype is designed under the assumption that the company will expand in the future. With this multitenant architecture, ABC will be able to quickly add more PDBs in the future as the business needs increase.

Method Note: All scripts were created and run using the Oracle SQL Developer client application, unless otherwise specified.

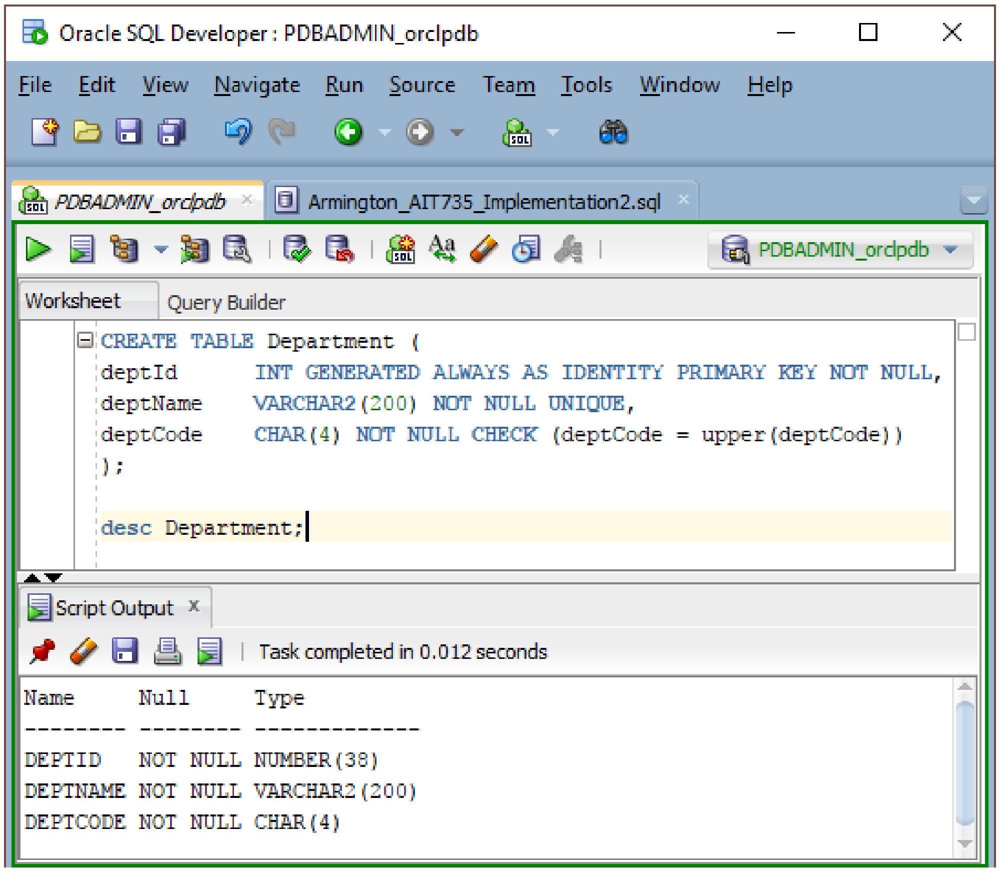
Create Tables, Columns, and Relationships

The following SQL code was used to create the tables, columns, and relationships in Oracle, as illustrated in the ERD. Relationships are created using primary key and foreign key constraints for each table accordingly. The code also includes all factors identified in the Data Dictionary (from Deliverable 2) such as data type, data length, `UNIQUE` and `NOT NULL` constraints, etc.

Method Note: All table objects were created under the `PDBADMIN` schema in the PDB orclpdb.

Department Table

```
CREATE TABLE Department (
deptId          INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
deptName        VARCHAR2(200) NOT NULL UNIQUE,
deptCode        CHAR(4) NOT NULL CHECK (deptCode = upper(deptCode))
);
```

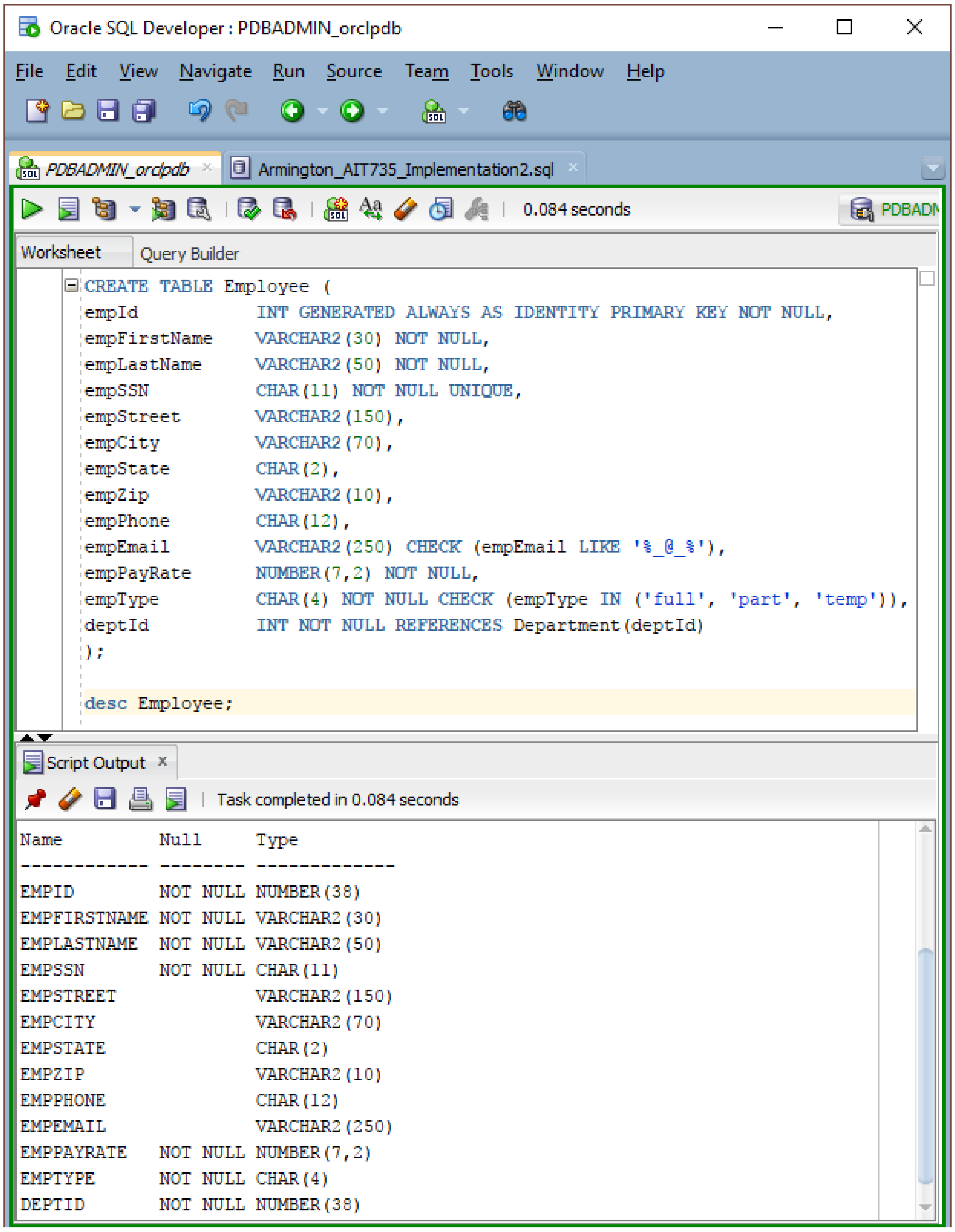

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.012 seconds

```
CREATE TABLE Department (
deptId       INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
deptName     VARCHAR2(200) NOT NULL UNIQUE,
deptCode     CHAR(4) NOT NULL CHECK (deptCode = upper(deptCode))
);

desc Department;
```

```
Name     Null     Type
-------- -------- -------------
DEPTID   NOT NULL NUMBER(38)
DEPTNAME NOT NULL VARCHAR2(200)
DEPTCODE NOT NULL CHAR(4)
```

Employee Table

```
CREATE TABLE Employee (
empId           INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
empFirstName    VARCHAR2(30) NOT NULL,
empLastName     VARCHAR2(50) NOT NULL,
empSSN          CHAR(11) NOT NULL UNIQUE,
empStreet       VARCHAR2(150),
empCity         VARCHAR2(70),
empState        CHAR(2),
empZip          VARCHAR2(10),
empPhone        CHAR(12),
empEmail        VARCHAR2(250) CHECK (empEmail LIKE '%_@_%'),
empPayRate      NUMBER(7,2) NOT NULL,
empType         CHAR(4) NOT NULL CHECK (empType IN ('full', 'part',
'temp')),
deptId          INT NOT NULL REFERENCES Department(deptId)
);
```



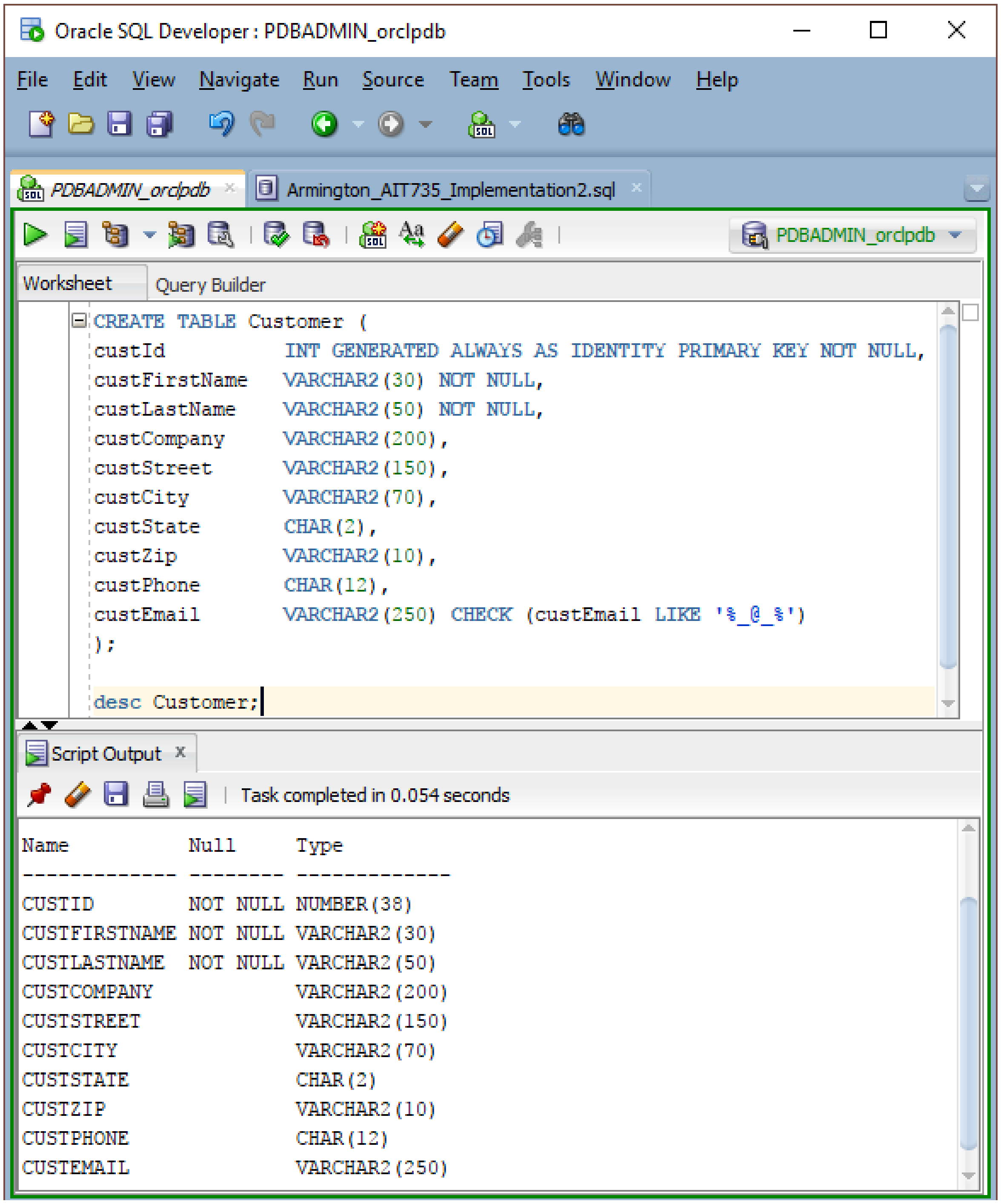
Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
0.084 seconds
Worksheet
Query Builder
Script Output
Task completed in 0.084 seconds

```
CREATE TABLE Employee (
empId               INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
empFirstName        VARCHAR2(30) NOT NULL,
empLastName         VARCHAR2(50) NOT NULL,
empSSN              CHAR(11) NOT NULL UNIQUE,
empStreet           VARCHAR2(150),
empCity             VARCHAR2(70),
empState            CHAR(2),
empZip              VARCHAR2(10),
empPhone            CHAR(12),
empEmail            VARCHAR2(250) CHECK (empEmail LIKE '%_@_%'),
empPayRate          NUMBER(7,2) NOT NULL,
empType             CHAR(4) NOT NULL CHECK (empType IN ('full', 'part', 'temp')),
deptId              INT NOT NULL REFERENCES Department(deptId)
);

desc Employee;
```

```
Name         Null     Type
------------ -------- -------------
EMPID        NOT NULL NUMBER(38)
EMPFIRSTNAME NOT NULL VARCHAR2(30)
EMPLASTNAME  NOT NULL VARCHAR2(50)
EMPSSN       NOT NULL CHAR(11)
EMPSTREET             VARCHAR2(150)
EMPCITY               VARCHAR2(70)
EMPSTATE              CHAR(2)
EMPZIP                VARCHAR2(10)
EMPPHONE              CHAR(12)
EMPEMAIL              VARCHAR2(250)
EMPPAYRATE   NOT NULL NUMBER(7,2)
EMPTYPE      NOT NULL CHAR(4)
DEPTID       NOT NULL NUMBER(38)
```

Customer Table

```
CREATE TABLE Customer (
custId          INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
custFirstName   VARCHAR2(30) NOT NULL,
custLastName    VARCHAR2(50) NOT NULL,
custCompany     VARCHAR2(200),
custStreet      VARCHAR2(150),
custCity        VARCHAR2(70),
custState       CHAR(2),
custZip         VARCHAR2(10),
custPhone       CHAR(12),
custEmail       VARCHAR2(250) CHECK (custEmail LIKE '%_@_%')
);
```


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Task completed in 0.054 seconds

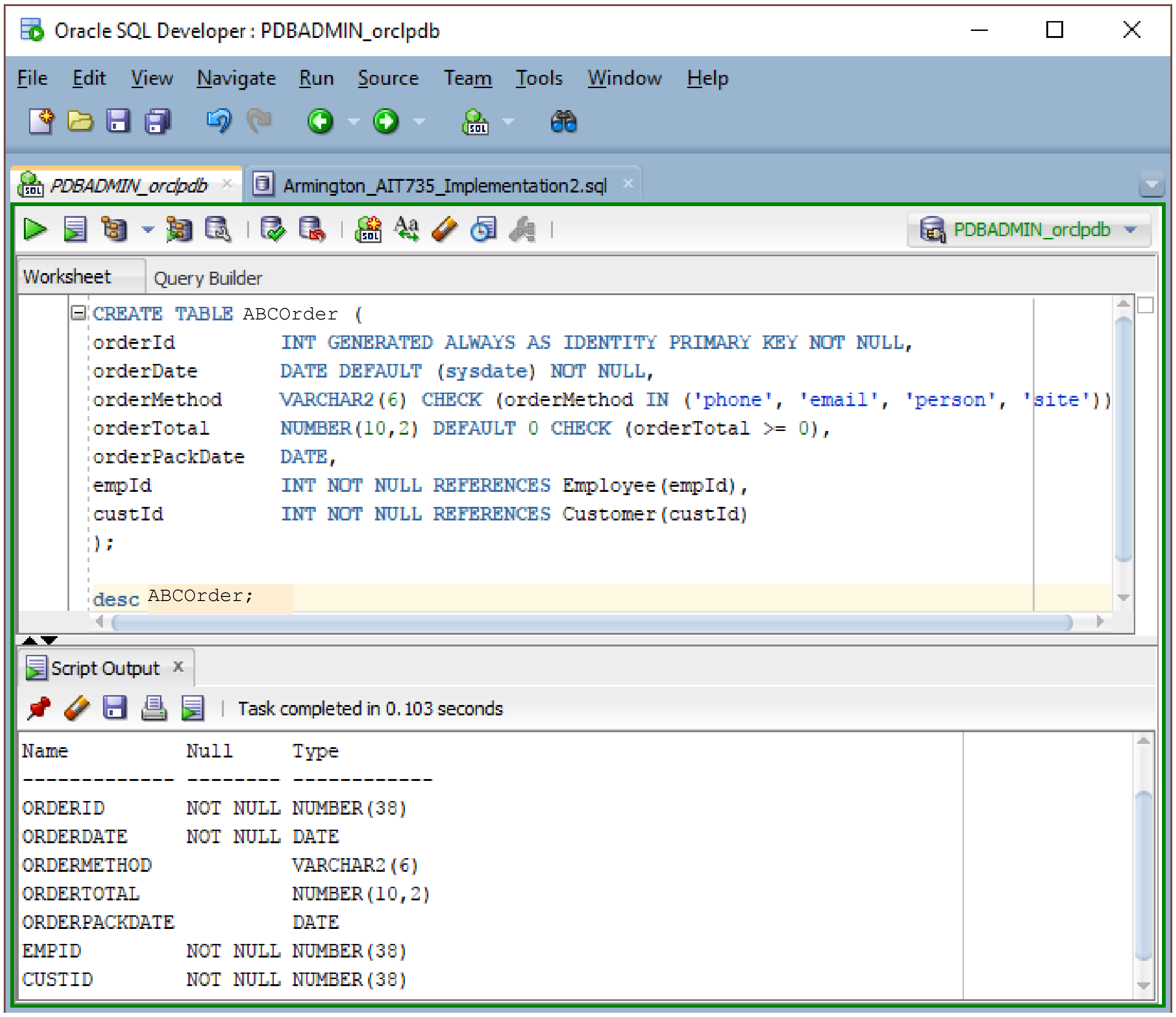
```
CREATE TABLE Customer (
custId          INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
custFirstName   VARCHAR2(30) NOT NULL,
custLastName    VARCHAR2(50) NOT NULL,
custCompany     VARCHAR2(200),
custStreet      VARCHAR2(150),
custCity        VARCHAR2(70),
custState       CHAR(2),
custZip         VARCHAR2(10),
custPhone       CHAR(12),
custEmail       VARCHAR2(250) CHECK (custEmail LIKE '%_@_%')
);

desc Customer;
```

```
Name          Null     Type
------------- -------- -------------
CUSTID        NOT NULL NUMBER(38)
CUSTFIRSTNAME NOT NULL VARCHAR2(30)
CUSTLASTNAME  NOT NULL VARCHAR2(50)
CUSTCOMPANY            VARCHAR2(200)
CUSTSTREET             VARCHAR2(150)
CUSTCITY               VARCHAR2(70)
CUSTSTATE              CHAR(2)
CUSTZIP                VARCHAR2(10)
CUSTPHONE              CHAR(12)
CUSTEMAIL              VARCHAR2(250)
```

ABCOrder Table

The `TO_DATE` function is used to insert dates into the orderDate and orderPackDate columns.

```
CREATE TABLE ABCOrder (
orderId          INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
orderDate        DATE DEFAULT (sysdate) NOT NULL,
orderMethod      VARCHAR2(6) CHECK (orderMethod IN ('phone', 'email',
'person', 'site')),
orderTotal       NUMBER(10,2) DEFAULT 0 CHECK (orderTotal >= 0),
orderPackDate    DATE,
empId            INT NOT NULL REFERENCES Employee(empId),
custId           INT NOT NULL REFERENCES Customer(custId)
);
```


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output

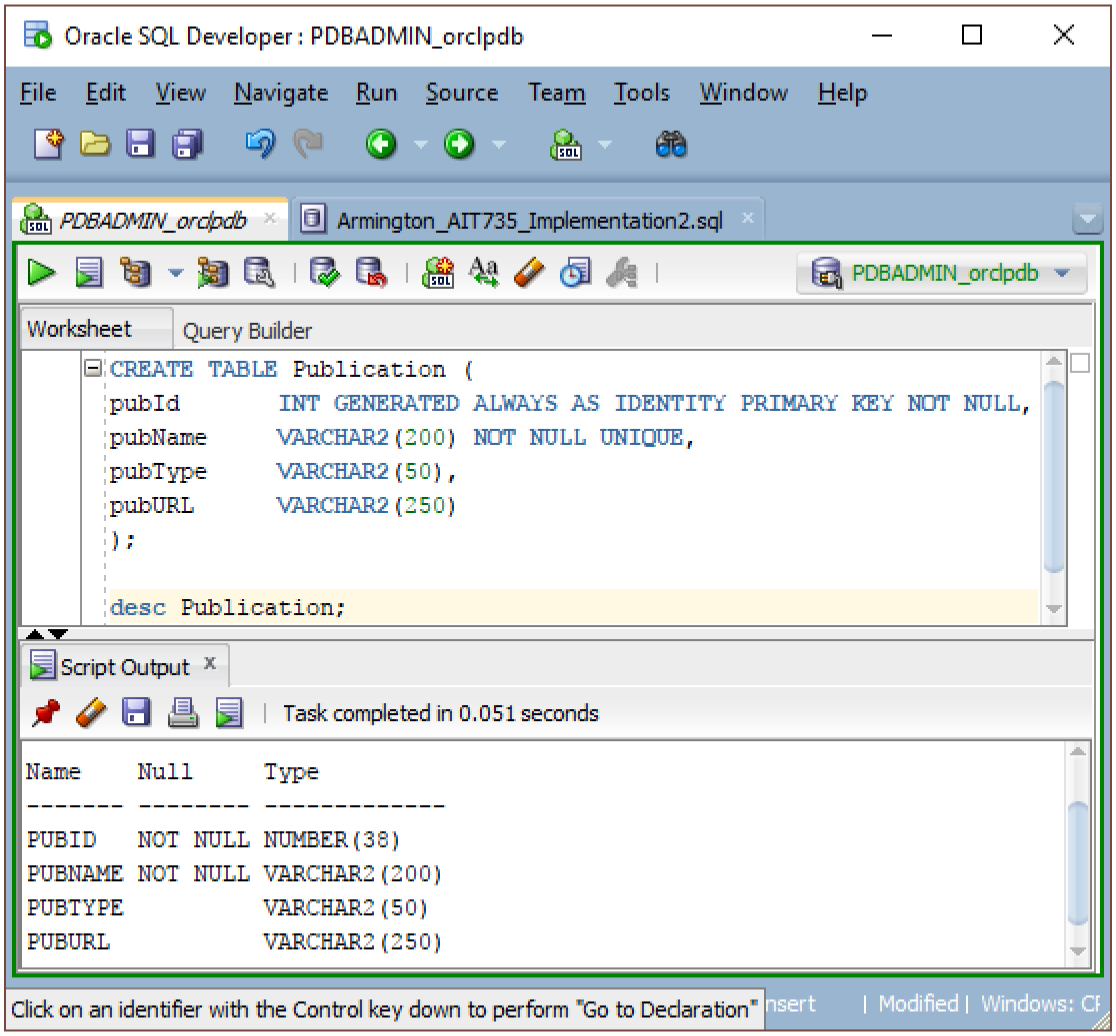
```
CREATE TABLE ABCOrder (
orderId          INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
orderDate        DATE DEFAULT (sysdate) NOT NULL,
orderMethod      VARCHAR2(6) CHECK (orderMethod IN ('phone', 'email', 'person', 'site'))
orderTotal       NUMBER(10,2) DEFAULT 0 CHECK (orderTotal >= 0),
orderPackDate    DATE,
empId            INT NOT NULL REFERENCES Employee(empId),
custId           INT NOT NULL REFERENCES Customer(custId)
);

desc ABCOrder;
```

Task completed in 0.103 seconds

```
Name          Null     Type
------------- -------- ------------
ORDERID       NOT NULL NUMBER(38)
ORDERDATE     NOT NULL DATE
ORDERMETHOD            VARCHAR2(6)
ORDERTOTAL             NUMBER(10,2)
ORDERPACKDATE          DATE
EMPID         NOT NULL NUMBER(38)
CUSTID        NOT NULL NUMBER(38)
```

Publication Table

```
CREATE TABLE Publication (
pubId            INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
pubName          VARCHAR2(200) NOT NULL UNIQUE,
pubType          VARCHAR2(50),
pubURL           VARCHAR2(250)
);
```

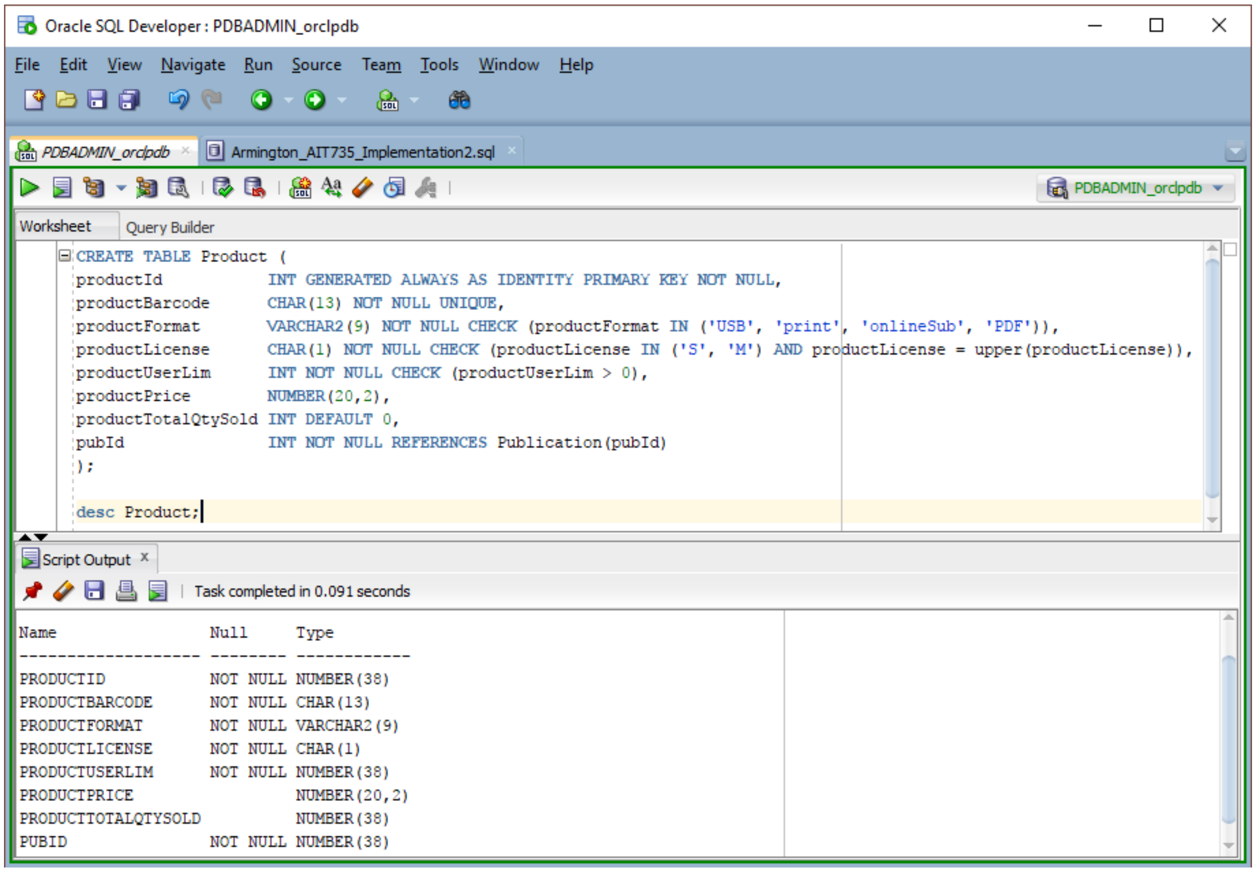

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Task completed in 0.051 seconds
Click on an identifier with the Control key down to perform "Go to Declaration"
nsert | Modified | Windows: CI

```
CREATE TABLE Publication (
pubId         INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
pubName       VARCHAR2(200) NOT NULL UNIQUE,
pubType       VARCHAR2(50),
pubURL        VARCHAR2(250)
);

desc Publication;
```

```
Name     Null     Type
-------- -------- -------------
PUBID    NOT NULL NUMBER(38)
PUBNAME  NOT NULL VARCHAR2(200)
PUBTYPE           VARCHAR2(50)
PUBURL            VARCHAR2(250)
```

Product Table

```
CREATE TABLE Product (
productId              INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT
NULL,
productBarcode         CHAR(13) NOT NULL UNIQUE,
productFormat          VARCHAR2(9) NOT NULL CHECK (productFormat IN
('USB', 'print', 'onlineSub', 'PDF')),
productLicense         CHAR(1) NOT NULL CHECK (productLicense IN ('S',
'M') AND productLicense = upper(productLicense)),
productUserLim         INT NOT NULL CHECK (productUserLim > 0),
productPrice           NUMBER(20,2),
productTotalQtySold    INT DEFAULT 0,
pubId                  INT NOT NULL REFERENCES Publication(pubId)
);
```

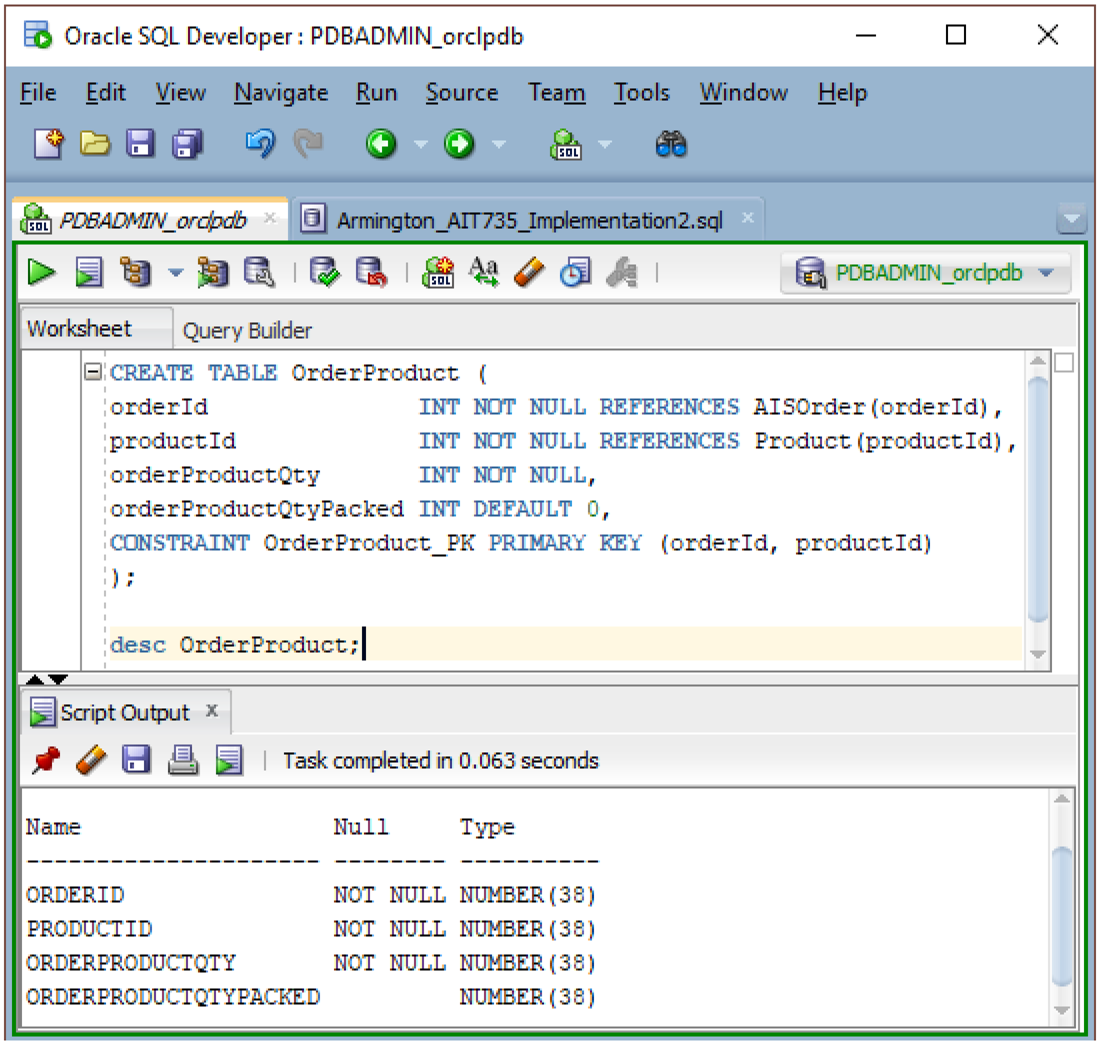

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.091 seconds

```
CREATE TABLE Product (
productId            INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL,
productBarcode       CHAR(13) NOT NULL UNIQUE,
productFormat        VARCHAR2(9) NOT NULL CHECK (productFormat IN ('USB', 'print', 'onlineSub', 'PDF')),
productLicense       CHAR(1) NOT NULL CHECK (productLicense IN ('S', 'M') AND productLicense = upper(productLicense)),
productUserLim       INT NOT NULL CHECK (productUserLim > 0),
productPrice         NUMBER(20,2),
productTotalQtySold INT DEFAULT 0,
pubId                INT NOT NULL REFERENCES Publication(pubId)
);

desc Product;
```

```
Name                Null     Type
------------------- -------- ------------
PRODUCTID           NOT NULL NUMBER(38)
PRODUCTBARCODE      NOT NULL CHAR(13)
PRODUCTFORMAT       NOT NULL VARCHAR2(9)
PRODUCTLICENSE      NOT NULL CHAR(1)
PRODUCTUSERLIM      NOT NULL NUMBER(38)
PRODUCTPRICE                 NUMBER(20,2)
PRODUCTTOTALQTYSOLD          NUMBER(38)
PUBID               NOT NULL NUMBER(38)
```

Order Product Table

```
CREATE TABLE OrderProduct (
orderId                INT NOT NULL REFERENCES ABCOrder(orderId),
productId              INT NOT NULL REFERENCES Product(productId),
orderProductQty        INT NOT NULL,
orderProductQtyPacked INT DEFAULT 0,
CONSTRAINT OrderProduct_PK PRIMARY KEY (orderId, productId)
);
```


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.063 seconds

```
CREATE TABLE OrderProduct (
orderId              INT NOT NULL REFERENCES AISOrder(orderId),
productId            INT NOT NULL REFERENCES Product(productId),
orderProductQty      INT NOT NULL,
orderProductQtyPacked INT DEFAULT 0,
CONSTRAINT OrderProduct_PK PRIMARY KEY (orderId, productId)
);

desc OrderProduct;
```

```
Name                  Null     Type
--------------------- -------- ----------
ORDERID               NOT NULL NUMBER(38)
PRODUCTID             NOT NULL NUMBER(38)
ORDERPRODUCTQTY       NOT NULL NUMBER(38)
ORDERPRODUCTQTYPACKED          NUMBER(38)
```

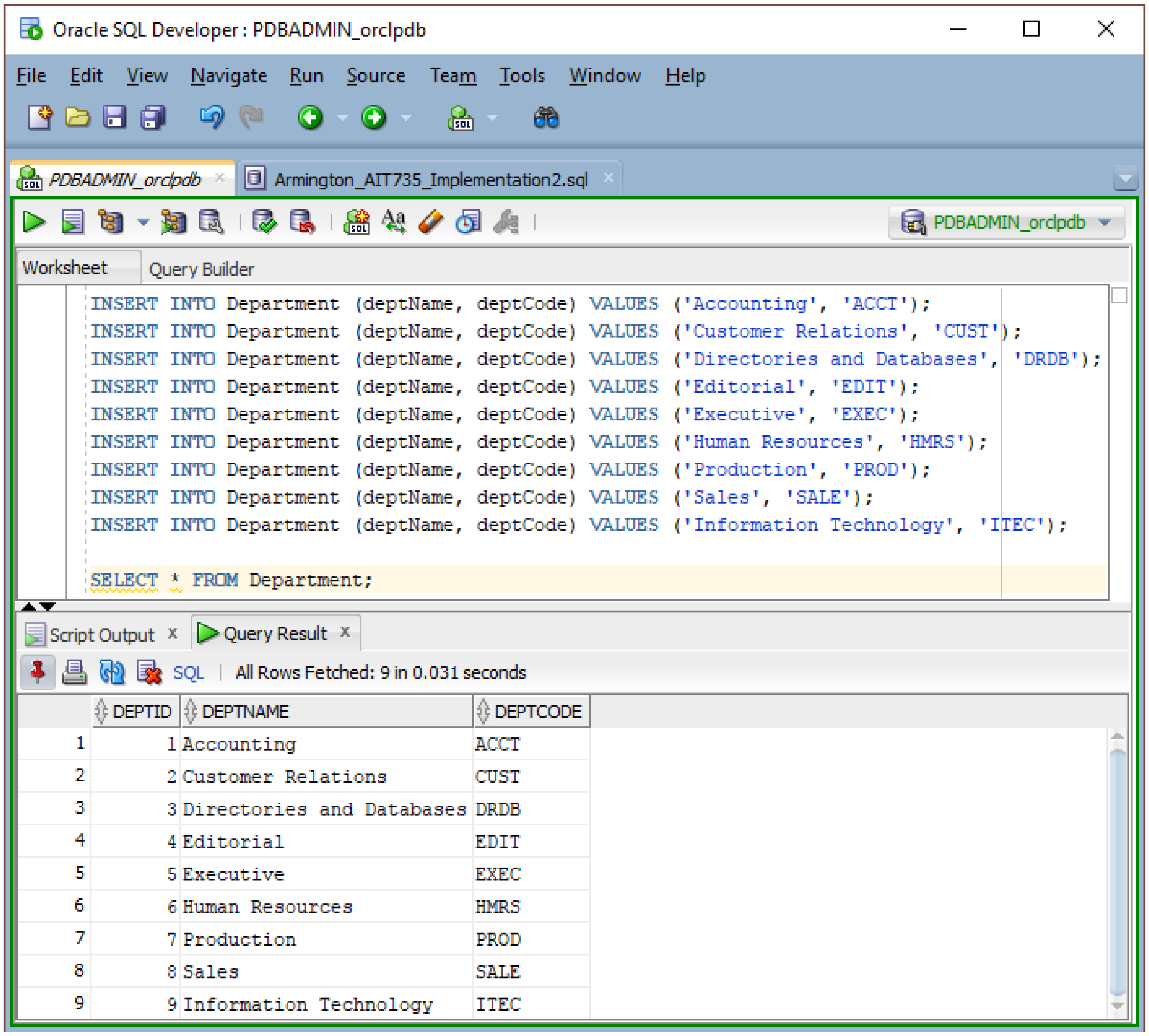
Insert Sample Data Into Tables

The following SQL code was used to insert sample data into the tables.

Department Table

```
INSERT INTO Department (deptName, deptCode) VALUES ('Accounting',
      'ACCT');
INSERT INTO Department (deptName, deptCode) VALUES ('Customer
      Relations', 'CUST');
INSERT INTO Department (deptName, deptCode) VALUES ('Directories and
      Databases', 'DRDB');
INSERT INTO Department (deptName, deptCode) VALUES ('Editorial',
      'EDIT');
INSERT INTO Department (deptName, deptCode) VALUES ('Executive',
      'EXEC');
INSERT INTO Department (deptName, deptCode) VALUES ('Human Resources',
      'HMRS');
INSERT INTO Department (deptName, deptCode) VALUES ('Production',
      'PROD');
```

```
INSERT INTO Department (deptName, deptCode) VALUES ('Sales', 'SALE');
INSERT INTO Department (deptName, deptCode) VALUES ('Information
      Technology', 'ITEC');
```

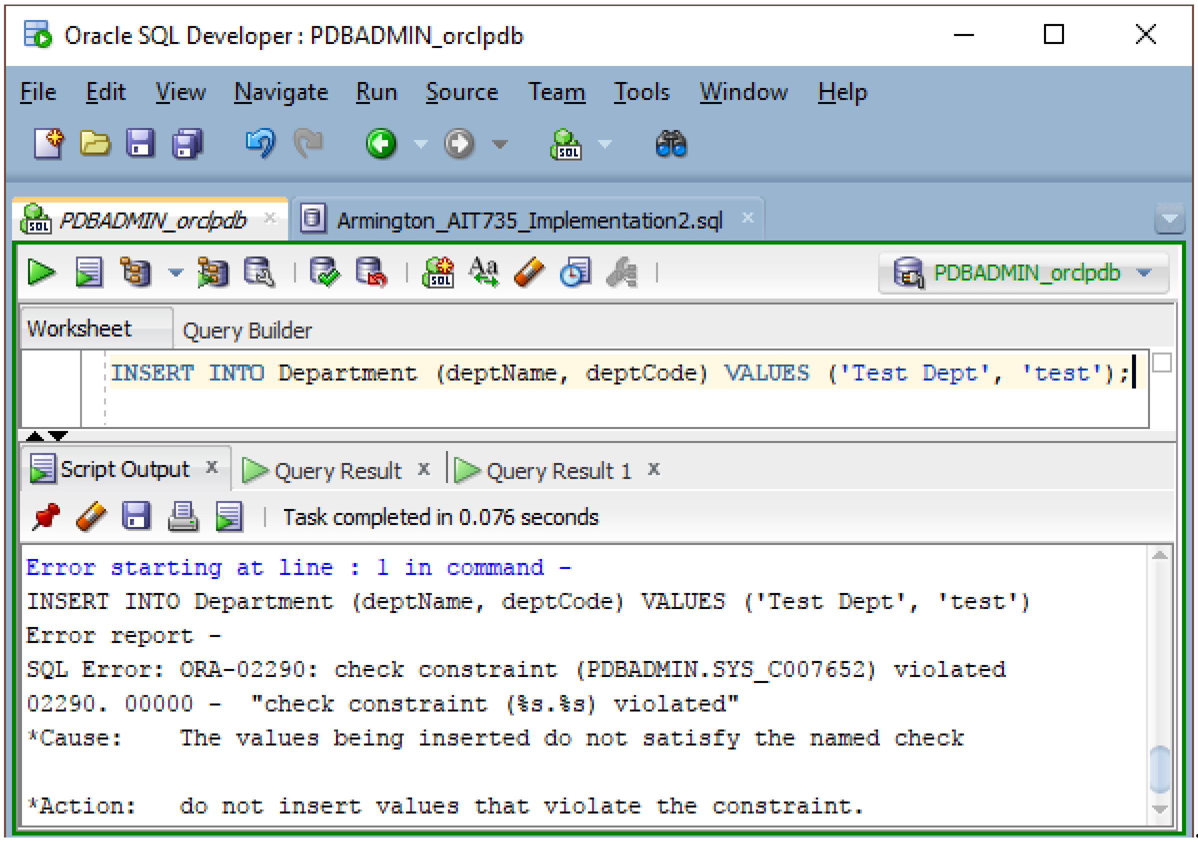

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO Department (deptName, deptCode) VALUES ('Accounting', 'ACCT');
INSERT INTO Department (deptName, deptCode) VALUES ('Customer Relations', 'CUST');
INSERT INTO Department (deptName, deptCode) VALUES ('Directories and Databases', 'DRDB');
INSERT INTO Department (deptName, deptCode) VALUES ('Editorial', 'EDIT');
INSERT INTO Department (deptName, deptCode) VALUES ('Executive', 'EXEC');
INSERT INTO Department (deptName, deptCode) VALUES ('Human Resources', 'HMRS');
INSERT INTO Department (deptName, deptCode) VALUES ('Production', 'PROD');
INSERT INTO Department (deptName, deptCode) VALUES ('Sales', 'SALE');
INSERT INTO Department (deptName, deptCode) VALUES ('Information Technology', 'ITEC');

SELECT * FROM Department;
```

All Rows Fetched: 9 in 0.031 seconds

	DEPTID	DEPTNAME	DEPTCODE
1	1	Accounting	ACCT
2	2	Customer Relations	CUST
3	3	Directories and Databases	DRDB
4	4	Editorial	EDIT
5	5	Executive	EXEC
6	6	Human Resources	HMRS
7	7	Production	PROD
8	8	Sales	SALE
9	9	Information Technology	ITEC

The screenshot above shows that the 9 `INSERT` statements above were successfully inserted into the Department table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
Query Result 1
Task completed in 0.076 seconds

```
INSERT INTO Department (deptName, deptCode) VALUES ('Test Dept', 'test');
```

```
Error starting at line : 1 in command -
INSERT INTO Department (deptName, deptCode) VALUES ('Test Dept', 'test')
Error report -
SQL Error: ORA-02290: check constraint (PDBADMIN.SYS_C007652) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

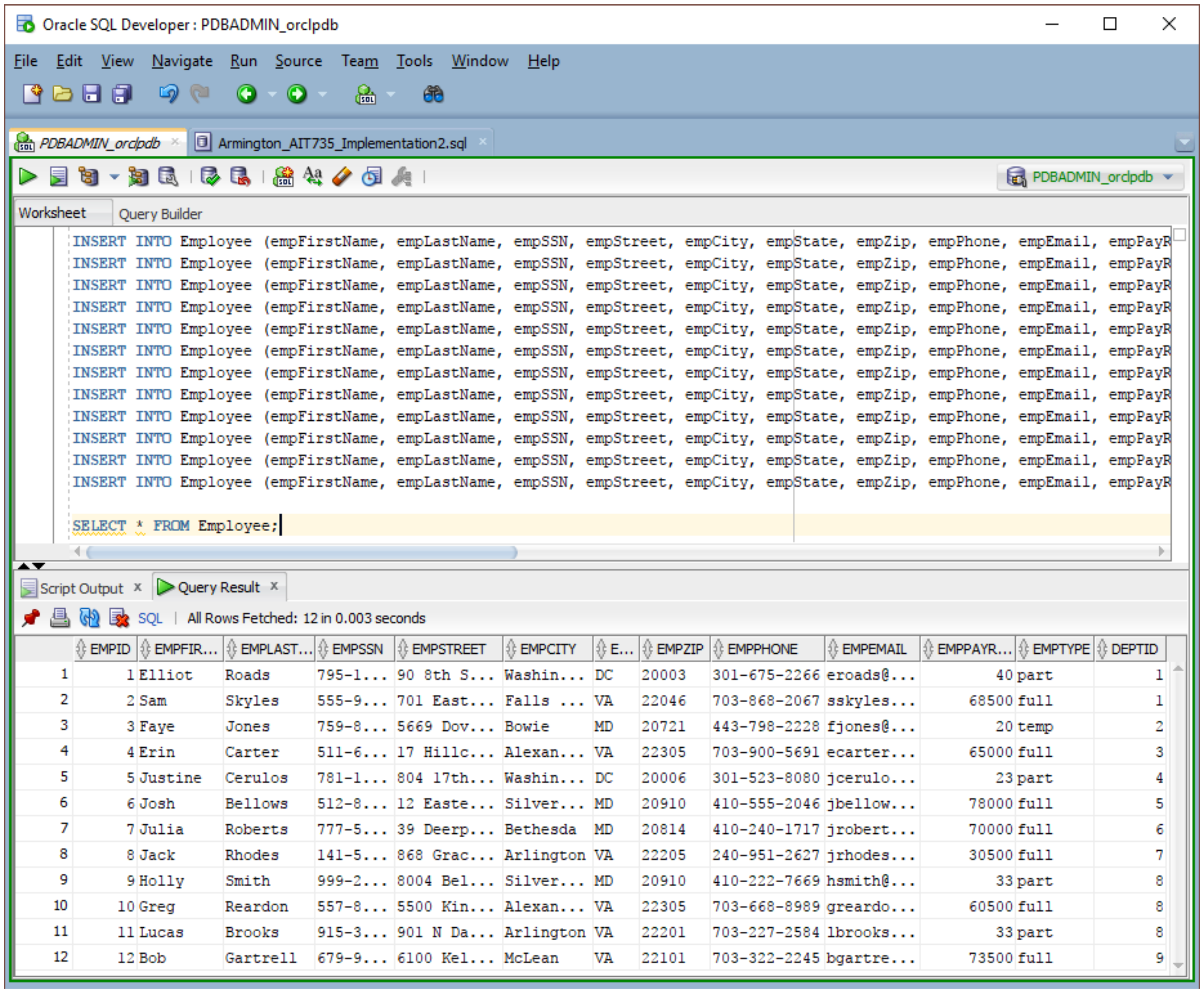
The screenshot above shows an unsuccessful `INSERT` statement into the Department table:

```
INSERT INTO Department (deptName, deptCode) VALUES ('Test Dept',
'test');
```

This table has a check constraint on the department code column (deptCode) to ensure that all incoming codes consist of four uppercase letters. The test case above attempts to insert a department code with four lowercase letters (test) so the insert fails.

Employee Table

```
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Elliot', 'Roads', '795-18-4981', '90
      8th Street SE', 'Washington', 'DC', '20003', '301-675-2266',
      'eroads@gmail.com', '40.00', 'part', '1');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Sam', 'Skyles', '555-99-1234', '701
      East Street', 'Falls Church', 'VA', '22046', '703-868-2067',
      'sskyles@gmail.com', '68500.00', 'full', '1');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Faye', 'Jones', '759-83-2280', '5669
      Dove Circle', 'Bowie', 'MD', '20721', '443-798-2228',
      'fjones@gmail.com', '20.00', 'temp', '2');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Erin', 'Carter', '511-69-8887', '17
      Hillcrest Drive', 'Alexandria', 'VA', '22305', '703-900-5691',
      'ecarter@gmail.com', '65000.00', 'full', '3');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Justine', 'Cerulos', '781-13-8494',
      '804 17th Street NW', 'Washington', 'DC', '20006', '301-523-
      8080', 'jcerulos@gmail.com', '23.00', 'part', '4');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Josh', 'Bellows', '512-88-6499', '12
      Eastern Drive', 'Silver Spring', 'MD', '20910', '410-555-2046',
      'jbellows@gmail.com', '78000.00', 'full', '5');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Julia', 'Roberts', '777-53-1649', '39
      Deerpark Road', 'Bethesda', 'MD', '20814', '410-240-1717',
      'jroberts@gmail.com', '70000.00', 'full', '6');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Jack', 'Rhodes', '141-51-4167', '868
      Grace Court', 'Arlington', 'VA', '22205', '240-951-2627',
      'jrhodes@gmail.com', '30500.00', 'full', '7');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Holly', 'Smith', '999-21-1890', '8004
      Belfast Drive', 'Silver Spring', 'MD', '20910', '410-222-7669',
      'hsmith@gmail.com', '33.00', 'part', '8');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Greg', 'Reardon', '557-88-1674', '5500
      King Street', 'Alexandria', 'VA', '22305', '703-668-8989',
      'greardon@gmail.com', '60500.00', 'full', '8');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Lucas', 'Brooks', '915-37-5156', '901 N
      Danville Street', 'Arlington', 'VA', '22201', '703-227-2584',
      'lbrooks@gmail.com', '33.00', 'part', '8');
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Bob', 'Gartrell', '679-92-6295', '6100
      Kellogg Drive', 'McLean', 'VA', '22101', '703-322-2245',
      'bgartrell@gmail.com', '73500.00', 'full', '9');
```

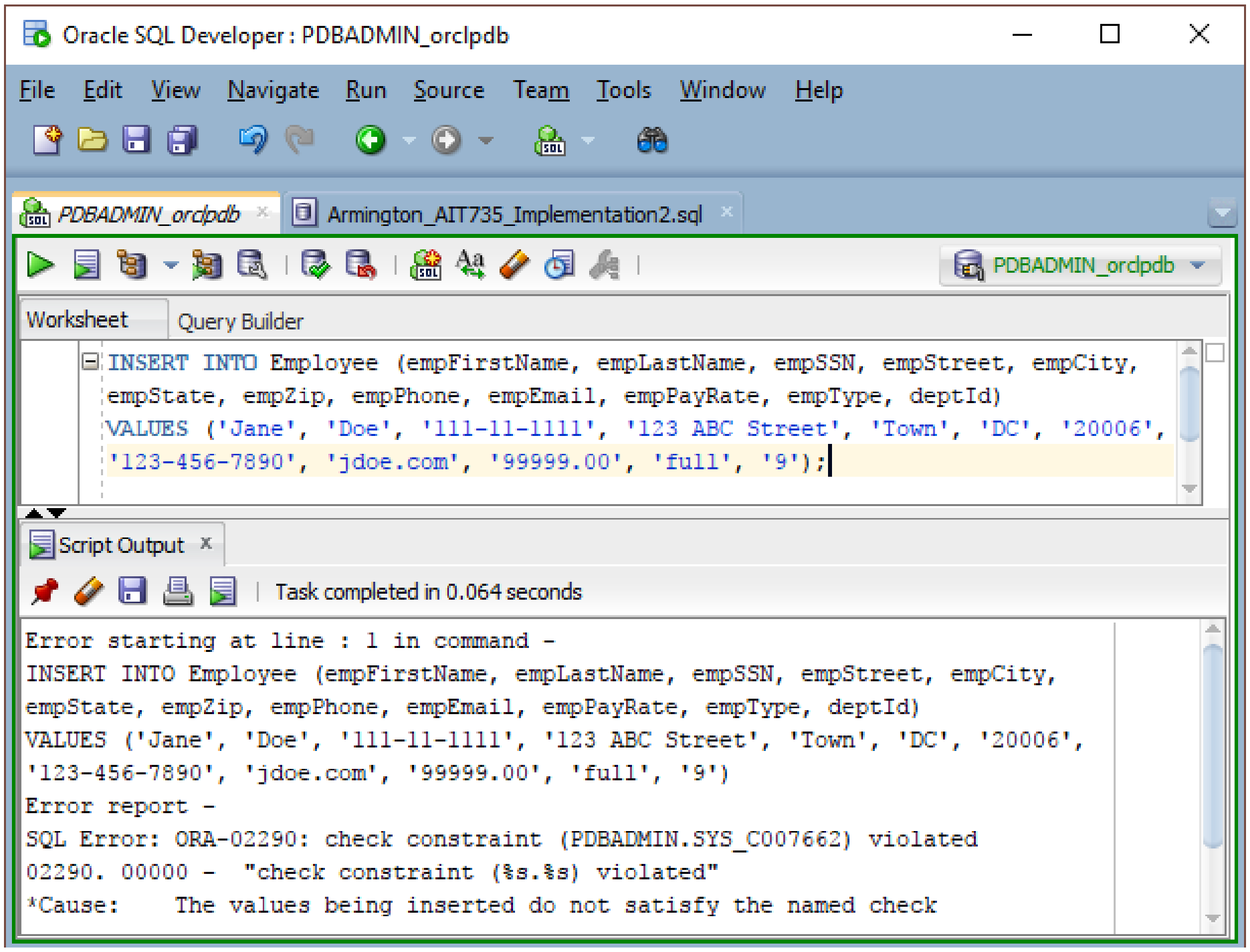

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity, empState, empZip, empPhone, empEmail, empPayR

SELECT * FROM Employee;
```

All Rows Fetched: 12 in 0.003 seconds

	EMPID	EMPFIR...	EMPLAST...	EMPSSN	EMPSTREET	EMPCITY	E...	EMPZIP	EMPPHONE	EMPEMAIL	EMPPAYR...	EMPTYPE	DEPTID
1	1	Elliot	Roads	795-1...	90 8th S...	Washin...	DC	20003	301-675-2266	eroads@...	40	part	1
2	2	Sam	Skyles	555-9...	701 East...	Falls ...	VA	22046	703-868-2067	sskyles...	68500	full	1
3	3	Faye	Jones	759-8...	5669 Dov...	Bowie	MD	20721	443-798-2228	fjones@...	20	temp	2
4	4	Erin	Carter	511-6...	17 Hillc...	Alexan...	VA	22305	703-900-5691	ecarter...	65000	full	3
5	5	Justine	Cerulos	781-1...	804 17th...	Washin...	DC	20006	301-523-8080	jcerulo...	23	part	4
6	6	Josh	Bellows	512-8...	12 Easte...	Silver...	MD	20910	410-555-2046	jbellow...	78000	full	5
7	7	Julia	Roberts	777-5...	39 Deerp...	Bethesda	MD	20814	410-240-1717	jrobert...	70000	full	6
8	8	Jack	Rhodes	141-5...	868 Grac...	Arlington	VA	22205	240-951-2627	jrhodes...	30500	full	7
9	9	Holly	Smith	999-2...	8004 Bel...	Silver...	MD	20910	410-222-7669	hsmith@...	33	part	8
10	10	Greg	Reardon	557-8...	5500 Kin...	Alexan...	VA	22305	703-668-8989	greardo...	60500	full	8
11	11	Lucas	Brooks	915-3...	901 N Da...	Arlington	VA	22201	703-227-2584	lbrooks...	33	part	8
12	12	Bob	Gartrell	679-9...	6100 Kel...	McLean	VA	22101	703-322-2245	bgartre...	73500	full	9

The screenshot above shows that the 12 `INSERT` statements above were successfully inserted into the Employee table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.064 seconds

```
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity,
empState, empZip, empPhone, empEmail, empPayRate, empType, deptId)
VALUES ('Jane', 'Doe', '111-11-1111', '123 ABC Street', 'Town', 'DC', '20006',
'123-456-7890', 'jdoe.com', '99999.00', 'full', '9');
```

```
Error starting at line : 1 in command -
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet, empCity,
empState, empZip, empPhone, empEmail, empPayRate, empType, deptId)
VALUES ('Jane', 'Doe', '111-11-1111', '123 ABC Street', 'Town', 'DC', '20006',
'123-456-7890', 'jdoe.com', '99999.00', 'full', '9')
Error report -
SQL Error: ORA-02290: check constraint (PDBADMIN.SYS_C007662) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check
```

The screenshot above shows an unsuccessful `INSERT` statement into the Employee table:

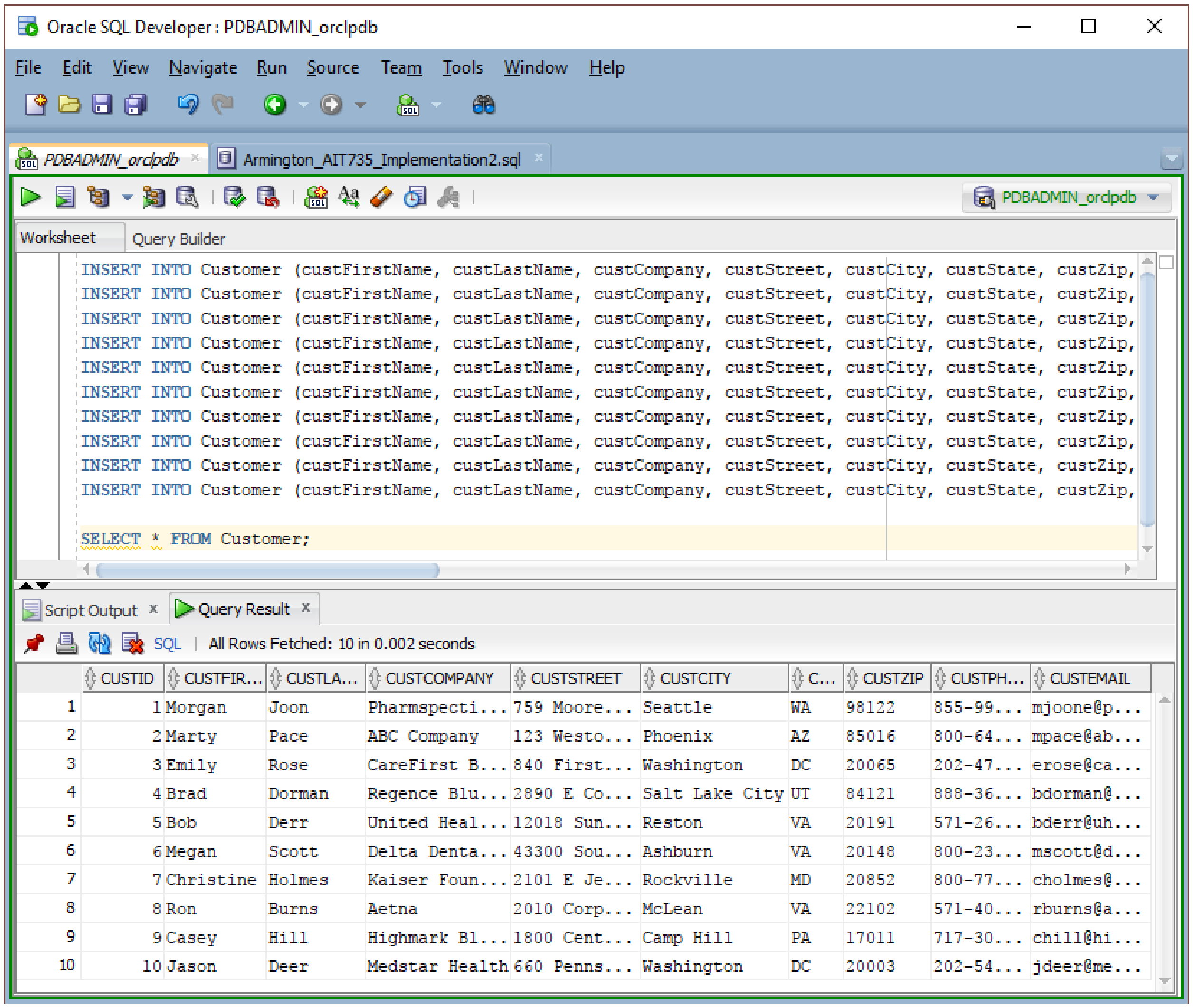
```
INSERT INTO Employee (empFirstName, empLastName, empSSN, empStreet,
      empCity, empState, empZip, empPhone, empEmail, empPayRate,
      empType, deptId) VALUES ('Jane', 'Doe', '111-11-1111', '123 ABC
      Street', 'Town', 'DC', '20006', '123-456-7890', 'jdoe.com',
      '99999.00', 'full', '9');
```

This table has a check constraint on the employee email column (empEmail) to ensure that all incoming email addresses contain the @ character. The test case above attempts to insert an invalid email address (jdoe.com) so the insert fails.

Customer Table

```
INSERT INTO Customer (custFirstName, custLastName, custCompany,
      custStreet, custCity, custState, custZip, custPhone, custEmail)
      VALUES ('Morgan', 'Joon', 'Pharmspective LLC', '759 Moores
      Circle', 'Seattle', 'WA', '98122', '855-999-1787',
      'mjoone@pharmspective.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
      custStreet, custCity, custState, custZip, custPhone, custEmail)
      VALUES ('Marty', 'Pace', 'ABC Company', '123 Weston Road',
      'Phoenix', 'AZ', '85016', '800-647-7070',
      'mpace@abccompany.com');
```

```
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Emily', 'Rose', 'CareFirst BlueCross BlueShield', '840
     First Street NE', 'Washington', 'DC', '20065', '202-479-8000',
     'erose@carefirst.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Brad', 'Dorman', 'Regence Bluecross Blueshield of Utah',
     '2890 E Cottonwood Parkway', 'Salt Lake City', 'UT', '84121',
     '888-367-2119', 'bdorman@regence.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Bob', 'Derr', 'United Healthcare', '12018 Sunrise Valley
     Drive, Suite 400', 'Reston', 'VA', '20191', '571-262-2245',
     'bderr@uhc.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Megan', 'Scott', 'Delta Dental Plan of Virginia', '43300
     Southern Walk Plaza, Suite 116', 'Ashburn', 'VA', '20148', '800-
     237-6060', 'mscott@deltadentalva.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Christine', 'Holmes', 'Kabcer Foundation Health Plan of
     the Mid-Atlantic States', '2101 E Jefferson Street', 'Rockville',
     'MD', '20852', '800-777-7904', 'cholmes@kabcer.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Ron', 'Burns', 'Aetna', '2010 Corporate Ridge',
     'McLean', 'VA', '22102', '571-406-3103', 'rburns@aetna.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Casey', 'Hill', 'Highmark Blue Shield', '1800 Center
     Street', 'Camp Hill', 'PA', '17011', '717-302-5000',
     'chill@highmark.com');
INSERT INTO Customer (custFirstName, custLastName, custCompany,
     custStreet, custCity, custState, custZip, custPhone, custEmail)
     VALUES ('Jason', 'Deer', 'Medstar Health', '660 Pennsylvania
     Avenue SE', 'Washington', 'DC', '20003', '202-546-4504',
     'jdeer@medstar.com');
```

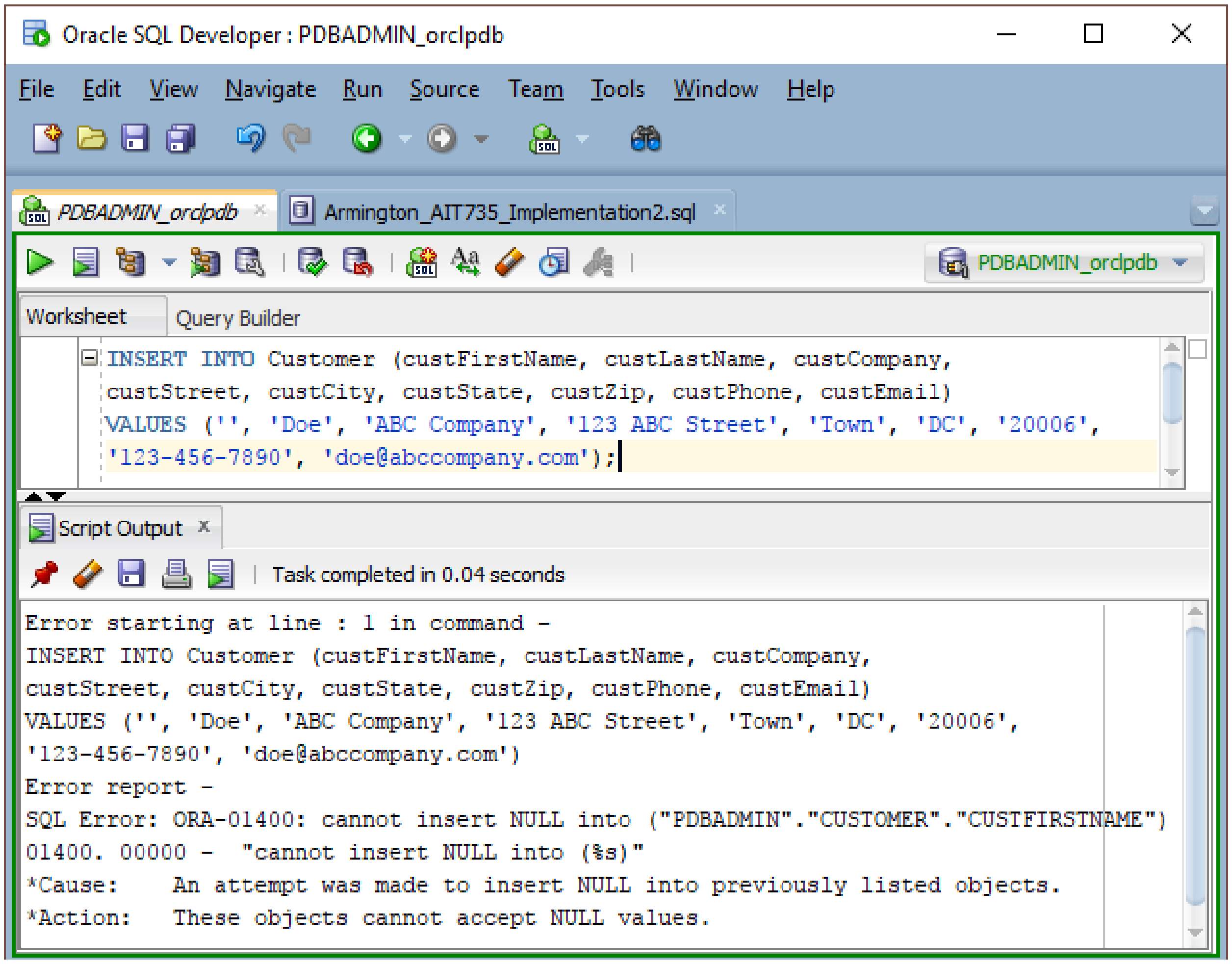

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,
INSERT INTO Customer (custFirstName, custLastName, custCompany, custStreet, custCity, custState, custZip,

SELECT * FROM Customer;
```

All Rows Fetched: 10 in 0.002 seconds

	CUSTID	CUSTFIR...	CUSTLA...	CUSTCOMPANY	CUSTSTREET	CUSTCITY	C...	CUSTZIP	CUSTPH...	CUSTEMAIL
1	1	Morgan	Joon	Pharmspecti...	759 Moore...	Seattle	WA	98122	855-99...	mjoone@p...
2	2	Marty	Pace	ABC Company	123 Westo...	Phoenix	AZ	85016	800-64...	mpace@ab...
3	3	Emily	Rose	CareFirst B...	840 First...	Washington	DC	20065	202-47...	erose@ca...
4	4	Brad	Dorman	Regence Blu...	2890 E Co...	Salt Lake City	UT	84121	888-36...	bdorman@...
5	5	Bob	Derr	United Heal...	12018 Sun...	Reston	VA	20191	571-26...	bderr@uh...
6	6	Megan	Scott	Delta Denta...	43300 Sou...	Ashburn	VA	20148	800-23...	mscott@d...
7	7	Christine	Holmes	Kaiser Foun...	2101 E Je...	Rockville	MD	20852	800-77...	cholmes@...
8	8	Ron	Burns	Aetna	2010 Corp...	McLean	VA	22102	571-40...	rburns@a...
9	9	Casey	Hill	Highmark Bl...	1800 Cent...	Camp Hill	PA	17011	717-30...	chill@hi...
10	10	Jason	Deer	Medstar Health	660 Penns...	Washington	DC	20003	202-54...	jdeer@me...

The screenshot above shows that the 10 `INSERT` statements above were successfully inserted into the Customer table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.04 seconds

```
INSERT INTO Customer (custFirstName, custLastName, custCompany,
custStreet, custCity, custState, custZip, custPhone, custEmail)
VALUES ('', 'Doe', 'ABC Company', '123 ABC Street', 'Town', 'DC', '20006',
'123-456-7890', 'doe@abccompany.com');
```

```
Error starting at line : 1 in command -
INSERT INTO Customer (custFirstName, custLastName, custCompany,
custStreet, custCity, custState, custZip, custPhone, custEmail)
VALUES ('', 'Doe', 'ABC Company', '123 ABC Street', 'Town', 'DC', '20006',
'123-456-7890', 'doe@abccompany.com')
Error report -
SQL Error: ORA-01400: cannot insert NULL into ("PDBADMIN"."CUSTOMER"."CUSTFIRSTNAME")
01400. 00000 -  "cannot insert NULL into (%s)"
*Cause:    An attempt was made to insert NULL into previously listed objects.
*Action:   These objects cannot accept NULL values.
```

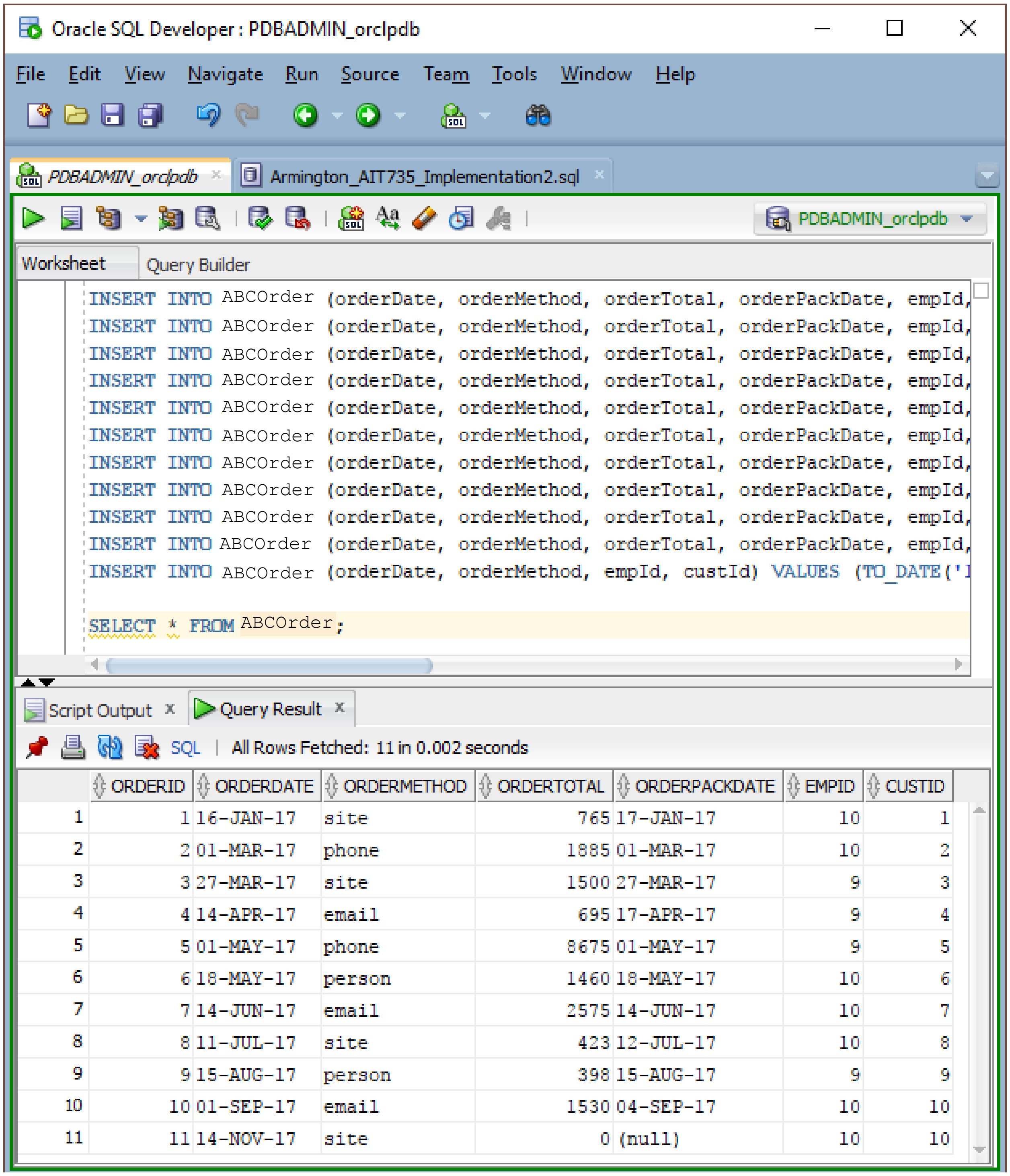
The screenshot above shows an unsuccessful `INSERT` statement into the Customer table:

```
INSERT INTO Customer (custFirstName, custLastName, custCompany,
custStreet, custCity, custState, custZip, custPhone, custEmail) VALUES
('', 'Doe', 'ABC Company', '123 ABC Street', 'Town', 'DC', '20006',
'123-456-7890', 'doe@abccompany.com');
```

This table has a `NOT NULL` constraint on the customer first name column (custFirstName) to ensure that all inserted rows contain the customers' first name. The test case above attempts to insert a customer without a first name so the insert fails.

ABCOrder Table

```
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('01/16/2017',
      'MM/DD/YYYY'), 'site', '765.00', TO_DATE('01/17/2017',
      'MM/DD/YYYY'), '10', '1');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('03/01/2017',
      'MM/DD/YYYY'), 'phone', '1885.00', TO_DATE('03/01/2017',
      'MM/DD/YYYY'), '10', '2');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('03/27/2017',
      'MM/DD/YYYY'), 'site', '1500.00', TO_DATE('03/27/2017',
      'MM/DD/YYYY'), '9', '3');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('04/14/2017',
      'MM/DD/YYYY'), 'email', '695.00', TO_DATE('04/17/2017',
      'MM/DD/YYYY'), '9', '4');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('05/01/2017',
      'MM/DD/YYYY'), 'phone', '8675.00', TO_DATE('05/01/2017',
      'MM/DD/YYYY'), '9', '5');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('05/18/2017',
      'MM/DD/YYYY'), 'person', '1460.00', TO_DATE('05/18/2017',
      'MM/DD/YYYY'), '10', '6');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('06/14/2017',
      'MM/DD/YYYY'), 'email', '2575.00', TO_DATE('06/14/2017',
      'MM/DD/YYYY'), '10', '7');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('07/11/2017',
      'MM/DD/YYYY'), 'site', '423.00', TO_DATE('07/12/2017',
      'MM/DD/YYYY'), '10', '8');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('08/15/2017',
      'MM/DD/YYYY'), 'person', '398.00', TO_DATE('08/15/2017',
      'MM/DD/YYYY'), '9', '9');
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal,
      orderPackDate, empId, custId) VALUES (TO_DATE('09/01/2017',
      'MM/DD/YYYY'), 'email', '1530.00', TO_DATE('09/04/2017',
      'MM/DD/YYYY'), '10', '10');
INSERT INTO ABCOrder (orderDate, orderMethod, empId, custId) VALUES
      (TO_DATE('11/14/2017', 'MM/DD/YYYY'), 'site', '10', '10');
```

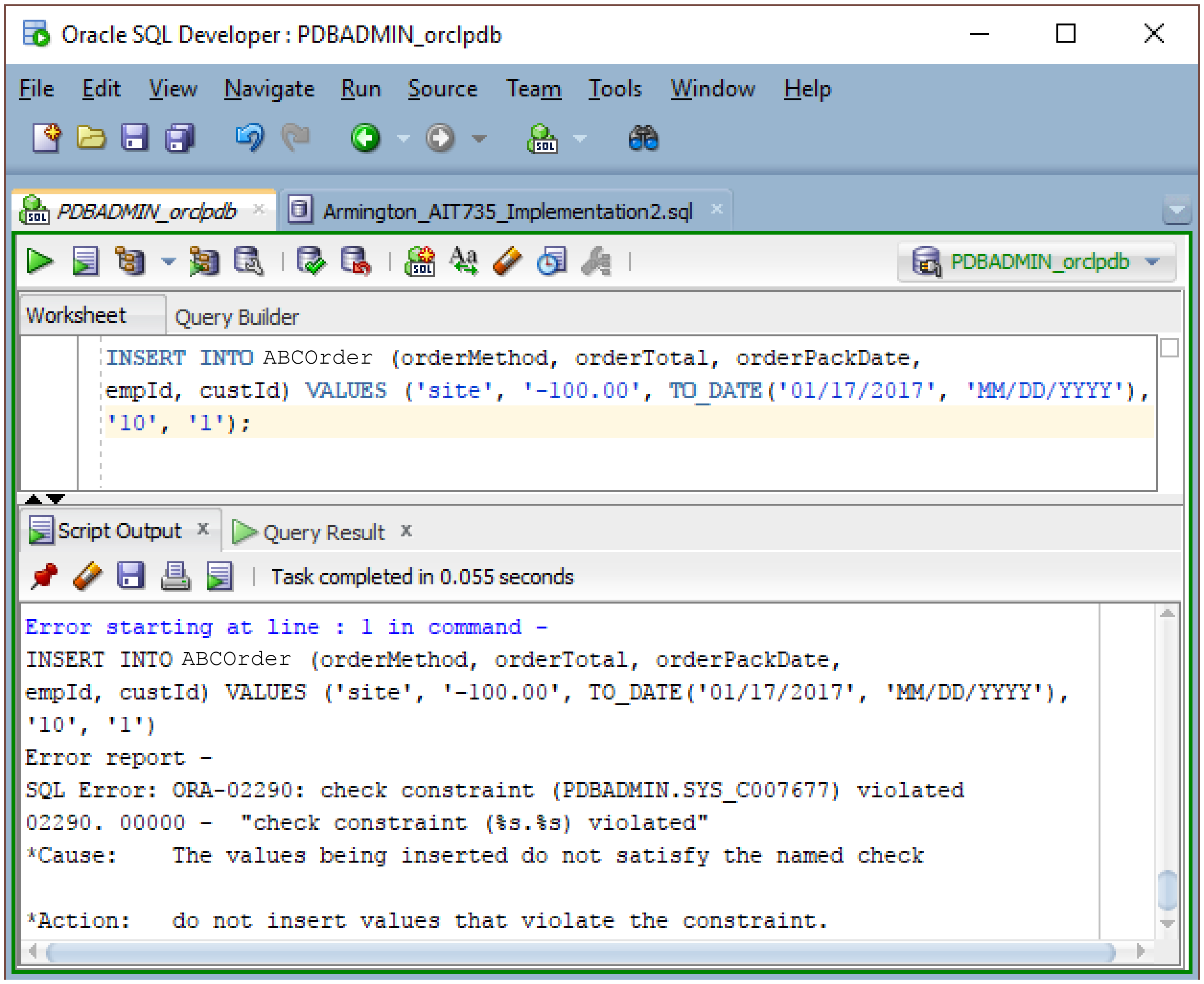

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, orderTotal, orderPackDate, empId,
INSERT INTO ABCOrder (orderDate, orderMethod, empId, custId) VALUES (TO_DATE('

SELECT * FROM ABCOrder;
```

All Rows Fetched: 11 in 0.002 seconds

	ORDERID	ORDERDATE	ORDERMETHOD	ORDERTOTAL	ORDERPACKDATE	EMPID	CUSTID
1	1	16-JAN-17	site	765	17-JAN-17	10	1
2	2	01-MAR-17	phone	1885	01-MAR-17	10	2
3	3	27-MAR-17	site	1500	27-MAR-17	9	3
4	4	14-APR-17	email	695	17-APR-17	9	4
5	5	01-MAY-17	phone	8675	01-MAY-17	9	5
6	6	18-MAY-17	person	1460	18-MAY-17	10	6
7	7	14-JUN-17	email	2575	14-JUN-17	10	7
8	8	11-JUL-17	site	423	12-JUL-17	10	8
9	9	15-AUG-17	person	398	15-AUG-17	9	9
10	10	01-SEP-17	email	1530	04-SEP-17	10	10
11	11	14-NOV-17	site	0	(null)	10	10

The screenshot above shows that the 11 `INSERT` statements above were successfully inserted into the ABCOrder table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
Task completed in 0.055 seconds

```
INSERT INTO ABCOrder (orderMethod, orderTotal, orderPackDate,
empId, custId) VALUES ('site', '-100.00', TO_DATE('01/17/2017', 'MM/DD/YYYY'),
'10', '1');
```

```
Error starting at line : 1 in command -
INSERT INTO ABCOrder (orderMethod, orderTotal, orderPackDate,
empId, custId) VALUES ('site', '-100.00', TO_DATE('01/17/2017', 'MM/DD/YYYY'),
'10', '1')
Error report -
SQL Error: ORA-02290: check constraint (PDBADMIN.SYS_C007677) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

The screenshot above shows an unsuccessful `INSERT` statement into the ABCOrder table:

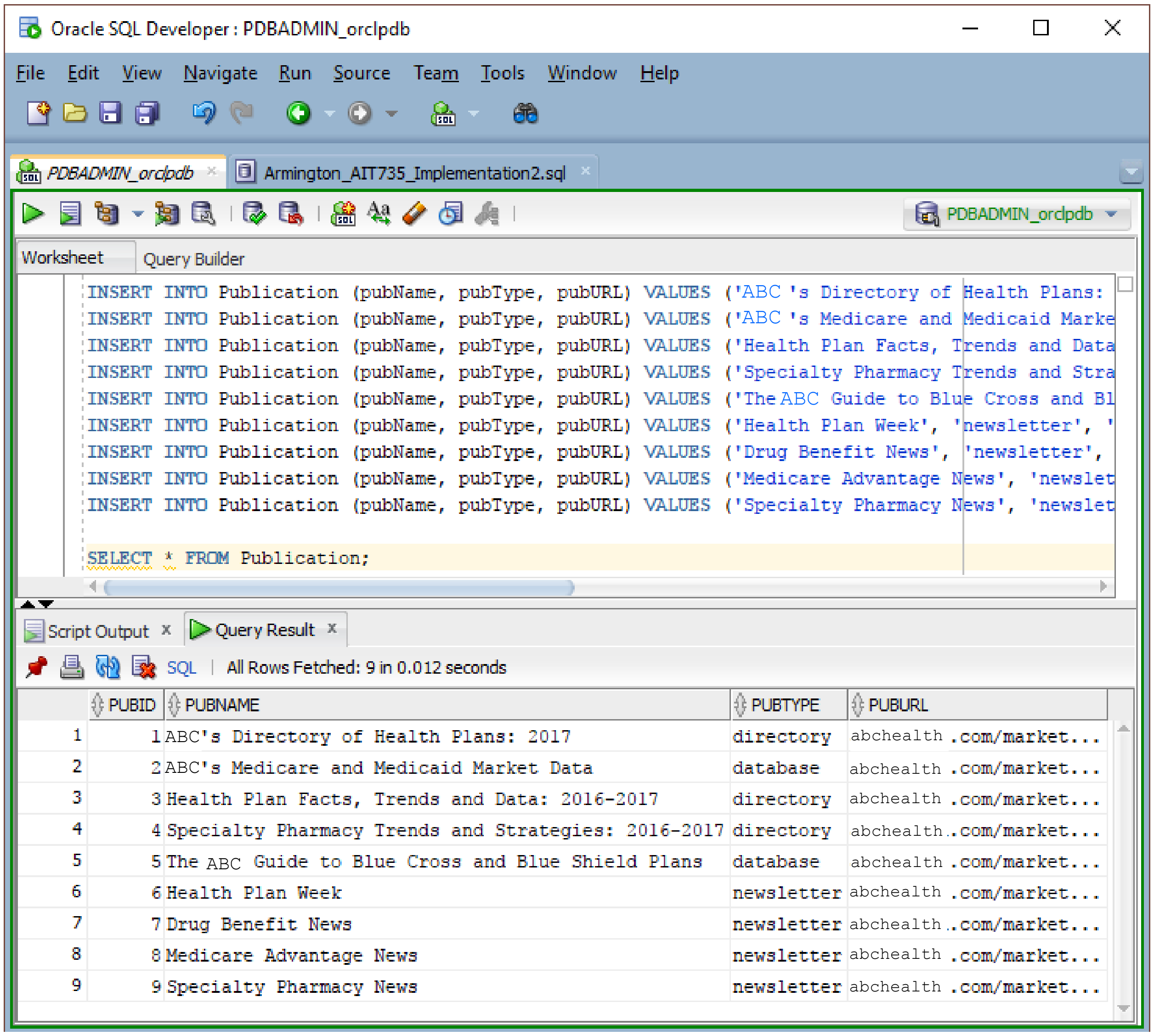
```
INSERT INTO ABCOrder (orderMethod, orderTotal, orderPackDate, empId,
custId) VALUES ('site', '-100.00', TO_DATE('01/17/2017',
'MM/DD/YYYY'), '10', '1');
```

This table has a check constraint on the order total column (orderTotal) to ensure that all incoming order totals are greater than or equal to 0. The test case above attempts to insert a negative order total (-100.00) so the insert fails.

Publication Table

```
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('ABC''s
      Directory of Health Plans: 2017', 'directory',
      'abchealth.com/marketplace/abcs-directory-health-plans');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('ABC''s
      Medicare and Medicaid Market Data', 'database',
      'abchealth.com/marketplace/managed-medicare-and-medicaid-market-
      data');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Health
      Plan Facts, Trends and Data: 2016-2017', 'directory',
      'abchealth.com/marketplace/health-plan-facts-trends-and-data');
```

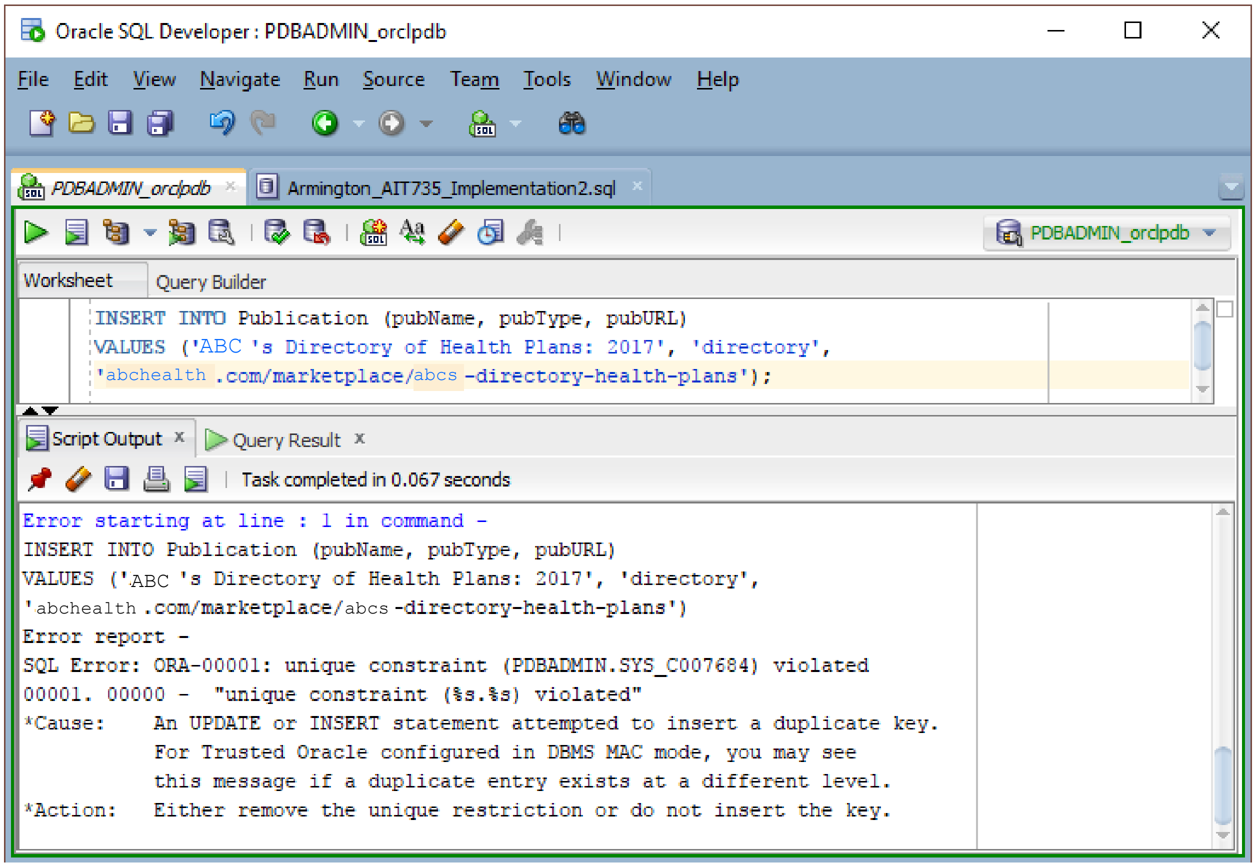
```
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Specialty
     Pharmacy Trends and Strategies: 2016-2017', 'directory',
     'abchealth.com/marketplace/specialty-pharmacy-trends-and-
     strategies');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('The ABC
     Guide to Blue Cross and Blue Shield Plans', 'database',
     'abchealth.com/marketplace/abc-guide-blue-cross-and-blue-shield-
     plans');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Health
     Plan Week', 'newsletter', 'abchealth.com/marketplace/health-plan-
     week');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Drug
     Benefit News', 'newsletter', 'abchealth.com/marketplace/drug-
     benefit-news');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Medicare
     Advantage News', 'newsletter',
     'abchealth.com/marketplace/medicare-advantage-news');
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Specialty
     Pharmacy News', 'newsletter',
     'abchealth.com/marketplace/specialty-pharmacy-news');
```


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
SQL
All Rows Fetched: 9 in 0.012 seconds

```
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('ABC 's Directory of Health Plans:
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('ABC 's Medicare and Medicaid Marke
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Health Plan Facts, Trends and Data
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Specialty Pharmacy Trends and Stra
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('The ABC Guide to Blue Cross and Bl
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Health Plan Week', 'newsletter', '
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Drug Benefit News', 'newsletter',
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Medicare Advantage News', 'newslet
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('Specialty Pharmacy News', 'newslet

SELECT * FROM Publication;
```

	PUBID	PUBNAME	PUBTYPE	PUBURL
1	1	ABC's Directory of Health Plans: 2017	directory	abchealth .com/market...
2	2	ABC's Medicare and Medicaid Market Data	database	abchealth .com/market...
3	3	Health Plan Facts, Trends and Data: 2016-2017	directory	abchealth .com/market...
4	4	Specialty Pharmacy Trends and Strategies: 2016-2017	directory	abchealth .com/market...
5	5	The ABC Guide to Blue Cross and Blue Shield Plans	database	abchealth .com/market...
6	6	Health Plan Week	newsletter	abchealth .com/market...
7	7	Drug Benefit News	newsletter	abchealth .com/market...
8	8	Medicare Advantage News	newsletter	abchealth .com/market...
9	9	Specialty Pharmacy News	newsletter	abchealth .com/market...

The screenshot above shows that the 9 `INSERT` statements above were successfully inserted into the Publication table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
Task completed in 0.067 seconds

```
INSERT INTO Publication (pubName, pubType, pubURL)
VALUES ('ABC 's Directory of Health Plans: 2017', 'directory',
'abchealth .com/marketplace/abcs -directory-health-plans');
```

```
Error starting at line : 1 in command -
INSERT INTO Publication (pubName, pubType, pubURL)
VALUES ('ABC 's Directory of Health Plans: 2017', 'directory',
'abchealth .com/marketplace/abcs -directory-health-plans')
Error report -
SQL Error: ORA-00001: unique constraint (PDBADMIN.SYS_C007684) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
           this message if a duplicate entry exists at a different level.
*Action:   Either remove the unique restriction or do not insert the key.
```

The screenshot above shows an unsuccessful `INSERT` statement into the Publication table:

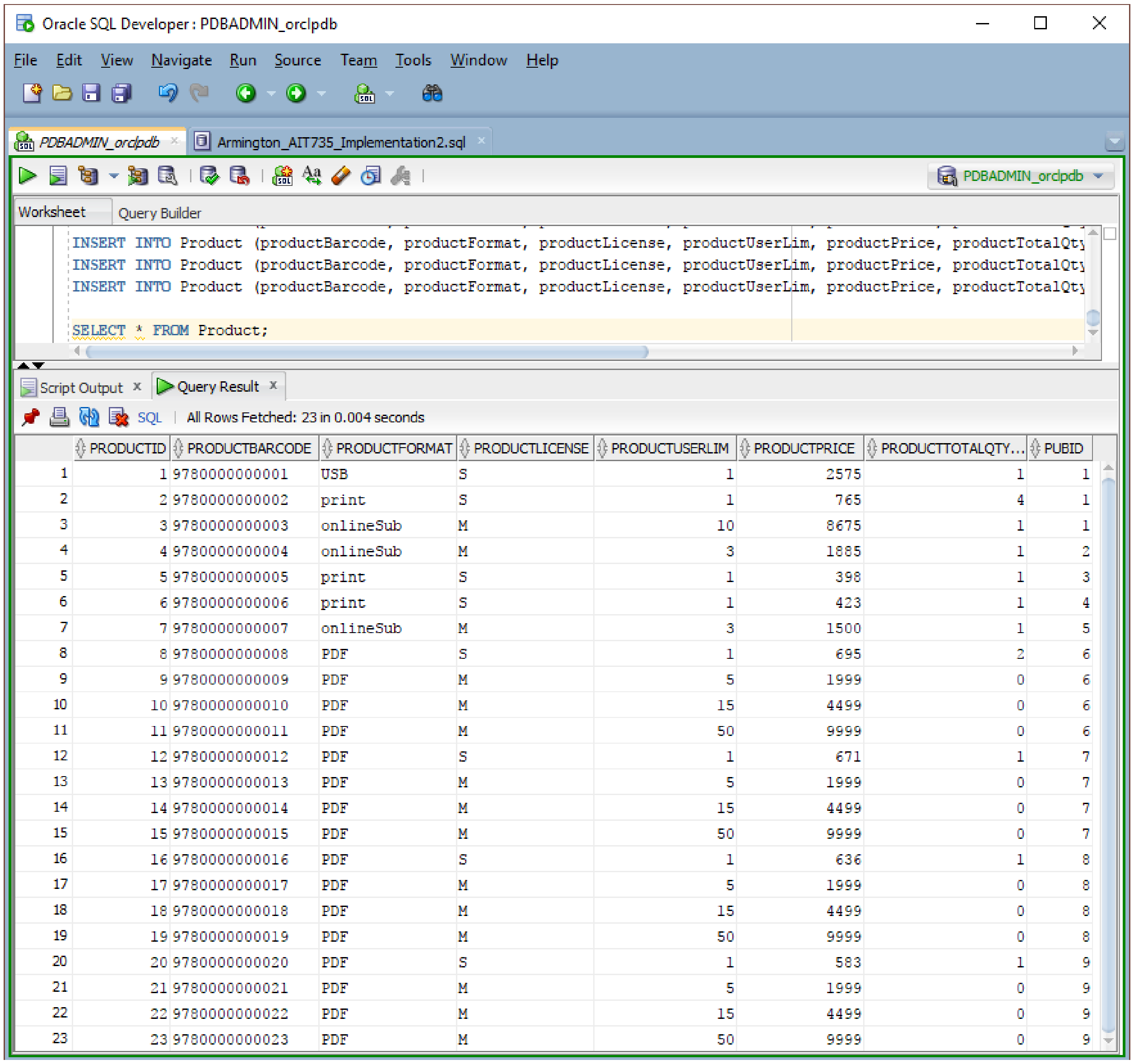
```
INSERT INTO Publication (pubName, pubType, pubURL) VALUES ('ABC''s
Directory of Health Plans: 2017', 'directory',
'abchealth.com/marketplace/abcs-directory-health-plans');
```

This table has a `UNIQUE` constraint on the publication name column (pubName) to ensure that all incoming publication names are unique. The test case above attempts to insert a publication name that already exists in the table (ABC's Directory of Health Plans: 2017) so the insert fails.

Product Table

```
INSERT INTO Product (productBarcode, productFormat, productLicense,
      productUserLim, productPrice, productTotalQtySold, pubId) VALUES
      ('9780000000001', 'USB', 'S', '1', '2575.00', '1', '1');
INSERT INTO Product (productBarcode, productFormat, productLicense,
      productUserLim, productPrice, productTotalQtySold, pubId) VALUES
      ('9780000000002', 'print', 'S', '1', '765.00', '4', '1');
INSERT INTO Product (productBarcode, productFormat, productLicense,
      productUserLim, productPrice, productTotalQtySold, pubId) VALUES
      ('9780000000003', 'onlineSub', 'M', '10', '8675.00', '1', '1');
INSERT INTO Product (productBarcode, productFormat, productLicense,
      productUserLim, productPrice, productTotalQtySold, pubId) VALUES
      ('9780000000004', 'onlineSub', 'M', '3', '1885.00', '1', '2');
INSERT INTO Product (productBarcode, productFormat, productLicense,
      productUserLim, productPrice, productTotalQtySold, pubId) VALUES
      ('9780000000005', 'print', 'S', '1', '398.00', '1', '3');
```

```
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000006', 'print', 'S', '1', '423.00', '1', '4');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000007', 'onlineSub', 'M', '3', '1500.00', '1', '5');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000008', 'PDF', 'S', '1', '695.00', '2', '6');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000009', 'PDF', 'M', '5', '1999.00', '0', '6');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000010', 'PDF', 'M', '15', '4499.00', '0', '6');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000011', 'PDF', 'M', '50', '9999.00', '0', '6');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000012', 'PDF', 'S', '1', '671.00', '1', '7');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000013', 'PDF', 'M', '5', '1999.00', '0', '7');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000014', 'PDF', 'M', '15', '4499.00', '0', '7');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000015', 'PDF', 'M', '50', '9999.00', '0', '7');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000016', 'PDF', 'S', '1', '636.00', '1', '8');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000017', 'PDF', 'M', '5', '1999.00', '0', '8');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000018', 'PDF', 'M', '15', '4499.00', '0', '8');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000019', 'PDF', 'M', '50', '9999.00', '0', '8');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000020', 'PDF', 'S', '1', '583.00', '1', '9');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000021', 'PDF', 'M', '5', '1999.00', '0', '9');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000022', 'PDF', 'M', '15', '4499.00', '0', '9');
INSERT INTO Product (productBarcode, productFormat, productLicense,
     productUserLim, productPrice, productTotalQtySold, pubId) VALUES
     ('9780000000023', 'PDF', 'M', '50', '9999.00', '0', '9');
```

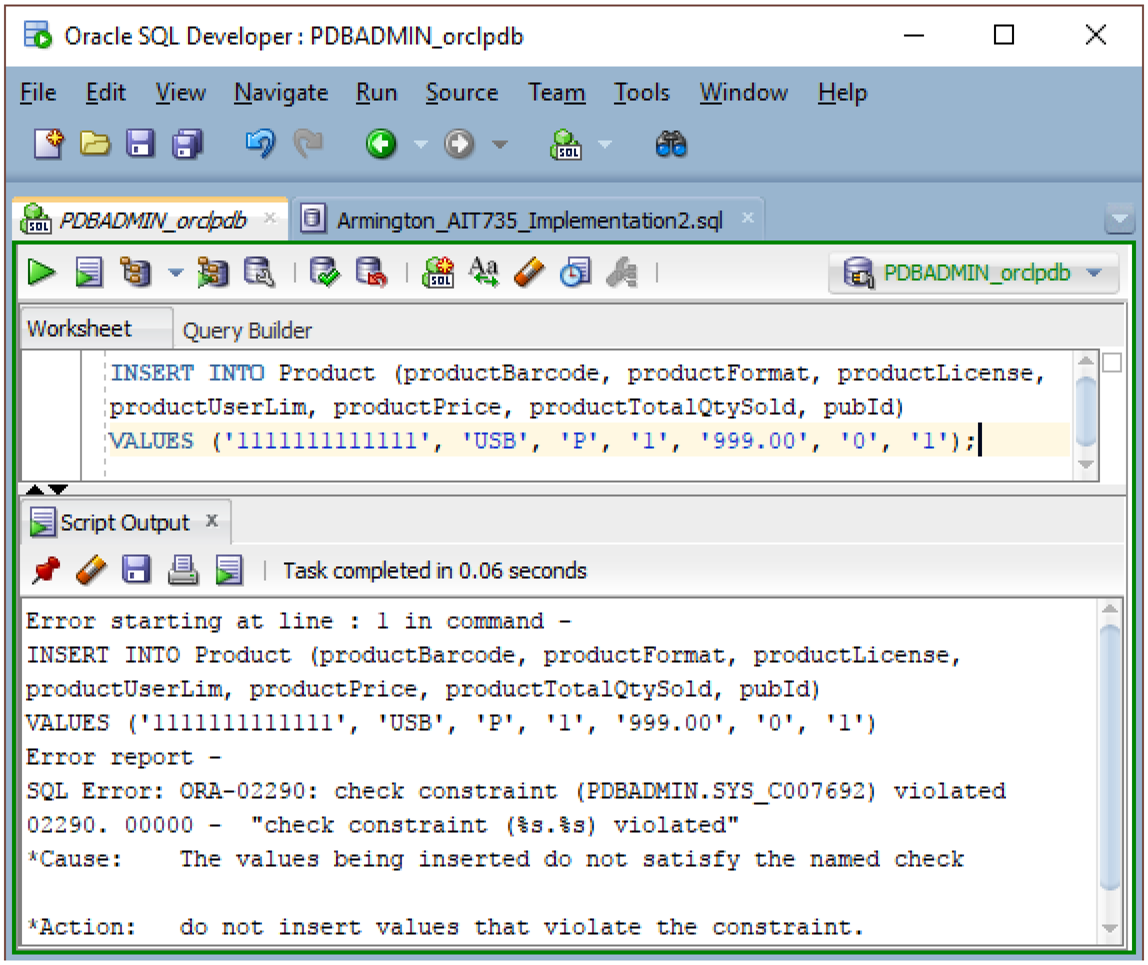

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO Product (productBarcode, productFormat, productLicense, productUserLim, productPrice, productTotalQty
INSERT INTO Product (productBarcode, productFormat, productLicense, productUserLim, productPrice, productTotalQty
INSERT INTO Product (productBarcode, productFormat, productLicense, productUserLim, productPrice, productTotalQty

SELECT * FROM Product;
```

All Rows Fetched: 23 in 0.004 seconds

	PRODUCTID	PRODUCTBARCODE	PRODUCTFORMAT	PRODUCTLICENSE	PRODUCTUSERLIM	PRODUCTPRICE	PRODUCTTOTALQTY...	PUBID
1	1	9780000000001	USB	S	1	2575	1	1
2	2	9780000000002	print	S	1	765	4	1
3	3	9780000000003	onlineSub	M	10	8675	1	1
4	4	9780000000004	onlineSub	M	3	1885	1	2
5	5	9780000000005	print	S	1	398	1	3
6	6	9780000000006	print	S	1	423	1	4
7	7	9780000000007	onlineSub	M	3	1500	1	5
8	8	9780000000008	PDF	S	1	695	2	6
9	9	9780000000009	PDF	M	5	1999	0	6
10	10	9780000000010	PDF	M	15	4499	0	6
11	11	9780000000011	PDF	M	50	9999	0	6
12	12	9780000000012	PDF	S	1	671	1	7
13	13	9780000000013	PDF	M	5	1999	0	7
14	14	9780000000014	PDF	M	15	4499	0	7
15	15	9780000000015	PDF	M	50	9999	0	7
16	16	9780000000016	PDF	S	1	636	1	8
17	17	9780000000017	PDF	M	5	1999	0	8
18	18	9780000000018	PDF	M	15	4499	0	8
19	19	9780000000019	PDF	M	50	9999	0	8
20	20	9780000000020	PDF	S	1	583	1	9
21	21	9780000000021	PDF	M	5	1999	0	9
22	22	9780000000022	PDF	M	15	4499	0	9
23	23	9780000000023	PDF	M	50	9999	0	9

The screenshot above shows that the 23 `INSERT` statements above were successfully inserted into the Product table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Task completed in 0.06 seconds

```
INSERT INTO Product (productBarcode, productFormat, productLicense,
productUserLim, productPrice, productTotalQtySold, pubId)
VALUES ('1111111111111', 'USB', 'P', '1', '999.00', '0', '1');
```

```
Error starting at line : 1 in command -
INSERT INTO Product (productBarcode, productFormat, productLicense,
productUserLim, productPrice, productTotalQtySold, pubId)
VALUES ('1111111111111', 'USB', 'P', '1', '999.00', '0', '1')
Error report -
SQL Error: ORA-02290: check constraint (PDBADMIN.SYS_C007692) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

The screenshot above shows an unsuccessful `INSERT` statement into the Product table:

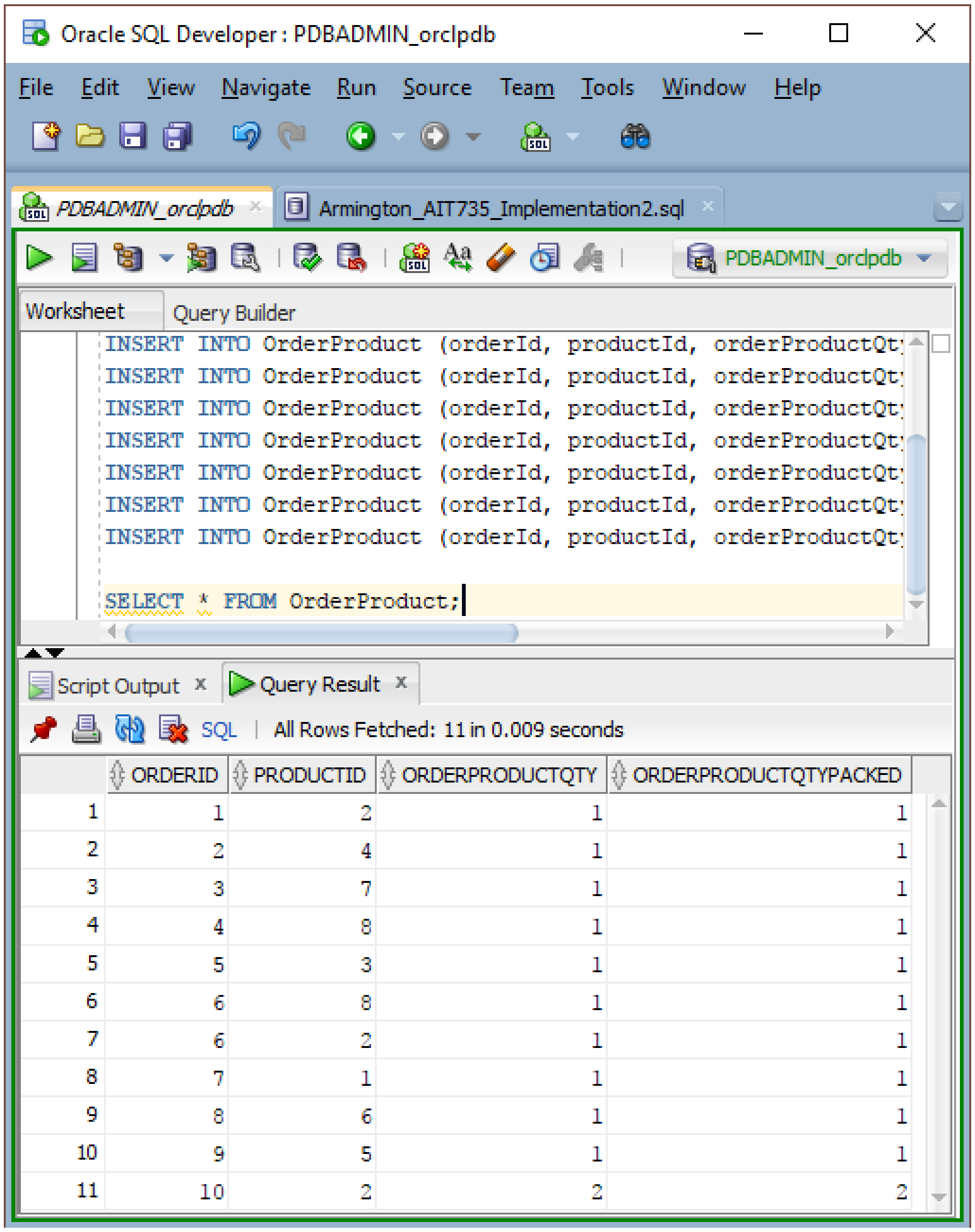
```
INSERT INTO Product (productBarcode, productFormat, productLicense,
productUserLim, productPrice, productTotalQtySold, pubId) VALUES
('1111111111111', 'USB', 'P', '1', '999.00', '0', '1');
```

This table has a check constraint on the product license column (productLicense) to ensure that all inserted products have either a single license (S) or a multi-user license (M). The test case above attempts to insert a product with an invalid license (P) so the insert fails.

Order Product Table

```
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('1', '2', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('2', '4', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('3', '7', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('4', '8', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('5', '3', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('6', '8', '1', '1');
```

```
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('6', '2', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('7', '1', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('8', '6', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('9', '5', '1', '1');
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
      orderProductQtyPacked) VALUES ('10', '2', '2', '2');
```

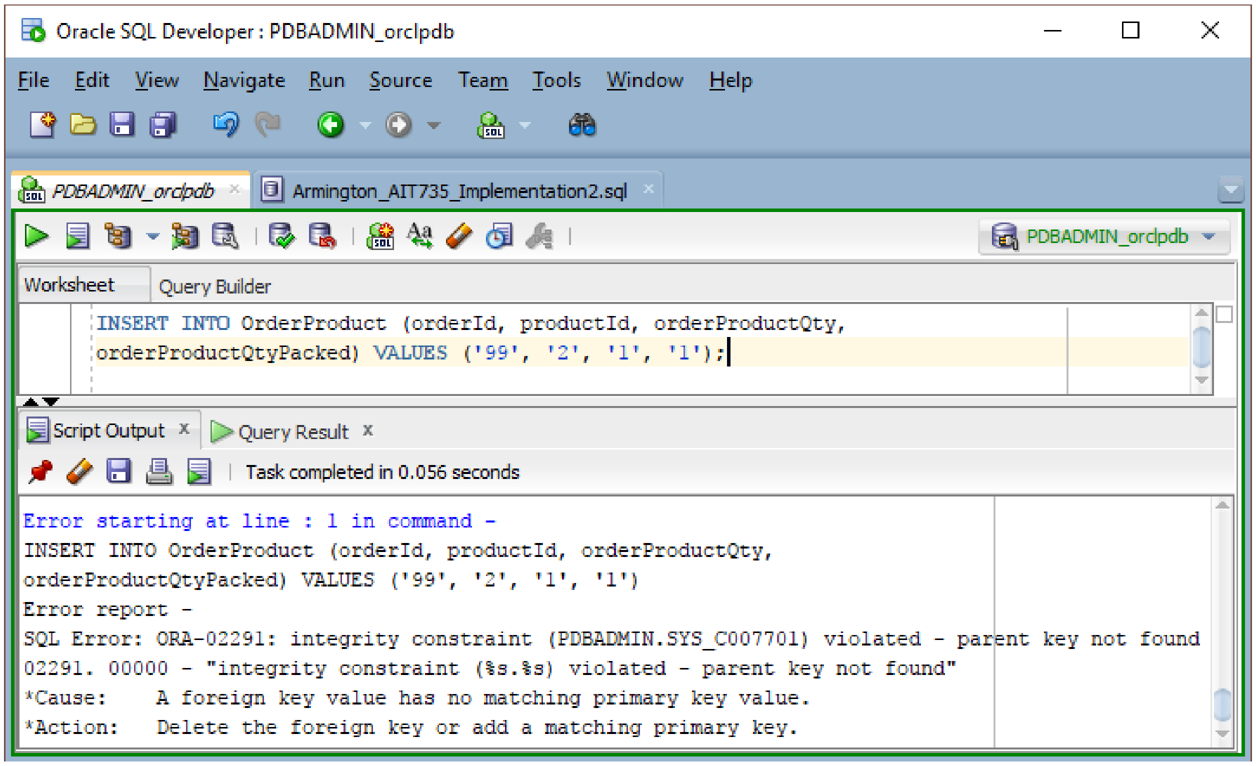

Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt
INSERT INTO OrderProduct (orderId, productId, orderProductQt

SELECT * FROM OrderProduct;
```

All Rows Fetched: 11 in 0.009 seconds

	ORDERID	PRODUCTID	ORDERPRODUCTQTY	ORDERPRODUCTQTYPACKED
1	1	2	1	1
2	2	4	1	1
3	3	7	1	1
4	4	8	1	1
5	5	3	1	1
6	6	8	1	1
7	6	2	1	1
8	7	1	1	1
9	8	6	1	1
10	9	5	1	1
11	10	2	2	2

The screenshot above shows that the 11 `INSERT` statements above were successfully inserted into the OrderProduct table.


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Query Result
Task completed in 0.056 seconds

```
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
orderProductQtyPacked) VALUES ('99', '2', '1', '1');
```

```
Error starting at line : 1 in command -
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
orderProductQtyPacked) VALUES ('99', '2', '1', '1')
Error report -
SQL Error: ORA-02291: integrity constraint (PDBADMIN.SYS_C007701) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

The screenshot above shows an unsuccessful `INSERT` statement into the OrderProduct table:

```
INSERT INTO OrderProduct (orderId, productId, orderProductQty,
orderProductQtyPacked) VALUES ('99', '2', '1', '1');
```

This table has a foreign key constraint on the order ID column (orderId) to ensure that all inserted order products reference a valid order ID in the ABCOrder table. At this point, the ABCOrder table only contains 11 rows – with order IDs 1-11. The test case above attempts to insert an order product with an invalid order ID (99) so the insert fails.

Sample Users, Profiles, and Roles

ABC employees must log in with a registered username and password in order to access the database. Users are granted privileges (through roles) and assigned limits to database resources (through profiles) based on their departments and job duties. All usernames, passwords, roles, and privileges are stored in Oracle's data dictionary.

Method Note: all users (with the exception of user `c##bgartrell`), profiles, and roles are local because they are applicable to the orclpdb PDB. Creating local users, profiles, and roles requires connecting as a user with appropriate privileges (`PDBADMIN`) in the PDB:

`connect PDBADMIN/Adm1nT3st@localhost:1521/orclpdb`.

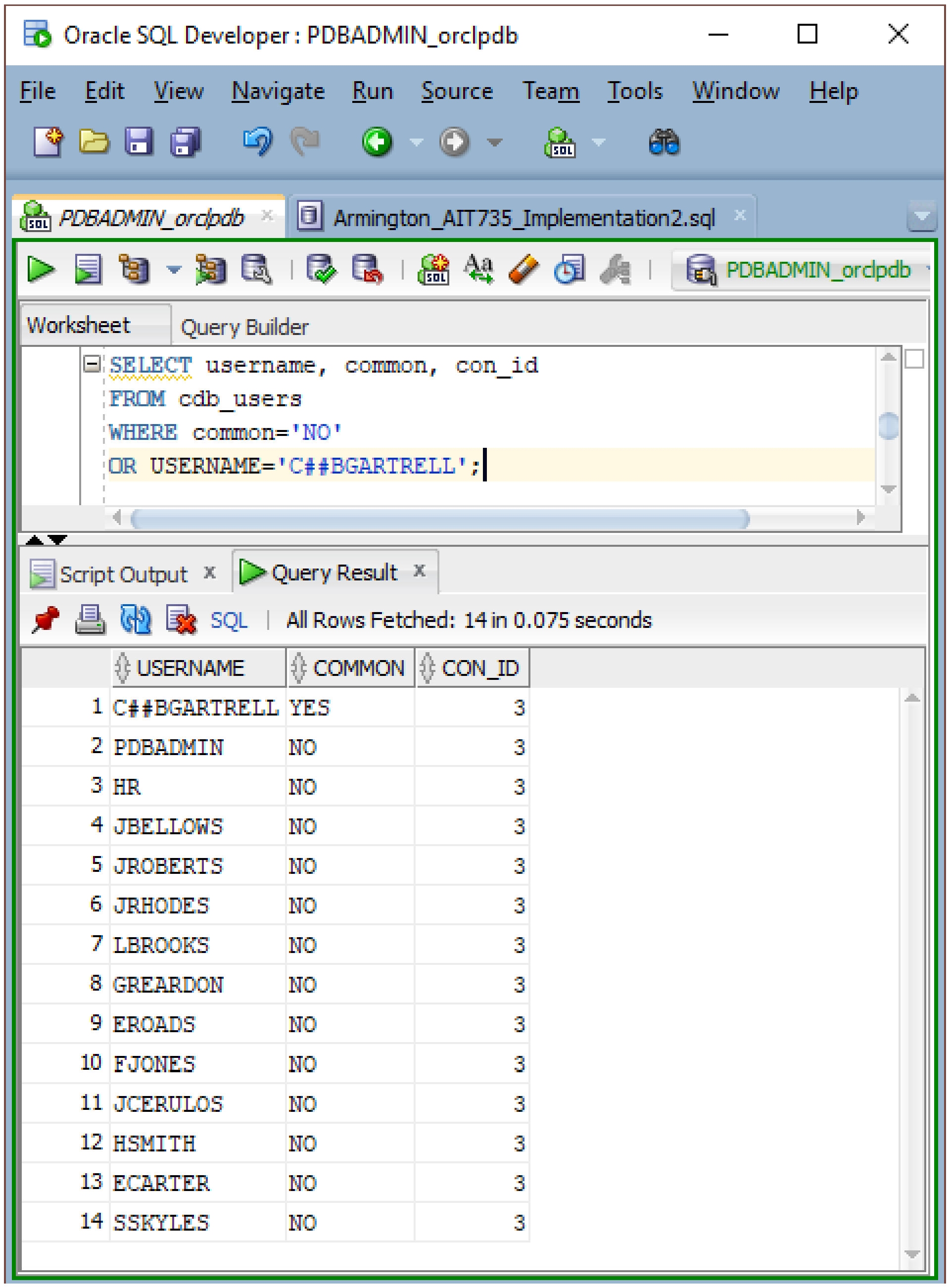
Users

A local user is created for each sample employee inserted into the Employee table, with the exception of `c##bgartrell`, who is a common user. Bob is an employee in the Information

Technology department and he will be conducting all system backup and recovery tasks for ABC. Thus, to be granted all required privileges needed to perform his job, Bob must be a common user. Common users must be created at the CDB level and names must be preceded by `c##`.

The naming convention for each user is as follows: the first letter of the employee's first name, followed by the employee's last name. The password for each user is the employee's first name (the first letter is capitalized), followed by the numbers 123. However, these passwords are expired for security purposes, implemented by the `PASSWORD EXPIRE` statement. When each user attempts to login for the first time, the system will prompt the user to change his/her password before connecting to the database. Note: users cannot connect to the database until the `CREATE SESSION` privilege is granted (see Roles).

```
CREATE USER eroads
      IDENTIFIED BY Elliot123
      PASSWORD EXPIRE;
CREATE USER sskyles
      IDENTIFIED BY Sam123
      PASSWORD EXPIRE;
CREATE USER fjones
      IDENTIFIED BY Faye123
      PASSWORD EXPIRE;
CREATE USER ecarter
      IDENTIFIED BY Erin123
      PASSWORD EXPIRE;
CREATE USER jcerulos
      IDENTIFIED BY Justine123
      PASSWORD EXPIRE;
CREATE USER jbellows
      IDENTIFIED BY Josh123
      PASSWORD EXPIRE;
CREATE USER jroberts
      IDENTIFIED BY Julia123
      PASSWORD EXPIRE;
CREATE USER jrhodes
      IDENTIFIED BY Jack123
      PASSWORD EXPIRE;
CREATE USER hsmith
      IDENTIFIED BY Holly123
      PASSWORD EXPIRE;
CREATE USER greardon
      IDENTIFIED BY Greg123
      PASSWORD EXPIRE;
CREATE USER lbrooks
      IDENTIFIED BY Lucas123
      PASSWORD EXPIRE;
/* remember, common user must be created at CDB level: */
CREATE USER c##bgartrell
      IDENTIFIED BY Bob123
      PASSWORD EXPIRE;
```


Oracle SQL Developer : PDBADMIN_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
Worksheet
Query Builder
Script Output
Query Result
SQL
All Rows Fetched: 14 in 0.075 seconds

```sql
SELECT username, common, con_id
FROM cdb_users
WHERE common='NO'
OR USERNAME='C##BGARTRELL';
```

	USERNAME	COMMON	CON_ID
1	C##BGARTRELL	YES	3
2	PDBADMIN	NO	3
3	HR	NO	3
4	JBELLOWS	NO	3
5	JROBERTS	NO	3
6	JRHODES	NO	3
7	LBROOKS	NO	3
8	GREARDON	NO	3
9	EROADS	NO	3
10	FJONES	NO	3
11	JCERULOS	NO	3
12	HSMITH	NO	3
13	ECARTER	NO	3
14	SSKYLES	NO	3

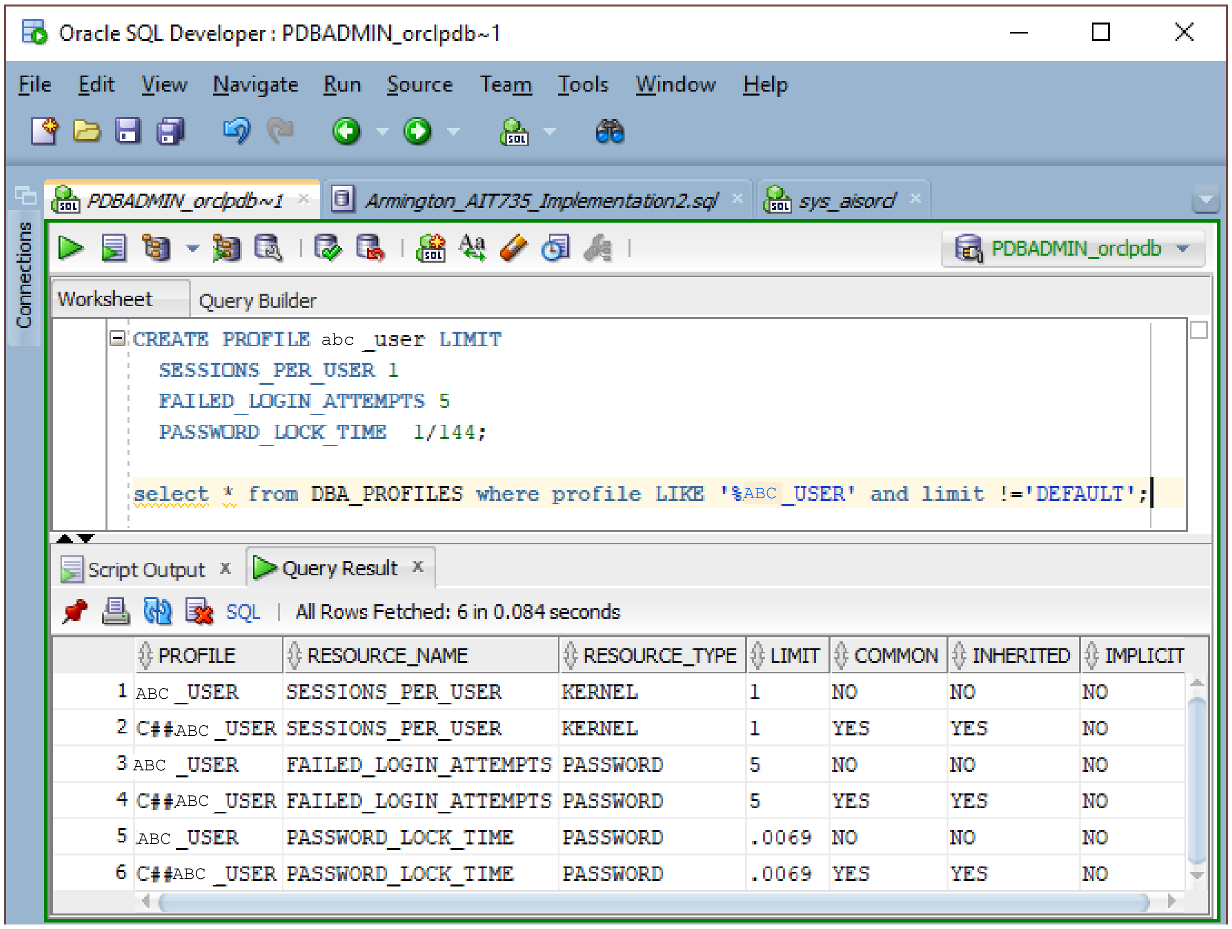
The screenshot above shows a `SELECT` query on the `cdb_users` table. It shows that the 11 users were successfully created as local users in the PDB, and that the user `c##bgartrell` was successfully created as a common user.

Note: The admin user `PDBADMIN` was created when the orclpdb PDB was created. This user was granted the predefined DBA role and was used to create all objects in the PDB database. This admin user has admin privileges in the PDB, but not in the CDB. The `HR` user is a predefined sample user account that comes with Oracle.

Profiles

Profiles are used to place limits on database resources. All local users are assigned the `abc_user` profile and the common user (`c##bgartrell`) is assigned the `c##abc_user` profile. These profiles limit assigned users to 1 concurrent session (`SESSIONS_PER_USER` parameter), which prevents employees from using the same account to connect to the database from multiple computers simultaneously. The profiles also limit the number of consecutive failed login attempts to 5 (`FAILED_LOGIN_ATTEMPTS` parameter) and specify the time period (1/144 days, or 10 minutes) that the account will be locked for if a user exceeds the maximum number of allowed failed attempts (`PASSWORD_LOCK_TIME` parameter). These profiles improve the security of ABC Health’s Product Management System.

```
CREATE PROFILE abc_user LIMIT
      SESSIONS_PER_USER     1
      FAILED_LOGIN_ATTEMPTS 5
      PASSWORD_LOCK_TIME    1/144;

/* Note: common profile c##abc_user must be created at CDB level! */

CREATE PROFILE c##abc_user LIMIT
      SESSIONS_PER_USER     1
      FAILED_LOGIN_ATTEMPTS 5
      PASSWORD_LOCK_TIME    1/144;
```



```
CREATE PROFILE abc_user LIMIT
  SESSIONS_PER_USER 1
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LOCK_TIME  1/144;

select * from DBA_PROFILES where profile LIKE '%ABC_USER' and limit !='DEFAULT';
```

All Rows Fetched: 6 in 0.084 seconds

	PROFILE	RESOURCE_NAME	RESOURCE_TYPE	LIMIT	COMMON	INHERITED	IMPLICIT
1	ABC_USER	SESSIONS_PER_USER	KERNEL	1	NO	NO	NO
2	C##ABC_USER	SESSIONS_PER_USER	KERNEL	1	YES	YES	NO
3	ABC_USER	FAILED_LOGIN_ATTEMPTS	PASSWORD	5	NO	NO	NO
4	C##ABC_USER	FAILED_LOGIN_ATTEMPTS	PASSWORD	5	YES	YES	NO
5	ABC_USER	PASSWORD_LOCK_TIME	PASSWORD	.0069	NO	NO	NO
6	C##ABC_USER	PASSWORD_LOCK_TIME	PASSWORD	.0069	YES	YES	NO

The screenshot above shows a `SELECT` query on the `DBA_PROFILES` table. It shows that the `abc_user` and `c##abc_user` profiles were successfully compiled with the aforementioned resource parameters.

The `abc_user` profile is assigned to employees by using the following SQL. Remember, this profile is not assigned to `c##bgartrell` because this is a common user.

```
ALTER USER eroads
      PROFILE abc_user;
ALTER USER sskyles
      PROFILE abc_user;
ALTER USER fjones
      PROFILE abc_user;
ALTER USER ecarter
      PROFILE abc_user;
ALTER USER jcerulos
      PROFILE abc_user;
ALTER USER jbellows
      PROFILE abc_user;
ALTER USER jroberts
      PROFILE abc_user;
ALTER USER jrhodes
      PROFILE abc_user;
ALTER USER hsmith
      PROFILE abc_user;
ALTER USER greardon
      PROFILE abc_user;
ALTER USER lbrooks
      PROFILE abc_user;
```

The `c##abc_user` profile is assigned to the common user – at the CDB level – by using the following SQL:

```
ALTER USER c##bgartrell
      PROFILE c##abc_user;
```

Roles

Privileges enable users to access or interact with objects in the database. Privileges can be assigned to each individual user, or they can be assigned to roles, which can then be assigned to users. In sum, roles are used to assign a set of privileges to users.

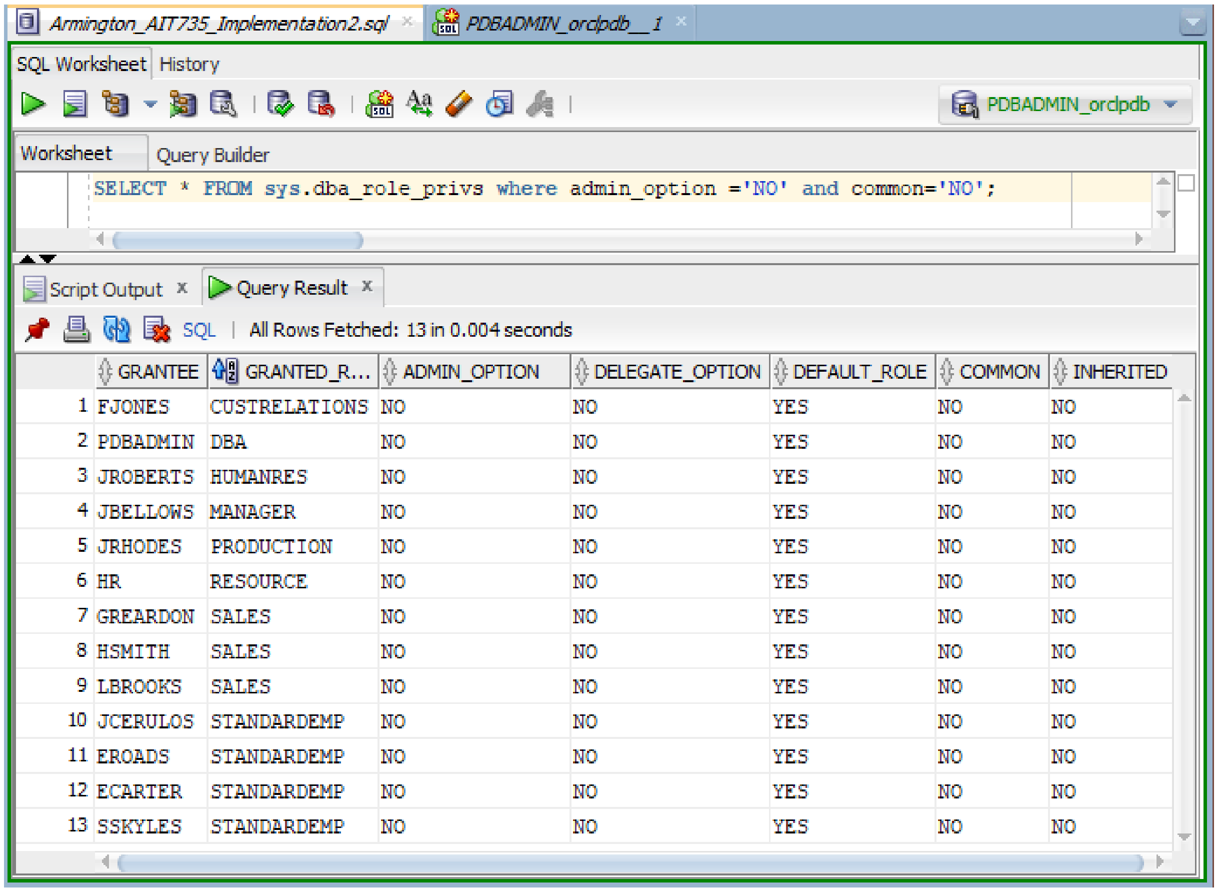
Privileges are assigned based on the principle of least privilege. In other words, users are only granted the privileges needed to complete their job functions. For example, a manager is granted more privileges than a standard employee because s/he oversees all business processes. Furthermore, an employee in the Human Resources department is granted the privilege to view the Employee table because s/he needs employees' social security numbers to complete job duties, whereas an employee in the Sales department does not. Therefore, all users, with the exception of `c##bgartrell`, are assigned an applicable role based on their department and job function. There are 6 roles in ABC Health's Product Management System: custrelations, humanres, production, sales, manager, and standardemp.

```
CREATE ROLE custrelations; --customer relations role
```

```
     GRANT CREATE SESSION TO custrelations;
CREATE ROLE humanres; --hr role
     GRANT CREATE SESSION TO humanres;
     GRANT SELECT, INSERT, UPDATE ON Employee TO humanres;
     -- allows user to select, insert, or update Employee table
CREATE ROLE production; --production role
     GRANT CREATE SESSION TO production;
CREATE ROLE sales; --sales role
     GRANT CREATE SESSION TO sales;
CREATE ROLE manager; --manager role
     GRANT CREATE SESSION TO manager;
     GRANT ALTER USER TO manager;
     /* allows user to change another user’s password if s/he gets
     locked out of system */
     GRANT SELECT, UPDATE ON Product TO manager;
     --allows user to select or update Product table
CREATE ROLE standardemp; --standardemp role
     GRANT CREATE SESSION TO standardemp;
```

All roles are granted the `CREATE SESSION` privilege, which enables users to connect to the database. Roles are granted to users with the following SQL:

```
GRANT standardemp TO eroads;
GRANT standardemp TO sskyles;
GRANT custrelations TO fjones;
GRANT standardemp TO ecarter;
GRANT standardemp TO jcerulos;
GRANT manager TO jbellows;
GRANT humanres TO jroberts;
GRANT production TO jrhodes;
GRANT sales TO hsmith;
GRANT sales TO greardon;
GRANT sales TO lbrooks;
```



```
SELECT * FROM sys.dba_role_privs where admin_option ='NO' and common='NO';
```

All Rows Fetched: 13 in 0.004 seconds

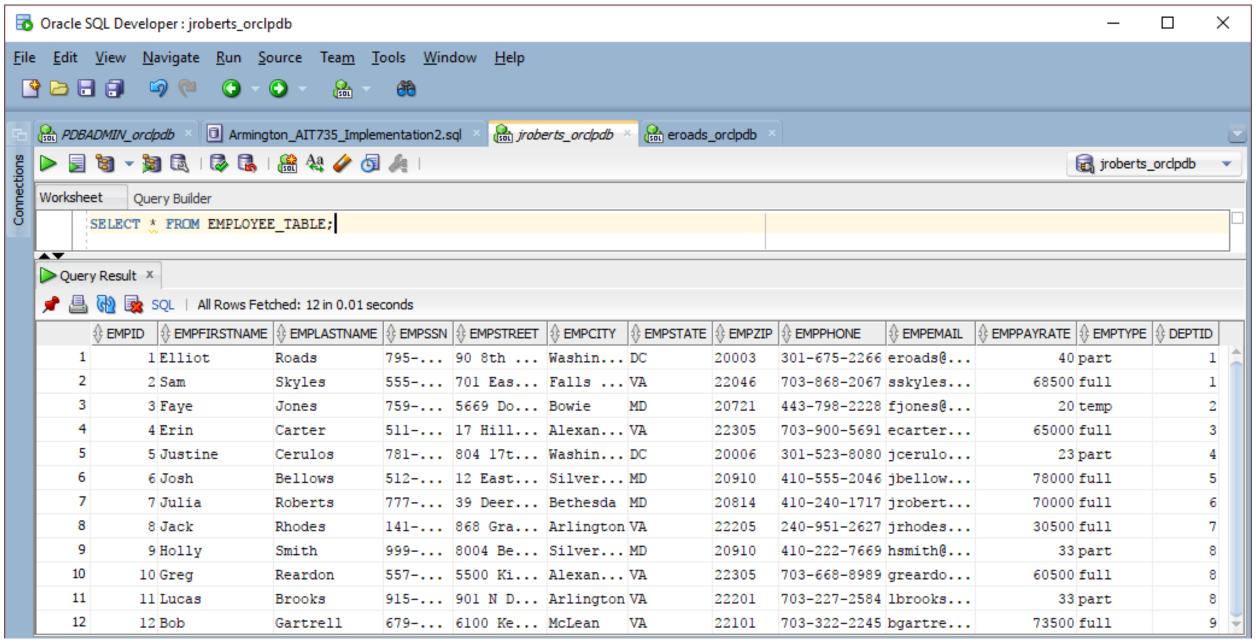
	GRANTEE	GRANTED_R...	ADMIN_OPTION	DELEGATE_OPTION	DEFAULT_ROLE	COMMON	INHERITED
1	FJONES	CUSTRELATIONS	NO	NO	YES	NO	NO
2	PDBADMIN	DBA	NO	NO	YES	NO	NO
3	JROBERTS	HUMANRES	NO	NO	YES	NO	NO
4	JBELLOWS	MANAGER	NO	NO	YES	NO	NO
5	JRHODES	PRODUCTION	NO	NO	YES	NO	NO
6	HR	RESOURCE	NO	NO	YES	NO	NO
7	GREARDON	SALES	NO	NO	YES	NO	NO
8	HSMITH	SALES	NO	NO	YES	NO	NO
9	LBROOKS	SALES	NO	NO	YES	NO	NO
10	JCERULOS	STANDARDEMP	NO	NO	YES	NO	NO
11	EROADS	STANDARDEMP	NO	NO	YES	NO	NO
12	ECARTER	STANDARDEMP	NO	NO	YES	NO	NO
13	SSKYLES	STANDARDEMP	NO	NO	YES	NO	NO

The screenshot above shows a `SELECT` query from the `DBA_ROLE_PRIVS` table, which belongs to the `SYS` user. It shows that the created roles were successfully granted to the respective local users.

In order for users who are assigned roles granted the privilege to select a table, a public synonym must be created for the table:

```
CREATE PUBLIC SYNONYM employee_table FOR PDBADMIN.employee;
CREATE PUBLIC SYNONYM product_table FOR PDBADMIN.product;
```

After these public synonyms are created, users assigned the roles can view the tables. For example, user `jroberts`, assigned the `humanres` role, can view the Employee table (see screenshot below).

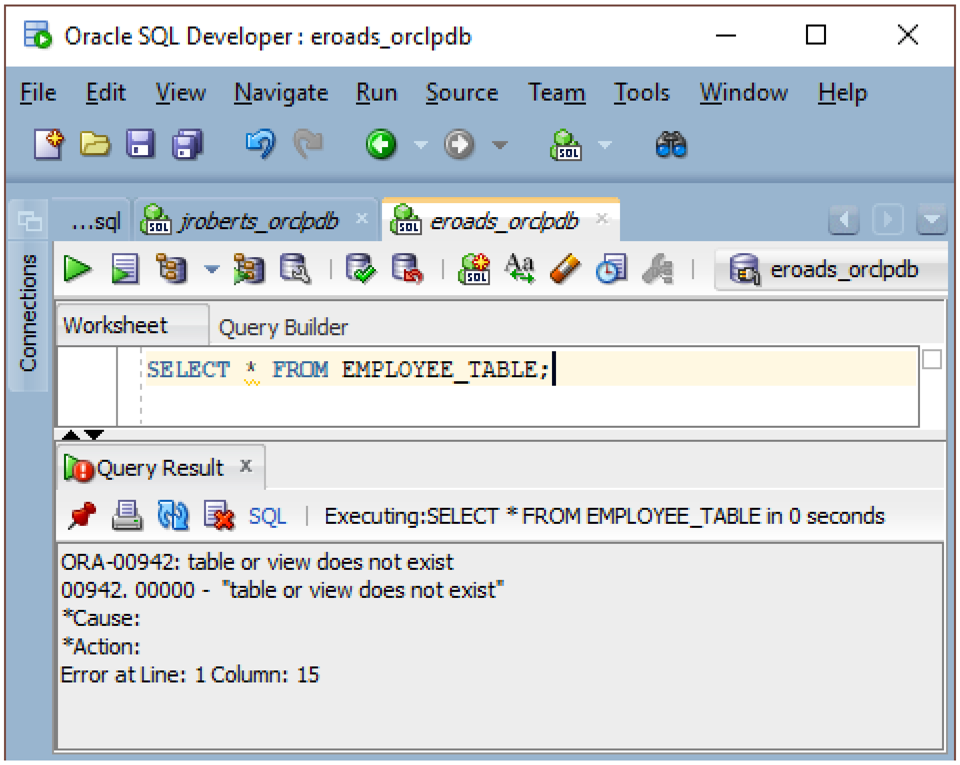

Oracle SQL Developer : jroberts_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
jroberts_orclpdb
eroads_orclpdb
Worksheet
Query Builder
Query Result
SQL

```
SELECT * FROM EMPLOYEE_TABLE;
```

All Rows Fetched: 12 in 0.01 seconds

	EMPID	EMPFIRSTNAME	EMPLASTNAME	EMPSSN	EMPSTREET	EMPCITY	EMPSTATE	EMPZIP	EMPPHONE	EMPEMAIL	EMPPAYRATE	EMPTYPE	DEPTID
1	1	Elliot	Roads	795-...	90 8th ...	Washin...	DC	20003	301-675-2266	eroads@...	40	part	1
2	2	Sam	Skyles	555-...	701 Eas...	Falls ...	VA	22046	703-868-2067	sskyles...	68500	full	1
3	3	Faye	Jones	759-...	5669 Do...	Bowie	MD	20721	443-798-2228	fjones@...	20	temp	2
4	4	Erin	Carter	511-...	17 Hill...	Alexan...	VA	22305	703-900-5691	ecarter...	65000	full	3
5	5	Justine	Cerulos	781-...	804 17t...	Washin...	DC	20006	301-523-8080	jcerulo...	23	part	4
6	6	Josh	Bellows	512-...	12 East...	Silver...	MD	20910	410-555-2046	jbellow...	78000	full	5
7	7	Julia	Roberts	777-...	39 Deer...	Bethesda	MD	20814	410-240-1717	jrobert...	70000	full	6
8	8	Jack	Rhodes	141-...	868 Gra...	Arlington	VA	22205	240-951-2627	jrhodes...	30500	full	7
9	9	Holly	Smith	999-...	8004 Be...	Silver...	MD	20910	410-222-7669	hsmith@...	33	part	8
10	10	Greg	Reardon	557-...	5500 Ki...	Alexan...	VA	22305	703-668-8989	greardo...	60500	full	8
11	11	Lucas	Brooks	915-...	901 N D...	Arlington	VA	22201	703-227-2584	lbrooks...	33	part	8
12	12	Bob	Gartrell	679-...	6100 Ke...	McLean	VA	22101	703-322-2245	bgartre...	73500	full	9

However, users who are not assigned the `humanres` role (e.g. user `eroads`, an employee in the Accounting department) cannot view the Employee table (see screenshot below).


Oracle SQL Developer : eroads_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
...sql
jroberts_orclpdb
eroads_orclpdb
Connections
Worksheet
Query Builder
Query Result
SQL

```
SELECT * FROM EMPLOYEE_TABLE;
```

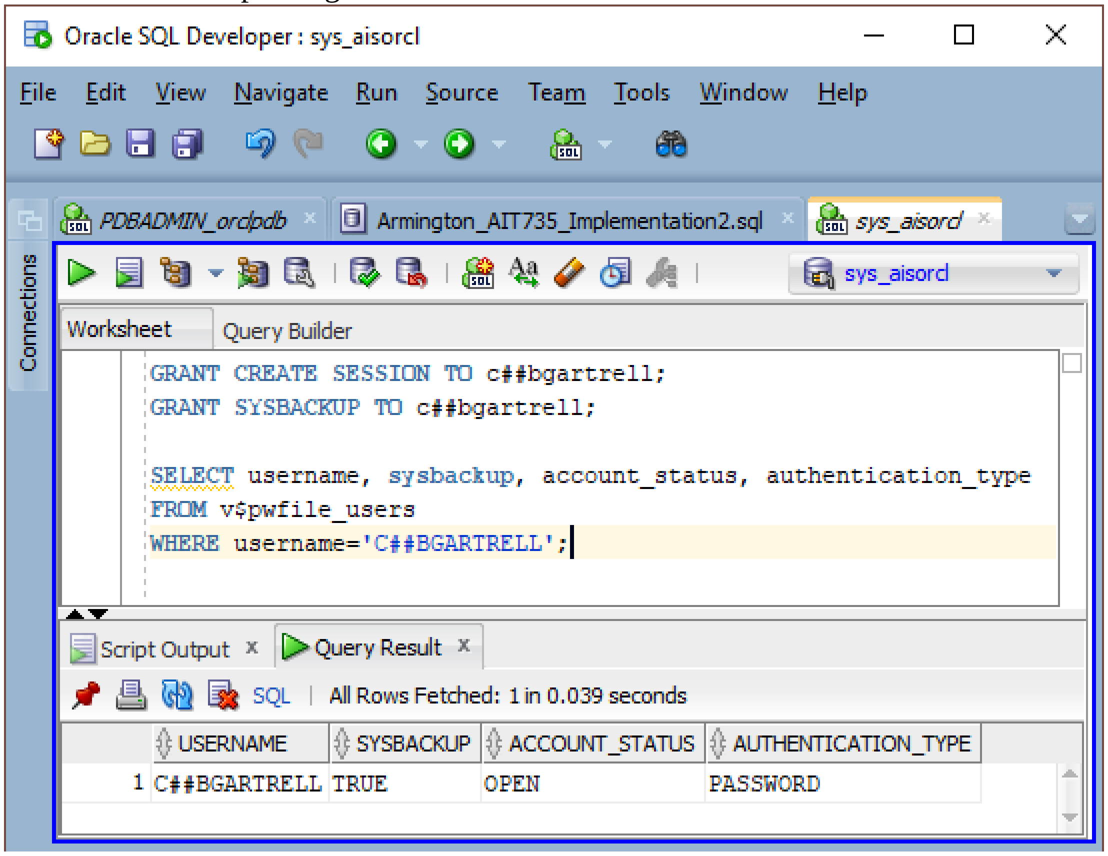
Executing:SELECT * FROM EMPLOYEE_TABLE in 0 seconds

```
ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
Error at Line: 1 Column: 15
```

Since `c##bgartrell` is a common user, privileges must be granted at the CDB level, not at the PDB level like with local users. Furthermore, the `abc_user` profile and roles cannot be assigned to `c##bgartrell` since they are specific to the orclpdb PDB. Privileges will be granted to Bob directly, not through a role.

```
GRANT CREATE SESSION TO c##bgartrell;
GRANT SYSBACKUP TO c##bgartrell;
```

The `CREATE SESSION` and `SYSBACKUP` privileges are granted to `c##bgartrell` so that he can connect to the CDB and perform system backup and recovery operations, respectively. Note that Bob will only be performing system backup and recovery tasks at ABC so he does not need the `SET CONTAINER` privilege to connect to the PDB.


Oracle SQL Developer : sys_aisorcl
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb
Armington_AIT735_Implementation2.sql
sys_aisorcl
Connections
Worksheet
Query Builder
Script Output
Query Result
SQL

```
GRANT CREATE SESSION TO c##bgartrell;
GRANT SYSBACKUP TO c##bgartrell;

SELECT username, sysbackup, account_status, authentication_type
FROM v$pwfile_users
WHERE username='C##BGARTRELL';
```

All Rows Fetched: 1 in 0.039 seconds

	USERNAME	SYSBACKUP	ACCOUNT_STATUS	AUTHENTICATION_TYPE
1	C##BGARTRELL	TRUE	OPEN	PASSWORD

The screenshot above shows a `SELECT` query from the `v$pwfile_users` table. The query indicates that `c##bgartrell` was successfully granted the `SYSBACKUP` privilege. Note that the user performing the action is `SYS`, a user who has access to the CDB level.

Procedures/Triggers/Views

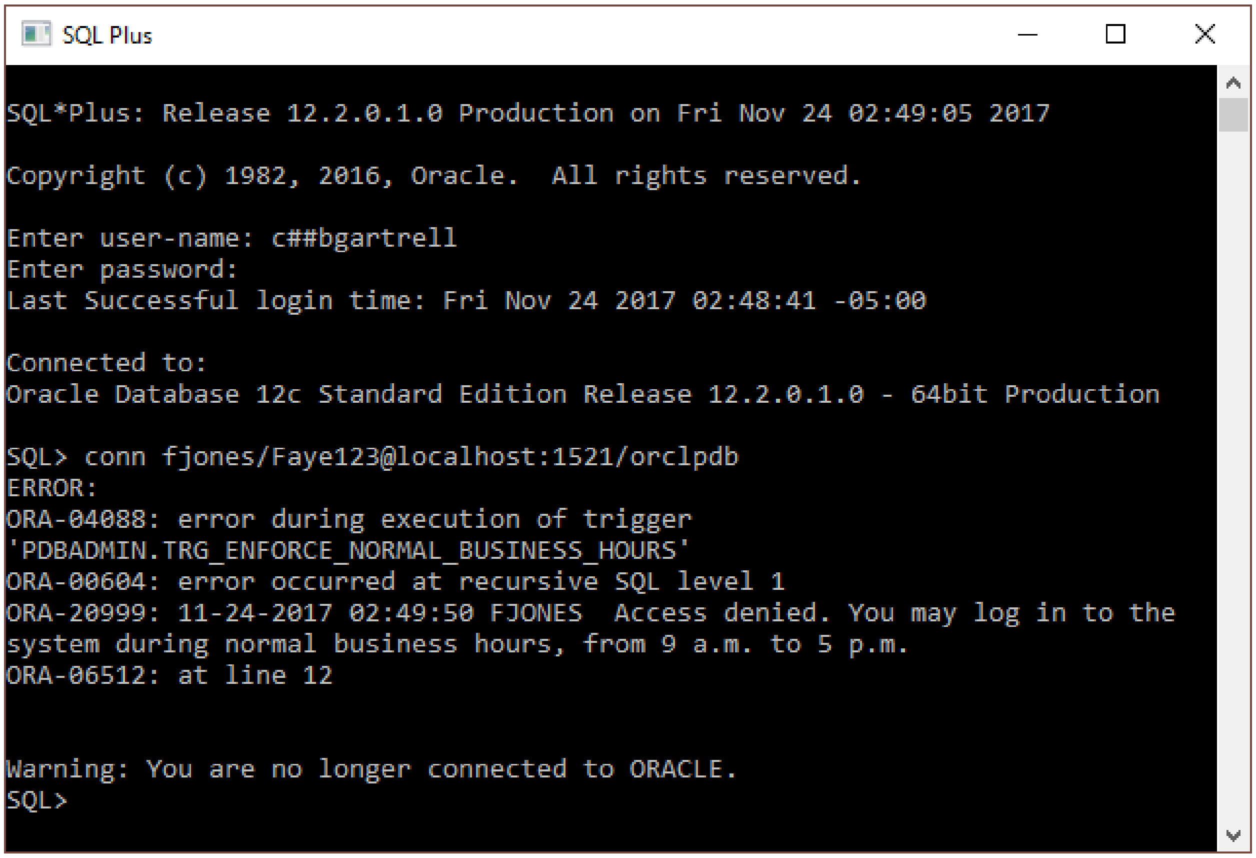
Before running procedures, run the following statement to display feedback:

```
set serveroutput on
```

FR-04

Requirement: System must restrict all employees from logging in to the system outside of normal business hours (9 a.m. to 5 p.m.), with the exception of the employee who conducts system backup and recovery tasks.

```
create or replace trigger trg_enforce_normal_business_hours
  after logon on database
DECLARE
 current_hour    pls_integer;
 restricted_user boolean;
 restricted_time boolean;
 v_currentUser varchar2(100);
BEGIN
  v_currentUser := USER;
  restricted_user := user = user;
  current_hour    := extract(hour from systimestamp);
  restricted_time := (current_hour < 9 or current_hour > 17);
  IF restricted_user AND restricted_time THEN
    rabce_application_error(-20999, TO_CHAR(SYSDATE, 'MM-DD-YYYY
HH24:MI:SS')
    || ' ' || v_currentUser || '  ' || 'Access denied. You may log in
to the
    system during normal business hours, from 9 a.m. to 5 p.m.');
  END IF;
END;
```



```
SQL*Plus: Release 12.2.0.1.0 Production on Fri Nov 24 02:49:05 2017

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Enter user-name: c##bgartrell
Enter password:
Last Successful login time: Fri Nov 24 2017 02:48:41 -05:00

Connected to:
Oracle Database 12c Standard Edition Release 12.2.0.1.0 - 64bit Production

SQL> conn fjones/Faye123@localhost:1521/orclpdb
ERROR:
ORA-04088: error during execution of trigger
'PDBADMIN.TRG_ENFORCE_NORMAL_BUSINESS_HOURS'
ORA-00604: error occurred at recursive SQL level 1
ORA-20999: 11-24-2017 02:49:50 FJONES  Access denied. You may log in to the
system during normal business hours, from 9 a.m. to 5 p.m.
ORA-06512: at line 12


Warning: You are no longer connected to ORACLE.
SQL>
```

The screenshot above shows that the trigger `trg_enforce_normal_business_hours` is working correctly. Common user `bgartrell` is able to login successfully at 2:48 a.m. However, user `fjones` is unable to login at 2:32 a.m. on November 24, 2017 because it is outside of normal business hours.

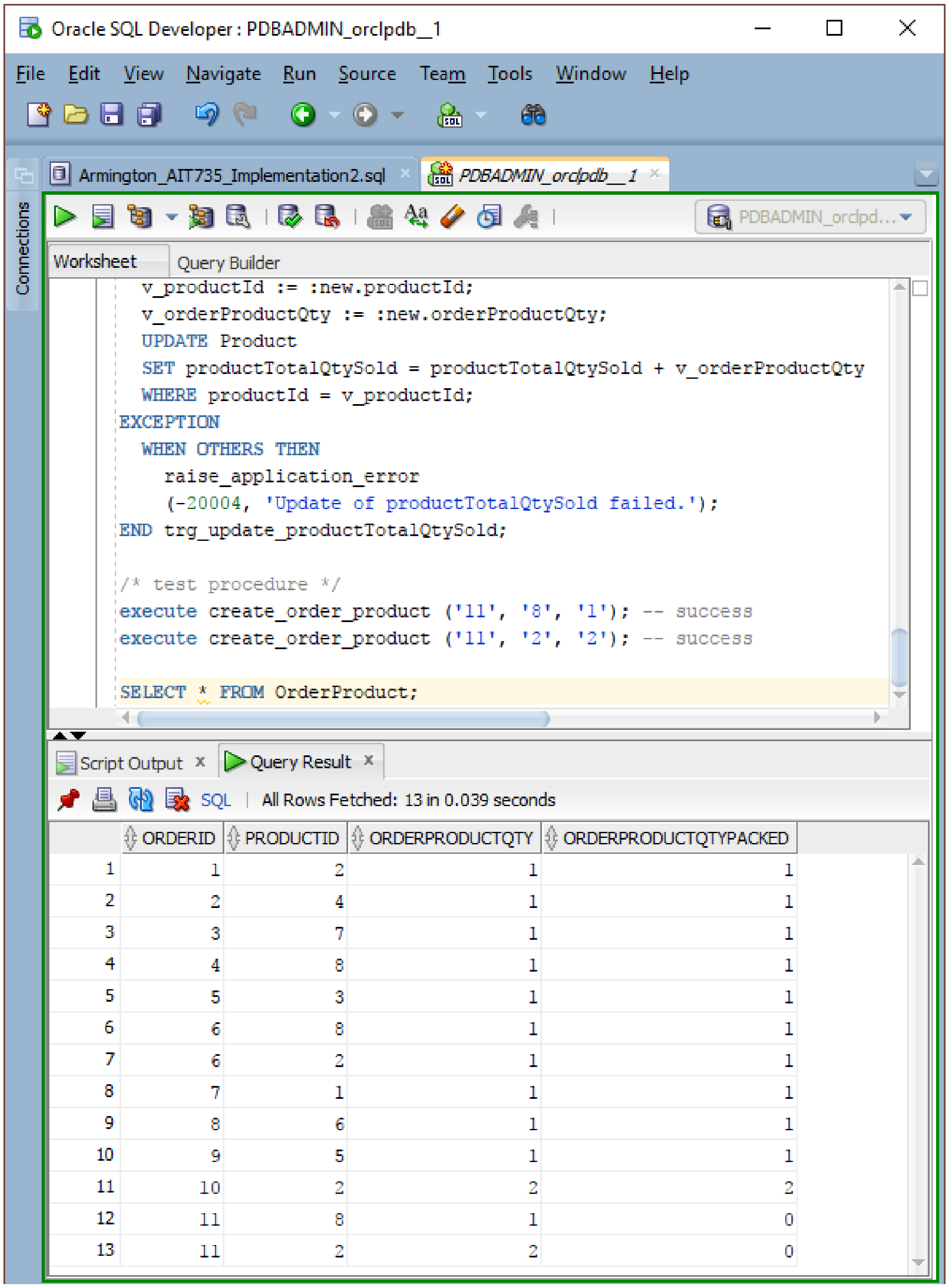
FR-10

Requirement: Given a valid order ID, product ID, and order product quantity, system must be able to create an order product, adding the product price to the order total (via trigger), and adding the order product quantity to the correct product's total quantity sold (via trigger).

```
Create or Replace Procedure create_orderProduct (v_orderId int,
v_productId int, v_orderProductQty int)
IS
      v_order_id int; --used to validate incoming orderId
      v_product_id int; --used to validate incoming productId
      v_errorCode number;
      v_errorMsg varchar2(200);
      v_currentUser varchar2(100);
BEGIN
      /* validate orderId input. If invalid, goes to exception */
      SELECT orderId
      INTO v_order_id
      FROM ABCOrder
      WHERE orderId = v_orderId;
      /* validate productId input */
      SELECT count(*)
      INTO v_product_id
      FROM Product
      WHERE productId = v_productId;
      IF v_product_id != 1 THEN
            DBMS_OUTPUT.PUT_LINE('Invalid productId. Insert failed.');
      ELSE
            BEGIN
            INSERT INTO OrderProduct (orderId, productId,
orderProductQty) VALUES (v_orderId, v_productId, v_orderProductQty);
            DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || '
rows.');
            COXXX; --transaction control
            END;
      END IF;
EXCEPTION
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Invalid orderId. Insert failed.');
      WHEN OTHERS THEN
            v_errorCode := SQLCODE;
            v_errorMsg := substr(SQLERRM,1,200);
            v_currentUser := USER;
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE) || v_currentUser ||
TO_CHAR(v_errorCode) || v_errorMsg || ' Insert failed.');
END;

/* TRIGGER 1 */ --validate orderProductQty input
Create or replace Trigger trg_orderProductQty_validate
BEFORE INSERT ON OrderProduct
For each row
BEGIN
```

```
     IF :new.orderProductQty <=0 THEN
          rabce_application_error (-20001, 'Invalid orderProductQty.
Insert failed.');
     END IF;
END trg_orderProductQty_validate;

/* TRIGGER 2 */ --check to make sure orderPackDate is null
Create or Replace Trigger trg_insert_orderProduct_validate
BEFORE INSERT ON OrderProduct
For each row
DECLARE
     v_orderId int;
     v_orderPackDate date;
BEGIN
     v_orderId := :new.orderId;
     SELECT orderPackDate
     INTO v_orderPackDate
     FROM ABCOrder
     WHERE orderId = v_orderId;
     IF v_orderPackDate IS NOT NULL THEN
          rabce_application_error (-20002, 'Cannot add OrderProduct.
Order has already been completely packed. Insert failed.');
     END IF;
END trg_insert_orderProduct_validate;

/* TRIGGER 3 */ --update orderTotal
Create or Replace Trigger trg_update_orderTotal
AFTER INSERT ON OrderProduct
For each row
DECLARE
     v_orderId int;
     v_productId int;
     v_productPrice number(20,2);
     v_orderProductQty int;
BEGIN
     v_orderId := :new.orderId;
     v_productId := :new.productId;
     v_orderProductQty := :new.orderProductQty;
     SELECT productPrice
     INTO v_productPrice
     FROM Product
     WHERE productId = v_productId;
     UPDATE ABCOrder
     SET orderTotal = orderTotal + (v_productPrice *
v_orderProductQty)
     WHERE orderId = v_orderId;
EXCEPTION
     WHEN OTHERS THEN
          rabce_application_error
          (-20003, 'Update of orderTotal failed.');
END trg_update_orderTotal;
```

```
/* TRIGGER 4 */ --update productTotalQtySold
Create or Replace Trigger trg_update_productTotalQtySold
AFTER INSERT ON OrderProduct
For each row
DECLARE
	v_productId int;
	v_orderProductQty int;
BEGIN
	v_productId := :new.productId;
	v_orderProductQty := :new.orderProductQty;
	UPDATE Product
	SET productTotalQtySold = productTotalQtySold + v_orderProductQty
	WHERE productId = v_productId;
EXCEPTION
	WHEN OTHERS THEN
		rabce_application_error
		(-20004, 'Update of productTotalQtySold failed.');
END trg_update_productTotalQtySold;

/* test procedure */
execute create_order_product ('11', '8', '1'); --success
execute create_order_product ('11', '2', '2'); --success
execute create_order_product ('11', '2', '0'); --failure
execute create_order_product ('50', '1', '1'); --failure
execute create_order_product ('11', '50', '1'); --failure
execute create_order_product ('10', '1', '1'); --failure
```

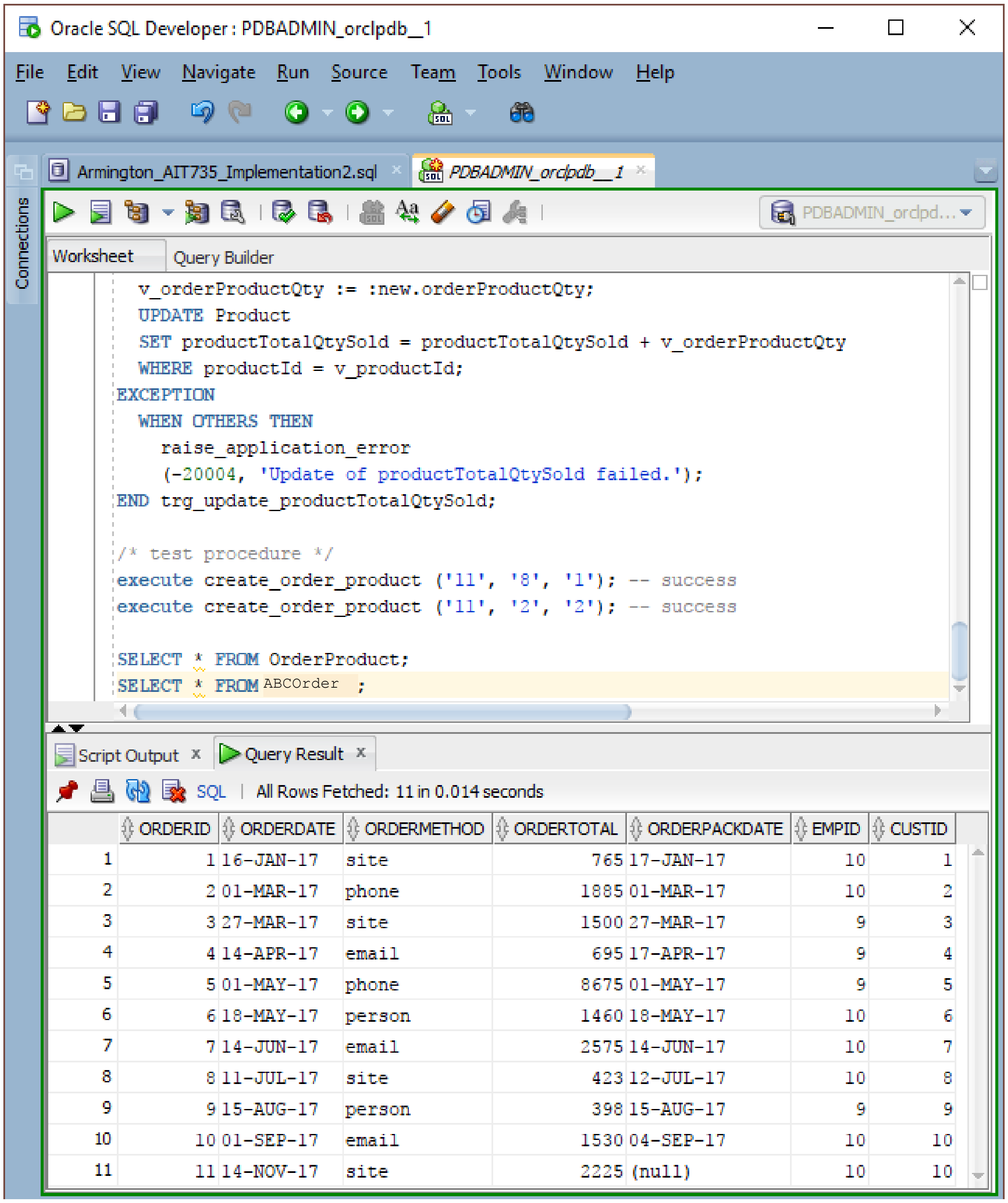

Oracle SQL Developer : PDBADMIN_orclpdb__1
File Edit View Navigate Run Source Team Tools Window Help
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
PDBADMIN_orclpd...
Connections
Worksheet
Query Builder
Script Output
Query Result
SQL

```
    v_productId := :new.productId;
    v_orderProductQty := :new.orderProductQty;
    UPDATE Product
    SET productTotalQtySold = productTotalQtySold + v_orderProductQty
    WHERE productId = v_productId;
EXCEPTION
    WHEN OTHERS THEN
      raise_application_error
      (-20004, 'Update of productTotalQtySold failed.');
END trg_update_productTotalQtySold;

/* test procedure */
execute create_order_product ('11', '8', '1'); -- success
execute create_order_product ('11', '2', '2'); -- success

SELECT * FROM OrderProduct;
```

All Rows Fetched: 13 in 0.039 seconds

	ORDERID	PRODUCTID	ORDERPRODUCTQTY	ORDERPRODUCTQTYPACKED
1	1	2	1	1
2	2	4	1	1
3	3	7	1	1
4	4	8	1	1
5	5	3	1	1
6	6	8	1	1
7	6	2	1	1
8	7	1	1	1
9	8	6	1	1
10	9	5	1	1
11	10	2	2	2
12	11	8	1	0
13	11	2	2	0

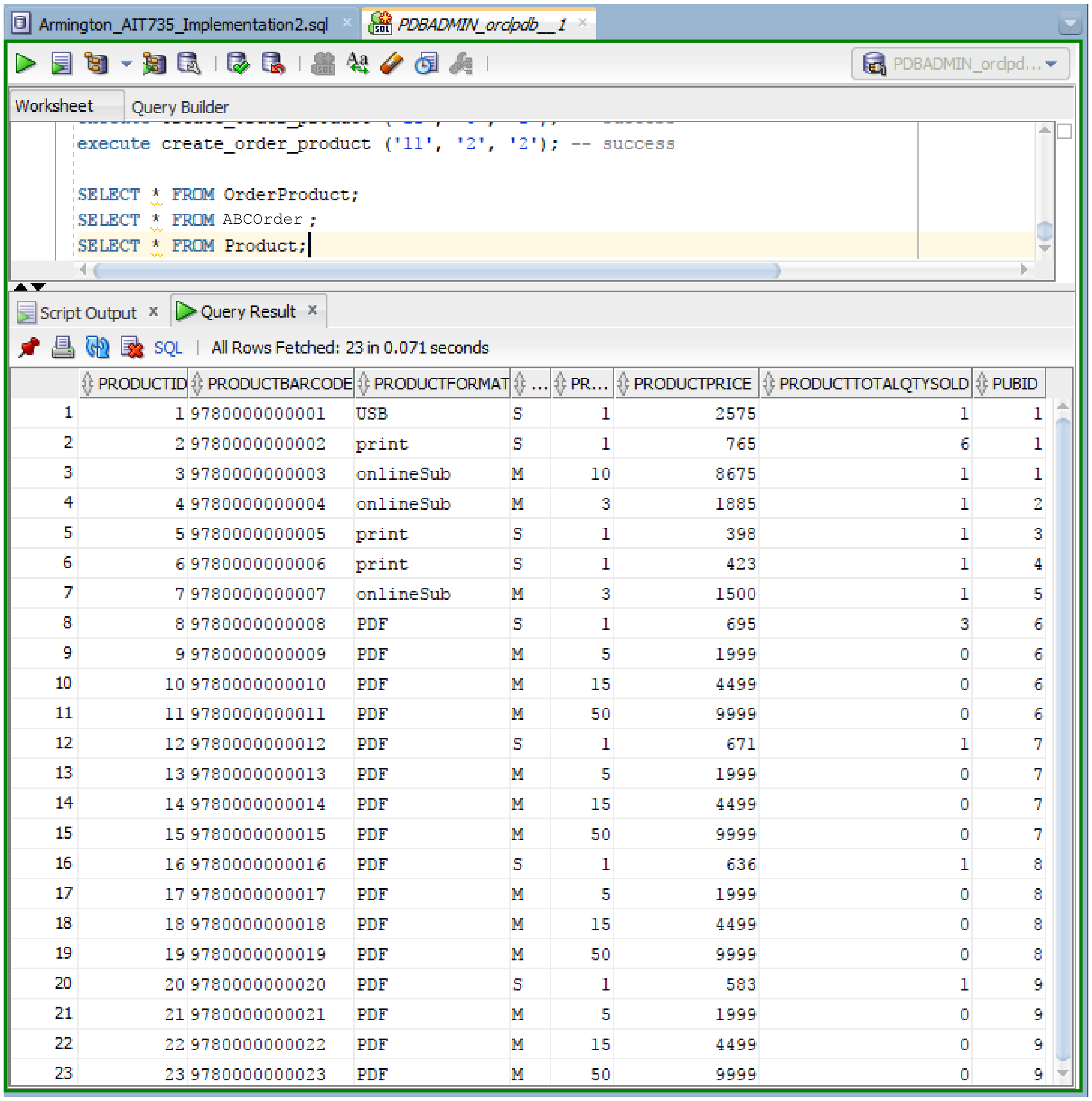
The screenshot above shows the result of successfully executing the procedure `create_order_product`. Both execute statements attempt to add order products for order ID 11. Per the screenshot, two rows have been added to the OrderProduct table for order ID 11.


Oracle SQL Developer : PDBADMIN_orclpdb_1
File Edit View Navigate Run Source Team Tools Window Help
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
Worksheet
Query Builder
Script Output
Query Result
SQL
All Rows Fetched: 11 in 0.014 seconds

```
  v_orderProductQty := :new.orderProductQty;
  UPDATE Product
  SET productTotalQtySold = productTotalQtySold + v_orderProductQty
  WHERE productId = v_productId;
EXCEPTION
  WHEN OTHERS THEN
    raise_application_error
    (-20004, 'Update of productTotalQtySold failed.');
END trg_update_productTotalQtySold;

/* test procedure */
execute create_order_product ('11', '8', '1'); -- success
execute create_order_product ('11', '2', '2'); -- success

SELECT * FROM OrderProduct;
SELECT * FROM ABCOrder  ;
```

	ORDERID	ORDERDATE	ORDERMETHOD	ORDERTOTAL	ORDERPACKDATE	EMPID	CUSTID
1	1	16-JAN-17	site	765	17-JAN-17	10	1
2	2	01-MAR-17	phone	1885	01-MAR-17	10	2
3	3	27-MAR-17	site	1500	27-MAR-17	9	3
4	4	14-APR-17	email	695	17-APR-17	9	4
5	5	01-MAY-17	phone	8675	01-MAY-17	9	5
6	6	18-MAY-17	person	1460	18-MAY-17	10	6
7	7	14-JUN-17	email	2575	14-JUN-17	10	7
8	8	11-JUL-17	site	423	12-JUL-17	10	8
9	9	15-AUG-17	person	398	15-AUG-17	9	9
10	10	01-SEP-17	email	1530	04-SEP-17	10	10
11	11	14-NOV-17	site	2225	(null)	10	10

The screenshot above shows the result of Trigger 3 (`trg_update_orderTotal`). After the two rows were inserted into the OrderProduct table using the `create_orderProduct` procedure, the trigger updated the total cost (orderTotal) of order ID 11 from 0 to 2225. This number was calculated by adding the price of each ordered product. The price of product ID 8 is 695 and the price of product ID 2 is 765. The customer ordered 1 unit of product ID 8 and 2 units of product ID 2. 695 + 1530 = 2225.

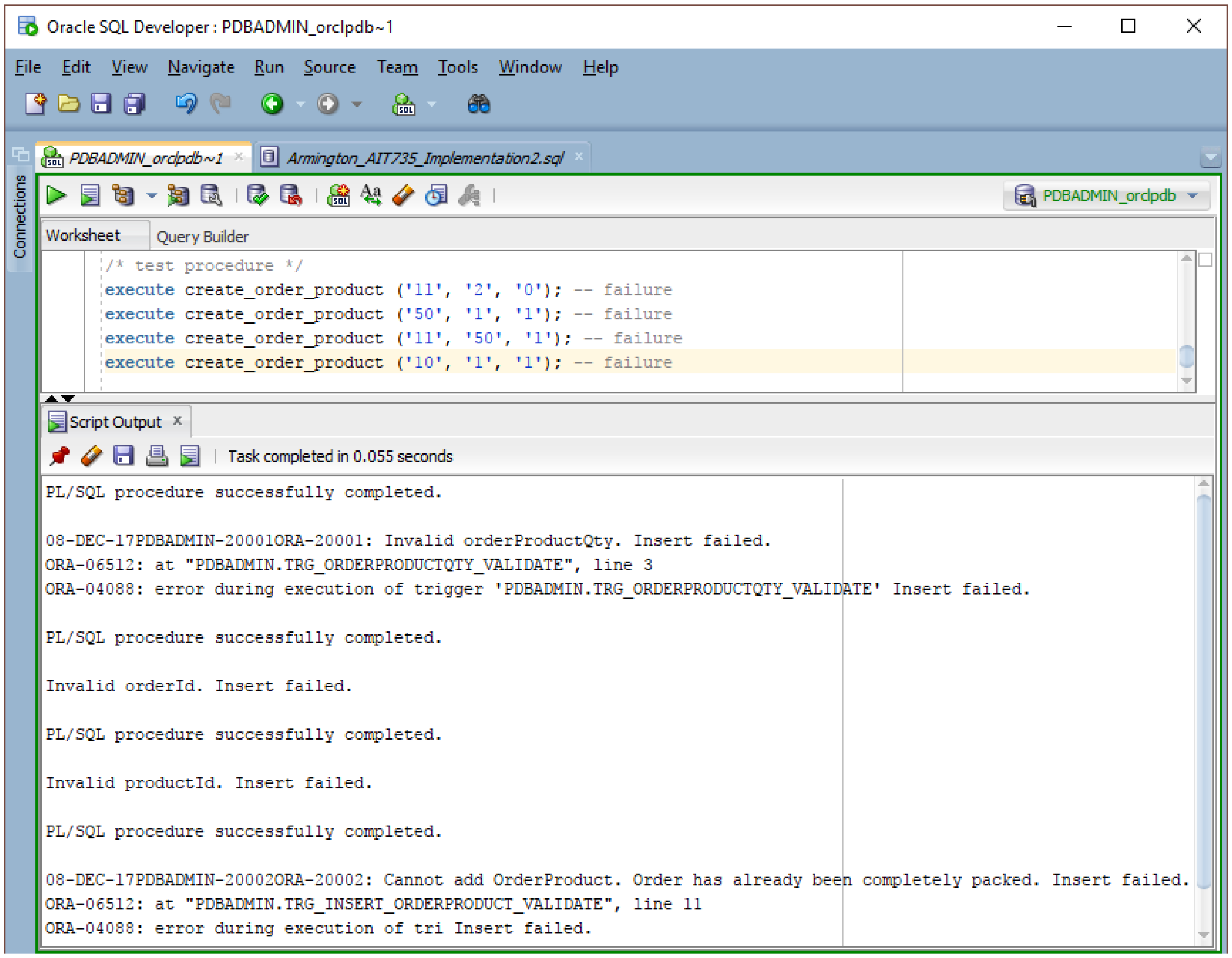


```
execute create_order_product ('11', '2', '2'); -- success

SELECT * FROM OrderProduct;
SELECT * FROM ABCOrder ;
SELECT * FROM Product;
```

All Rows Fetched: 23 in 0.071 seconds

	PRODUCTID	PRODUCTBARCODE	PRODUCTFORMAT	...	PR...	PRODUCTPRICE	PRODUCTTOTALQTYSOLD	PUBID
1	1	9780000000001	USB	S	1	2575	1	1
2	2	9780000000002	print	S	1	765	6	1
3	3	9780000000003	onlineSub	M	10	8675	1	1
4	4	9780000000004	onlineSub	M	3	1885	1	2
5	5	9780000000005	print	S	1	398	1	3
6	6	9780000000006	print	S	1	423	1	4
7	7	9780000000007	onlineSub	M	3	1500	1	5
8	8	9780000000008	PDF	S	1	695	3	6
9	9	9780000000009	PDF	M	5	1999	0	6
10	10	9780000000010	PDF	M	15	4499	0	6
11	11	9780000000011	PDF	M	50	9999	0	6
12	12	9780000000012	PDF	S	1	671	1	7
13	13	9780000000013	PDF	M	5	1999	0	7
14	14	9780000000014	PDF	M	15	4499	0	7
15	15	9780000000015	PDF	M	50	9999	0	7
16	16	9780000000016	PDF	S	1	636	1	8
17	17	9780000000017	PDF	M	5	1999	0	8
18	18	9780000000018	PDF	M	15	4499	0	8
19	19	9780000000019	PDF	M	50	9999	0	8
20	20	9780000000020	PDF	S	1	583	1	9
21	21	9780000000021	PDF	M	5	1999	0	9
22	22	9780000000022	PDF	M	15	4499	0	9
23	23	9780000000023	PDF	M	50	9999	0	9

The screenshot above shows the result of Trigger 4 (`trg_update_productTotalQtySold`). After the two rows were inserted into the OrderProduct table using the `create_orderProduct` procedure, the trigger updated the total quantities of each product sold (productTotalQtySold). The total quantity sold for product ID 2 updated from 4 to 6 and the total quantity sold for product ID 8 updated from 2 to 3.


Oracle SQL Developer : PDBADMIN_orclpdb~1
File Edit View Navigate Run Source Team Tools Window Help
PDBADMIN_orclpdb~1
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb
Worksheet
Query Builder
Script Output
Task completed in 0.055 seconds

```
/* test procedure */
execute create_order_product ('11', '2', '0'); -- failure
execute create_order_product ('50', '1', '1'); -- failure
execute create_order_product ('11', '50', '1'); -- failure
execute create_order_product ('10', '1', '1'); -- failure
```

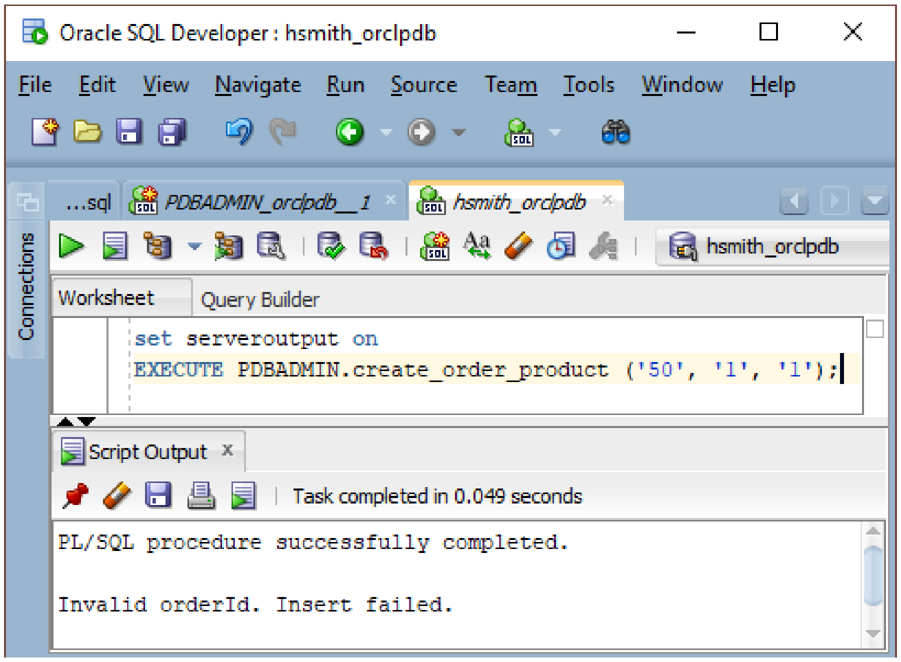
```
PL/SQL procedure successfully completed.

08-DEC-17PDBADMIN-20001ORA-20001: Invalid orderProductQty. Insert failed.
ORA-06512: at "PDBADMIN.TRG_ORDERPRODUCTQTY_VALIDATE", line 3
ORA-04088: error during execution of trigger 'PDBADMIN.TRG_ORDERPRODUCTQTY_VALIDATE' Insert failed.

PL/SQL procedure successfully completed.

Invalid orderId. Insert failed.

PL/SQL procedure successfully completed.

Invalid productId. Insert failed.

PL/SQL procedure successfully completed.

08-DEC-17PDBADMIN-20002ORA-20002: Cannot add OrderProduct. Order has already been completely packed. Insert failed.
ORA-06512: at "PDBADMIN.TRG_INSERT_ORDERPRODUCT_VALIDATE", line 11
ORA-04088: error during execution of tri Insert failed.
```

The screenshot above shows the result of executing the four failure test statements. The first statement attempts to add an invalid order product quantity (0) to an order. This invalid input is caught by Trigger 1 (`trg_orderProductQty_validate`), which checks to ensure that the input quantity is greater than 0. The second statement attempts to add an ordered product to an invalid order ID (50) and the third attempts to add an invalid product ID (50) to an order. Both of these errors are caught by the procedure `create_order_product`. The fourth failure test statement attempts to add an ordered product to an order that has already been packed completely (orderId 10). This is caught by Trigger 2 (`trg_insert_orderProduct_validate`), which checks to see if there is a value in the orderPackDate column of the ABCOrder table for the entered order ID.

This procedure should be used by employees in the Sales department. The privilege to execute this procedure and triggers can be granted to the sales role using the following code:

```
GRANT EXECUTE ON PDBADMIN.create_order_product TO sales;
```

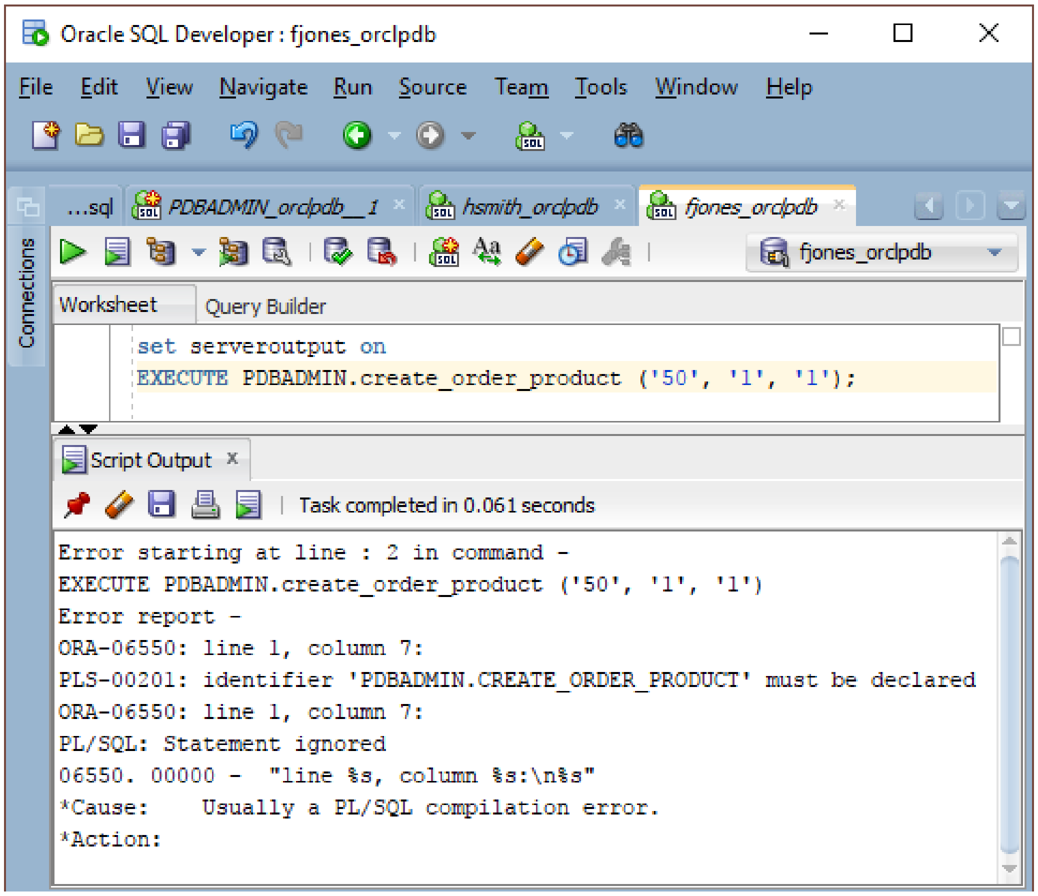
Logging in as an employee with the sales role (`hsmith`) and executing the `create_order_product` procedure proves that the sales employee is able to successfully call the procedure (see screenshot below).


Oracle SQL Developer : hsmith_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
...sql
PDBADMIN_orclpdb__1
hsmith_orclpdb
Worksheet
Query Builder
Script Output
Task completed in 0.049 seconds

```
set serveroutput on
EXECUTE PDBADMIN.create_order_product ('50', '1', '1');
```

```
PL/SQL procedure successfully completed.

Invalid orderId. Insert failed.
```

However, an employee in the Customer Relations department (`fjones`) is not able to call the procedure (see screenshot below).


Oracle SQL Developer : fjones_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
...sql
PDBADMIN_orclpdb__1
hsmith_orclpdb
fjones_orclpdb
Worksheet
Query Builder
Script Output
Task completed in 0.061 seconds

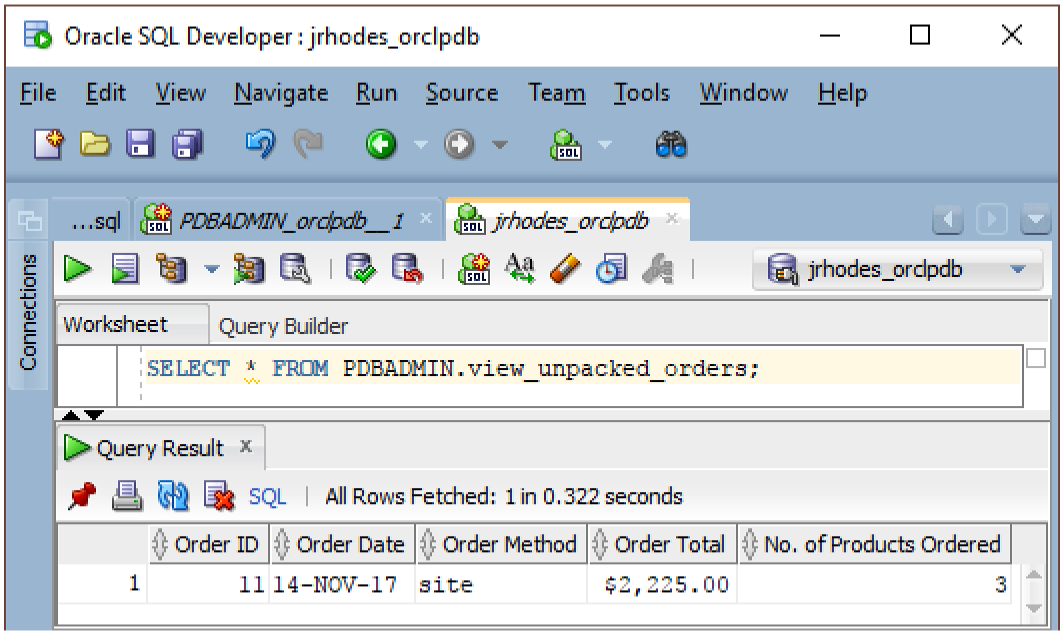
```
set serveroutput on
EXECUTE PDBADMIN.create_order_product ('50', '1', '1');
```

```
Error starting at line : 2 in command -
EXECUTE PDBADMIN.create_order_product ('50', '1', '1')
Error report -
ORA-06550: line 1, column 7:
PLS-00201: identifier 'PDBADMIN.CREATE_ORDER_PRODUCT' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

FR-11

Requirement: System must support a view which lists all orders that have not been completely packed, accessible by employees in the Production department.

```
CREATE OR REPLACE VIEW view_unpacked_orders("Order ID", "Order Date",
      "Order Method", "Order Total", "No. of Products Ordered") AS
SELECT ABCOrder.orderId, ABCOrder.orderDate, ABCOrder.orderMethod,
      to_char(ABCOrder.orderTotal, '$9,999.99'),
      SUM(OrderProduct.orderProductQty)
FROM ABCOrder, OrderProduct
WHERE ABCOrder.orderId = OrderProduct.orderId
AND ABCOrder.orderPackDate IS NULL
GROUP BY ABCOrder.orderId, ABCOrder.orderDate, ABCOrder.orderMethod,
      ABCOrder.orderTotal
ORDER BY ABCOrder.orderDate ASC;

/* Grant view to production role */
GRANT SELECT ON PDBADMIN.view_unpacked_orders TO production;
```


Oracle SQL Developer : jrhodes_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
...sql
PDBADMIN_orclpdb_1
jrhodes_orclpdb
Connections
Worksheet
Query Builder
Query Result
SQL

```
SELECT * FROM PDBADMIN.view_unpacked_orders;
```

All Rows Fetched: 1 in 0.322 seconds

	Order ID	Order Date	Order Method	Order Total	No. of Products Ordered
1	11	14-NOV-17	site	$2,225.00	3

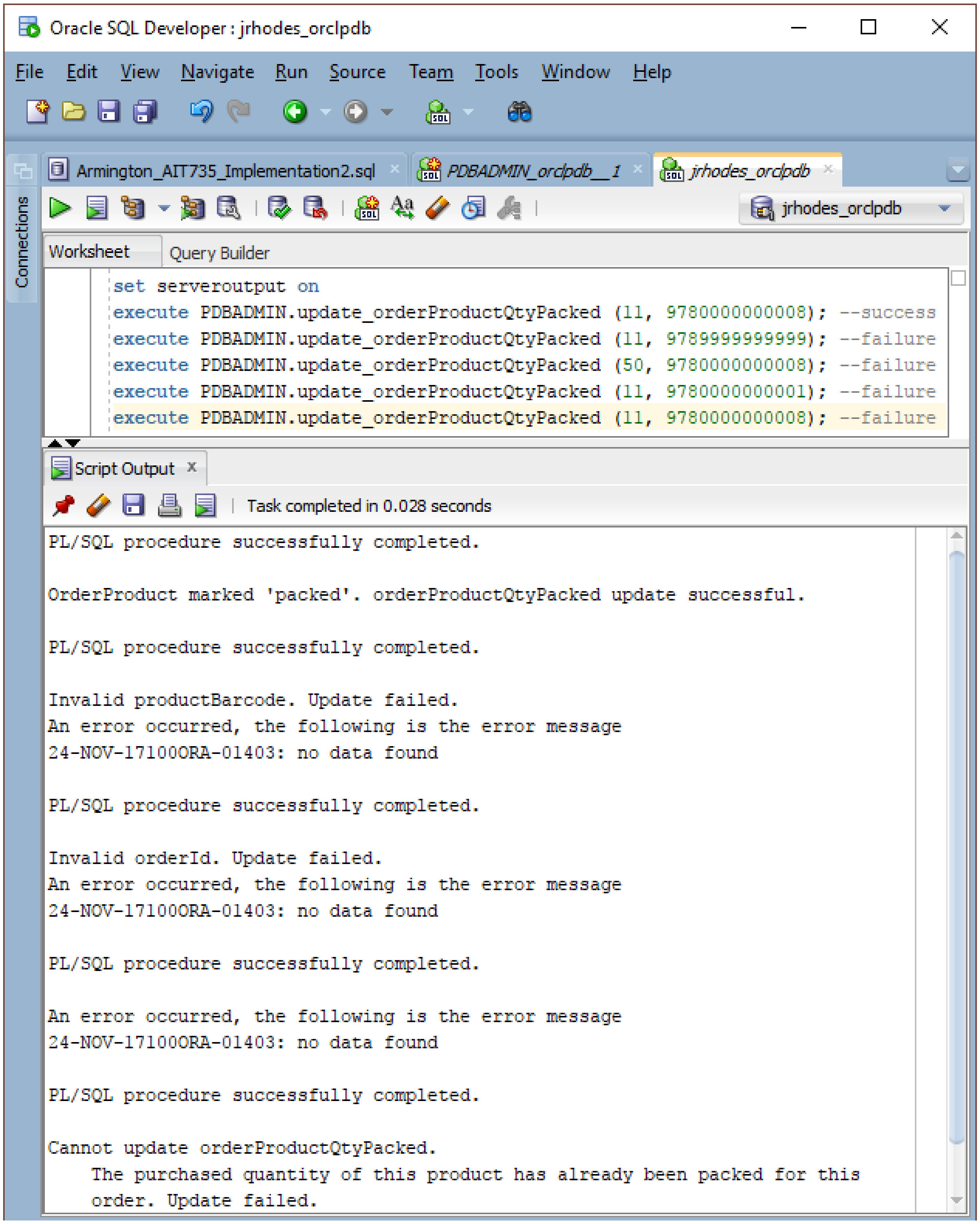
The screenshot above shows that `jrhodes`, a user granted the production role, is able to access the view `view_unpacked_orders`. It also shows that the view was created correctly. The order with order ID 11 is the only order that has not been completely packed (it has a null order pack date). The view shows the order date, order method, order total, and total number of products ordered. Therefore, an employee in the Production department can use this view to identify order IDs that need to be packed.

FR-12

Requirement: Given a valid order ID and product barcode, system must be able to update the quantity of the order product packed.

```
Create or Replace Procedure update_orderProductQtyPacked (v_orderId
int, v_productBarcode char)
```

```
IS
      v_order_id int; --used to validate orderId input
      v_product_barcode int; --used to validate productBarcode input
      v_productId int;
      v_orderProductQty int;
      v_orderProductQtyPacked int;
      v_errorCode number;
      v_errorMsg varchar2(200);
      v_currentUser varchar2(100);
      e_invalid_productBarcode EXCEPTION;
      v_status int := 0;
BEGIN
      BEGIN --Block 1: Validate orderId and productBarcode input
      SELECT orderId
      INTO v_order_id
      FROM ABCOrder
      WHERE orderId = v_orderId;
      SELECT count(*)
      INTO v_product_barcode
      FROM Product
      WHERE (productBarcode = v_productBarcode);
      IF v_product_barcode != 1 THEN
    RABCE e_invalid_productBarcode;
      END IF;
  EXCEPTION
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Invalid orderId. Update failed.');
    v_status := 1;
  WHEN e_invalid_productBarcode THEN
    DBMS_OUTPUT.PUT_LINE('Invalid productBarcode. Update failed.');
    v_status := 1;
      WHEN OTHERS THEN
            v_errorCode := SQLCODE;
            v_errorMsg := substr(SQLERRM,1,200);
            v_currentUser := USER;
      DBMS_OUTPUT.PUT_LINE('An error occurred, the following is the
error message');
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE) || v_currentUser ||
TO_CHAR(v_errorCode) || v_errorMsg || ' Update failed.');
    v_status := 1;
      END;
      BEGIN --Block 2: update OrderProduct
      SELECT productId
      INTO v_productId
      FROM Product
      WHERE (productBarcode = v_productBarcode);
      SELECT orderProductQty
      INTO v_orderProductQty
      FROM OrderProduct
      WHERE orderId = v_orderId
      AND productId = v_productId;
      SELECT orderProductQtyPacked
      INTO v_orderProductQtyPacked
      FROM OrderProduct
      WHERE orderId = v_orderId
      AND productId = v_productId;
      IF v_status != 0 THEN
      DBMS_OUTPUT.PUT_LINE('Update failed.');
      ELSIF (v_status = 0) AND (v_orderProductQty <=
v_orderProductQtyPacked) THEN
    DBMS_OUTPUT.PUT_LINE('Cannot update orderProductQtyPacked.
    The purchased quantity of this product has already been packed for
this order. Update failed.');
      ELSE
           BEGIN
           UPDATE OrderProduct
           SET orderProductQtyPacked = orderProductQtyPacked + 1
           WHERE productId = v_productId
           AND orderId = v_orderId;
           DBMS_OUTPUT.PUT_LINE('OrderProduct marked ''packed''.
orderProductQtyPacked update successful.');
           COXXX; --Transaction control
           END;
      END IF;
  EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
           v_errorCode := SQLCODE;
           v_errorMsg := substr(SQLERRM,1,200);
      DBMS_OUTPUT.PUT_LINE('An error occurred, the following is the
error message');
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE) || TO_CHAR(v_errorCode) ||
v_errorMsg);
      END;
END;

/* Grant execute on procedure to production role */
GRANT EXECUTE ON PDBADMIN.update_orderProductQtyPacked TO production;

/* Log in as jrhodes and test procedure */
conn jrhodes/Jack123

/* test procedure */
execute PDBADMIN.update_orderProductQtyPacked (11, 9780000000008);
--success
execute PDBADMIN.update_orderProductQtyPacked (11, 9789999999999);
--failure
execute PDBADMIN.update_orderProductQtyPacked (50, 9780000000008);
--failure
```

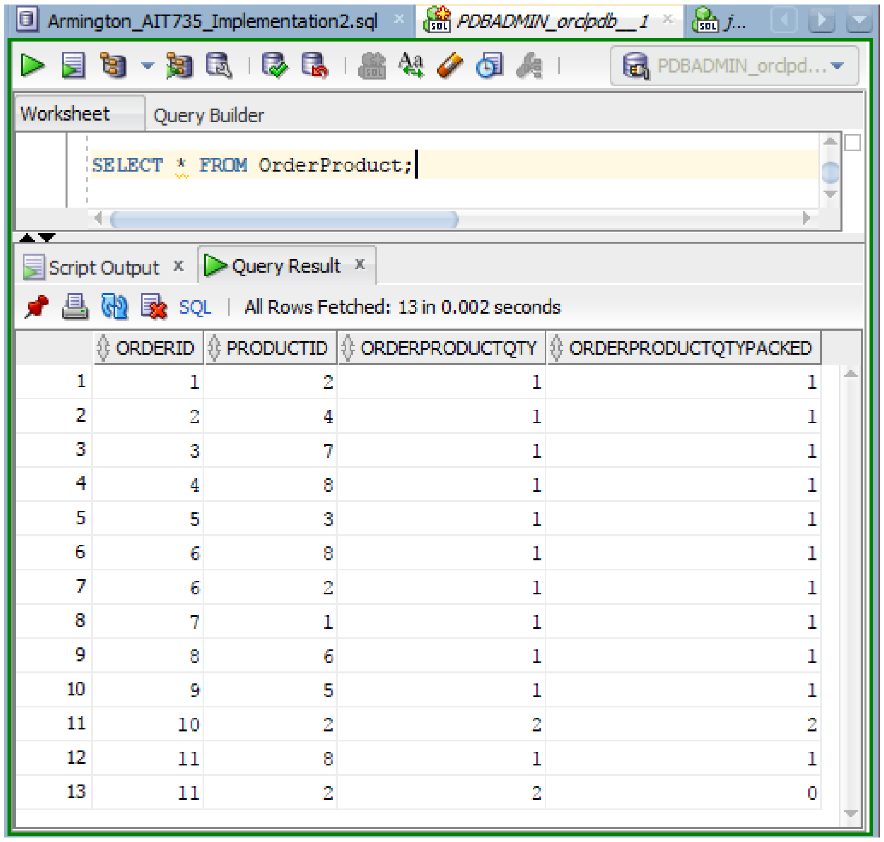

Oracle SQL Developer : jrhodes_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
jrhodes_orclpdb
Connections
Worksheet
Query Builder
Script Output

```
set serveroutput on
execute PDBADMIN.update_orderProductQtyPacked (11, 9780000000008); --success
execute PDBADMIN.update_orderProductQtyPacked (11, 9789999999999); --failure
execute PDBADMIN.update_orderProductQtyPacked (50, 9780000000008); --failure
execute PDBADMIN.update_orderProductQtyPacked (11, 9780000000001); --failure
execute PDBADMIN.update_orderProductQtyPacked (11, 9780000000008); --failure
```

Task completed in 0.028 seconds

```
PL/SQL procedure successfully completed.

OrderProduct marked 'packed'. orderProductQtyPacked update successful.

PL/SQL procedure successfully completed.

Invalid productBarcode. Update failed.
An error occurred, the following is the error message
24-NOV-1710ORA-01403: no data found

PL/SQL procedure successfully completed.

Invalid orderId. Update failed.
An error occurred, the following is the error message
24-NOV-1710ORA-01403: no data found

PL/SQL procedure successfully completed.

An error occurred, the following is the error message
24-NOV-1710ORA-01403: no data found

PL/SQL procedure successfully completed.

Cannot update orderProductQtyPacked.
    The purchased quantity of this product has already been packed for this
    order. Update failed.
```

The screenshot above shows the result of user `jrhodes`, an employee assigned the production role, executing the `update_orderProductQtyPacked` procedure. The first test case is successful and yields the message “OrderProduct marked ‘packed’. orderProductQtyPacked update successful.” This can be seen in the screenshot of the OrderProduct table below.


Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
Worksheet
Query Builder

```
SELECT * FROM OrderProduct;
```

Script Output
Query Result
SQL | All Rows Fetched: 13 in 0.002 seconds

	ORDERID	PRODUCTID	ORDERPRODUCTQTY	ORDERPRODUCTQTYPACKED
1	1	2	1	1
2	2	4	1	1
3	3	7	1	1
4	4	8	1	1
5	5	3	1	1
6	6	8	1	1
7	6	2	1	1
8	7	1	1	1
9	8	6	1	1
10	9	5	1	1
11	10	2	2	2
12	11	8	1	1
13	11	2	2	0

In the screenshot above, the order product quantity packed column (orderProductQtyPacked) has been updated from 0 to 1 for order ID 11, product ID 8. This illustrates that the `update_orderProductQtyPacked` procedure successfully updates the OrderProduct table.

The second, third, fourth, and fifth test cases executed by user `jrhodes` fail. The second test case attempts to insert an invalid product barcode (9789999999999) and the third test case attempts to insert an invalid order ID (50).

The fourth test case attempts to pass an invalid order ID and product barcode combination (11 and 9780000000001, respectively). Although both the order ID and product barcode are existing values, they are not valid together. In other words, 9780000000001 does not match the product barcodes of any of the ordered products in order ID 11. (Order ID 11 only consists of products with product barcodes of 9780000000008 and 9780000000002.)

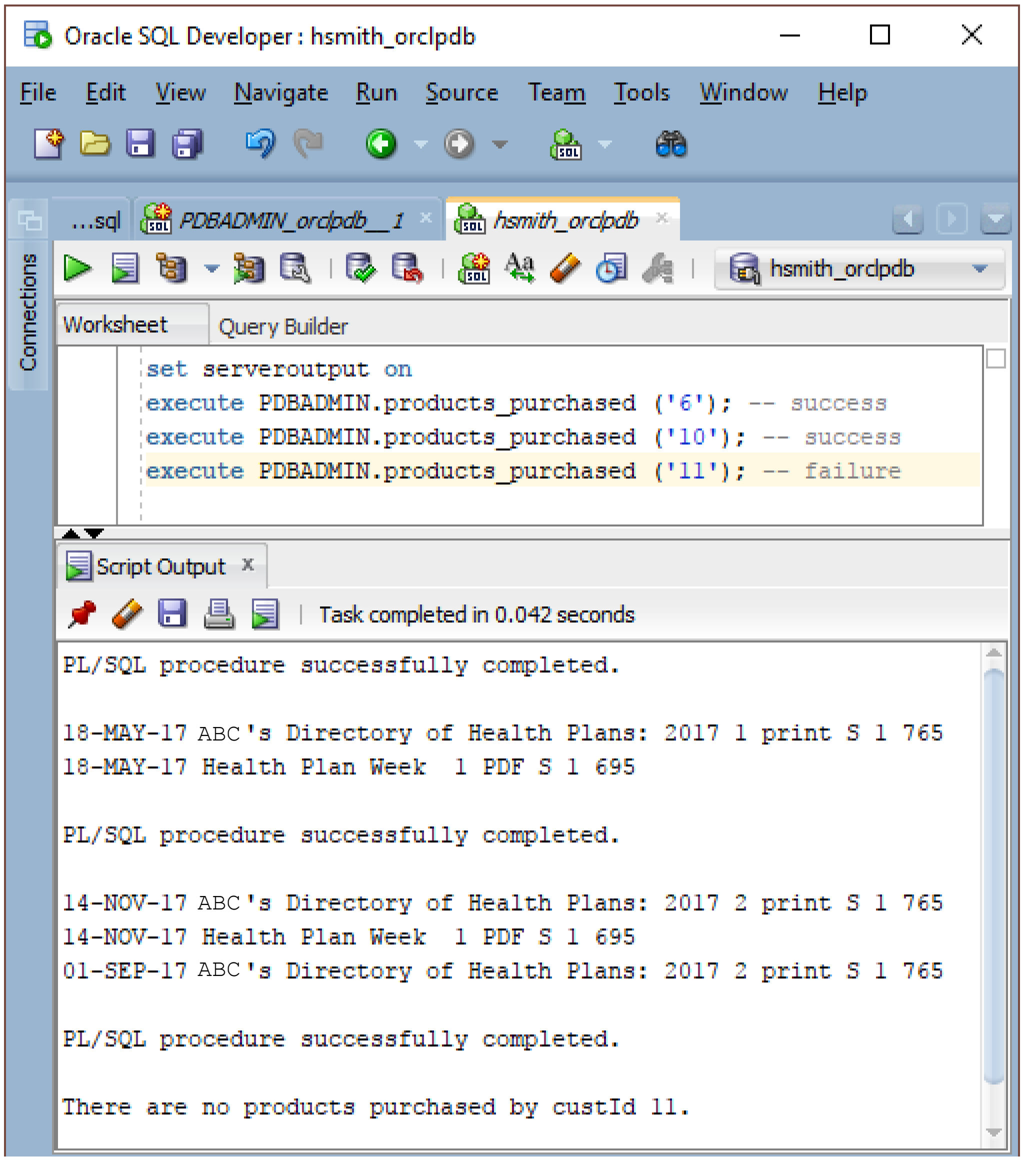
Finally, the fifth test case attempts to update the order product quantity packed for order ID 11, product ID 8. For order ID 11, the ordered quantity of product ID 8 is 1. This one product was already packed (when `jrhodes` successfully passed the first test case). Thus, the execution of the procedure fails and a message is returned to the user.

FR-13

Requirement: Given a valid customer ID, system must be able to list all products purchased and the dates that each was purchased.

```
Create or Replace Procedure products_purchased (v_custId int)
IS
        v_count int; --used for custId validation
```

```
      CURSOR c_products_purchased IS
      SELECT ABCOrder.orderDate, Publication.pubName,
OrderProduct.orderProductQty, Product.productFormat,
Product.productLicense, Product.productUserLim, Product.productPrice
      FROM ABCOrder, Product, Publication, OrderProduct
      WHERE ABCOrder.orderId = OrderProduct.orderId
      AND OrderProduct.productId = Product.productId
      AND Product.pubId = Publication.pubId
      AND custId = v_custId
      ORDER BY orderDate DESC, pubName ASC, productFormat ASC,
productLicense ASC;
BEGIN
      /* validate customerId input */
      SELECT count (*)
      INTO v_count
      FROM ABCOrder
      WHERE custId = v_custId;
      IF (v_count = 0) THEN
            DBMS_OUTPUT.PUT_LINE('There are no products purchased by
custId ' || TO_CHAR(v_custId) || '.');
      ELSE
            BEGIN
                  FOR v_product_data IN c_products_purchased LOOP
                  DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_product_data.orderDate)
|| '  ' || TO_CHAR(v_product_data.pubName) || '    ' ||
TO_CHAR(v_product_data.orderProductQty) || '      ' ||
TO_CHAR(v_product_data.productFormat) || '  ' ||
TO_CHAR(v_product_data.productLicense) || ' ' ||
TO_CHAR(v_product_data.productUserLim) || ' ' ||
TO_CHAR(v_product_data.productPrice));
                  END LOOP;
            END;
      END IF;
END;

/* grant execute permission to custrelations and sales roles */
GRANT EXECUTE ON PDBADMIN.products_purchased TO custrelations;
GRANT EXECUTE ON PDBADMIN.products_purchased TO sales;

/* test procedure */
execute PDBADMIN.products_purchased ('6'); --success
execute PDBADMIN.products_purchased ('10'); --success
execute PDBADMIN.products_purchased ('11'); --failure
```


Oracle SQL Developer : hsmith_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
...sql
PDBADMIN_orclpdb_1
hsmith_orclpdb
Connections
Worksheet
Query Builder
Script Output
Task completed in 0.042 seconds

```
set serveroutput on
execute PDBADMIN.products_purchased ('6'); -- success
execute PDBADMIN.products_purchased ('10'); -- success
execute PDBADMIN.products_purchased ('11'); -- failure
```

```
PL/SQL procedure successfully completed.

18-MAY-17 ABC's Directory of Health Plans: 2017 1 print S 1 765
18-MAY-17 Health Plan Week  1 PDF S 1 695

PL/SQL procedure successfully completed.

14-NOV-17 ABC's Directory of Health Plans: 2017 2 print S 1 765
14-NOV-17 Health Plan Week  1 PDF S 1 695
01-SEP-17 ABC's Directory of Health Plans: 2017 2 print S 1 765

PL/SQL procedure successfully completed.

There are no products purchased by custId 11.
```

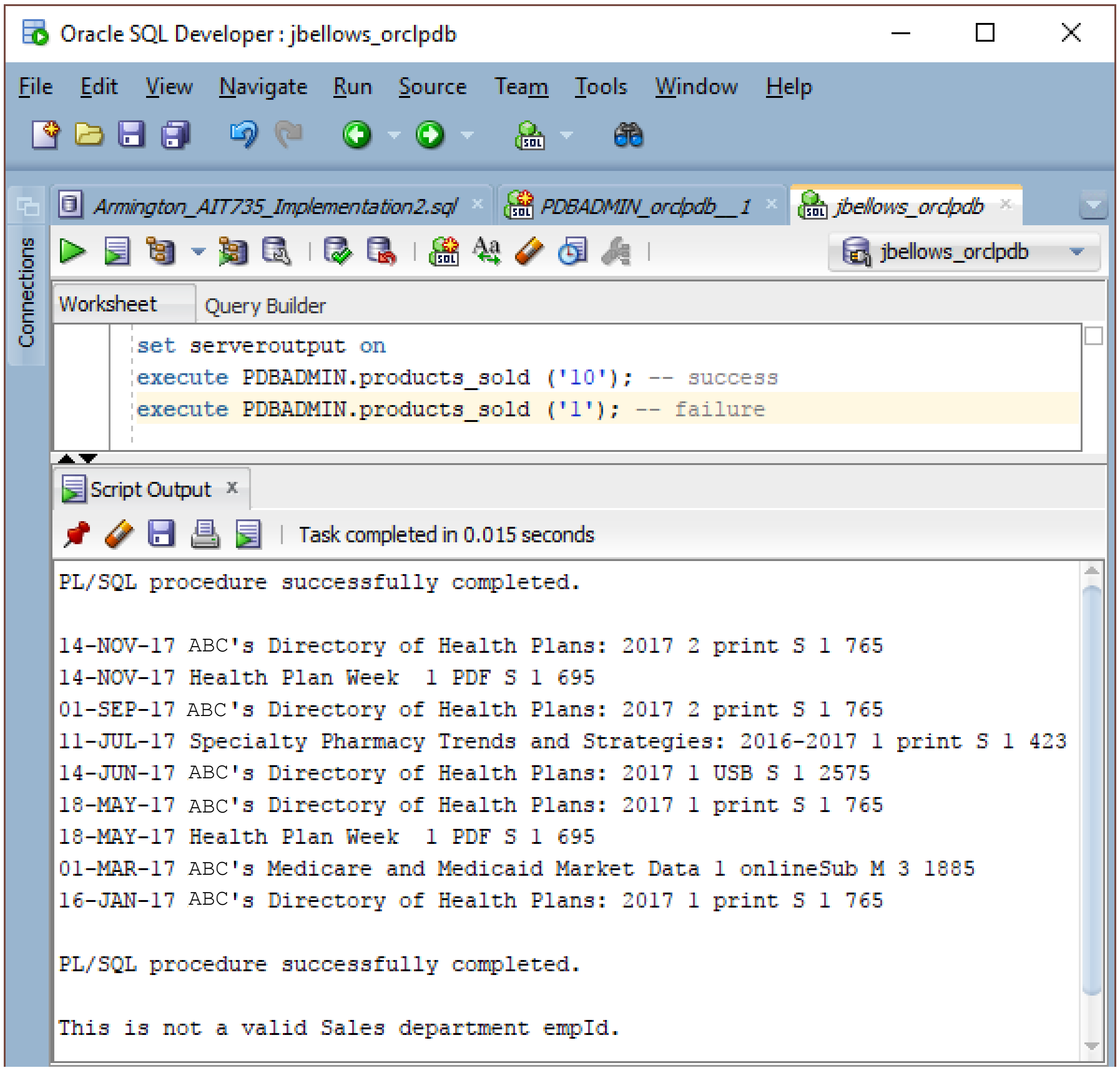
The screenshot above shows the results of `hsmith`, an employee assigned the sales role, executing the test cases for the `products_purchased` procedure. The first two are successful and the products purchased are displayed for the specified customers. The last test case attempts to pass an invalid customer ID (11) through the procedure, and it fails. Execute privileges on this procedure were granted to employees assigned the custrelations role and the sales role.

FR-14

Requirement: Given a valid sales representative's employee ID, system must be able to list all products sold and the date that each product was sold.

```
Create or Replace Procedure products_sold (v_empId int)
IS
        v_count int; --used for empId input validation
        v_deptId int; --used for department validation
        CURSOR c_products_sold IS
```

```
     SELECT ABCOrder.orderDate, Publication.pubName,
OrderProduct.orderProductQty, Product.productFormat,
Product.productLicense, Product.productUserLim, Product.productPrice
     FROM ABCOrder, Product, Publication, OrderProduct
     WHERE ABCOrder.orderId = OrderProduct.orderId
     AND OrderProduct.productId = Product.productId
     AND Product.pubId = Publication.pubId
     AND empId = v_empId
     ORDER BY orderDate DESC, pubName ASC, productFormat ASC,
productLicense ASC;
BEGIN
     SELECT deptId
     INTO v_deptId
     FROM Employee
     WHERE empId = v_empId;
     /* validate empId input: must be existing empId and belong to
employee in sales dept */
     SELECT count (*)
     INTO v_count
     FROM ABCOrder
     WHERE empId = v_empId;
     IF (v_deptId != 8) THEN
          DBMS_OUTPUT.PUT_LINE('This is not a valid Sales department
empId.');
     ELSIF (v_count = 0) THEN
          DBMS_OUTPUT.PUT_LINE('There are no products sold by empId '
|| TO_CHAR(v_empId) || '.');
     ELSE
          BEGIN
               FOR v_product_data IN c_products_sold LOOP
               DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_product_data.orderDate)
|| '  ' || TO_CHAR(v_product_data.pubName) || '   ' ||
TO_CHAR(v_product_data.orderProductQty) || '     ' ||
TO_CHAR(v_product_data.productFormat) || '  ' ||
TO_CHAR(v_product_data.productLicense) || ' ' ||
TO_CHAR(v_product_data.productUserLim) || ' ' ||
TO_CHAR(v_product_data.productPrice));
               END LOOP;
          END;
     END IF;
END;

/* grant execute privileges to manager role */
GRANT EXECUTE ON PDBADMIN.products_sold TO manager;

/* test procedure */
execute PDBADMIN.products_sold ('10'); --success
execute PDBADMIN.products_sold ('1'); --failure
```


Oracle SQL Developer : jbellows_orclpdb
File Edit View Navigate Run Source Team Tools Window Help
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
jbellows_orclpdb
Connections
Worksheet
Query Builder
Script Output
Task completed in 0.015 seconds

```
set serveroutput on
execute PDBADMIN.products_sold ('10'); -- success
execute PDBADMIN.products_sold ('1'); -- failure
```

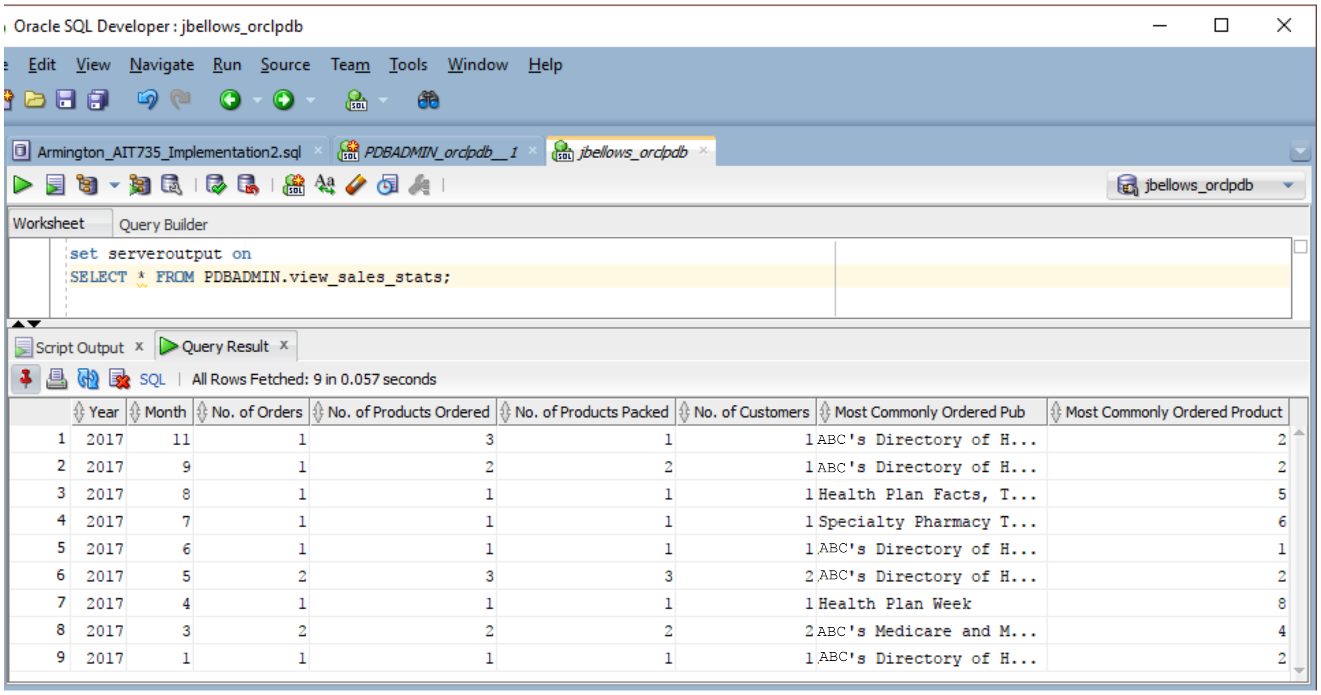
```
PL/SQL procedure successfully completed.

14-NOV-17 ABC's Directory of Health Plans: 2017 2 print S 1 765
14-NOV-17 Health Plan Week  1 PDF S 1 695
01-SEP-17 ABC's Directory of Health Plans: 2017 2 print S 1 765
11-JUL-17 Specialty Pharmacy Trends and Strategies: 2016-2017 1 print S 1 423
14-JUN-17 ABC's Directory of Health Plans: 2017 1 USB S 1 2575
18-MAY-17 ABC's Directory of Health Plans: 2017 1 print S 1 765
18-MAY-17 Health Plan Week  1 PDF S 1 695
01-MAR-17 ABC's Medicare and Medicaid Market Data 1 onlineSub M 3 1885
16-JAN-17 ABC's Directory of Health Plans: 2017 1 print S 1 765

PL/SQL procedure successfully completed.

This is not a valid Sales department empId.
```

The screenshot above shows the results of user `jbellows`, an employee assigned the manager role, executing the test cases for the `products_sold` procedure. The first case passes a valid employee ID and the system displays all of the products sold by that employee. The second test case attempts to pass an employee ID (1) that does not belong to an employee in the Sales department, and it fails. The system displays the failed message “This is not a valid Sales department empId.”

FR-15

<u>Requirement</u>: System must support a view which lists sales statistics per month, accessible by managers.

```
CREATE OR REPLACE VIEW view_sales_stats
AS SELECT EXTRACT(year FROM a.orderDate) "Year",
EXTRACT(month FROM a.orderDate) "Month",
COUNT(DISTINCT o.orderId) "No. of Orders",
SUM(o.orderProductQty) "No. of Products Ordered",
SUM(o.orderProductQtyPacked) "No. of Products Packed",
COUNT(DISTINCT a.custId) "No. of Customers",
STATS_MODE(pub.pubName) "Most Commonly Ordered Pub",
STATS_MODE(o.productId) "Most Commonly Ordered Product"
FROM ABCOrder a, OrderProduct o, Product prod, Publication pub
WHERE a.orderId = o.orderId
AND o.productId = prod.productId
AND prod.pubId = pub.pubId
GROUP BY EXTRACT(year FROM a.orderDate), EXTRACT(month FROM
      a.orderDate)
ORDER BY "Year" DESC, "Month" DESC;

/* Grant view to employee with Manager role */
GRANT SELECT ON PDBADMIN.view_sales_stats TO jbellows;
```


Oracle SQL Developer : jbellows_orclpdb
Edit View Navigate Run Source Team Tools Window Help
Armington_AIT735_Implementation2.sql
PDBADMIN_orclpdb__1
jbellows_orclpdb
Worksheet
Query Builder
Script Output
Query Result
SQL

```
set serveroutput on
SELECT * FROM PDBADMIN.view_sales_stats;
```

All Rows Fetched: 9 in 0.057 seconds

	Year	Month	No. of Orders	No. of Products Ordered	No. of Products Packed	No. of Customers	Most Commonly Ordered Pub	Most Commonly Ordered Product
1	2017	11	1	3	1	1	ABC's Directory of H...	2
2	2017	9	1	2	2	1	ABC's Directory of H...	2
3	2017	8	1	1	1	1	Health Plan Facts, T...	5
4	2017	7	1	1	1	1	Specialty Pharmacy T...	6
5	2017	6	1	1	1	1	ABC's Directory of H...	1
6	2017	5	2	3	3	2	ABC's Directory of H...	2
7	2017	4	1	1	1	1	Health Plan Week	8
8	2017	3	2	2	2	2	ABC's Medicare and M...	4
9	2017	1	1	1	1	1	ABC's Directory of H...	2

The screenshot above illustrates that user `jbellows`, an employee assigned the manager role, is able to successfully select from the view `view_sales_stats`. This view can be used by managers to deduct sales statistics per month. It pulls data from the ABCOrder, OrderProduct, Product, and Publication tables and combines them into an easy to read chart.

Configured Backups

Per the Database Backup Strategy from Deliverable 3, the Oracle database will run in `ARCHIVELOG` mode to archive redo logs while the database is open. The default database log mode is `NOARCHIVELOG` mode. Use the following command to determine whether the database is in `ARCHIVELOG` mode (note: you must be logged in as `SYS` user):

```
archive log list
```

If the database log mode says "No Archive Mode," the database is in `NOARCHIVELOG` mode. If the database log mode is in "Archive Mode," the database is in `ARCHIVELOG` mode. Use the same command to determine the archive destination. Per Deliverable 3, all backups and archives will be stored on the SAN.

Set Archive Destination

The following code should be run in Command Prompt

Login to SQL Plus as `SYS` user:

```
sqlplus sys/oracleAdm1n as sysdba
SQL> ALTER SYSTEM SET log_archive_dest_1 = 'LOCATION=<new
destination>';
```

Note: <new destination> should be replaced with a valid path name to the SAN (e.g. `/u01/oradata/MYDB/archive/`).

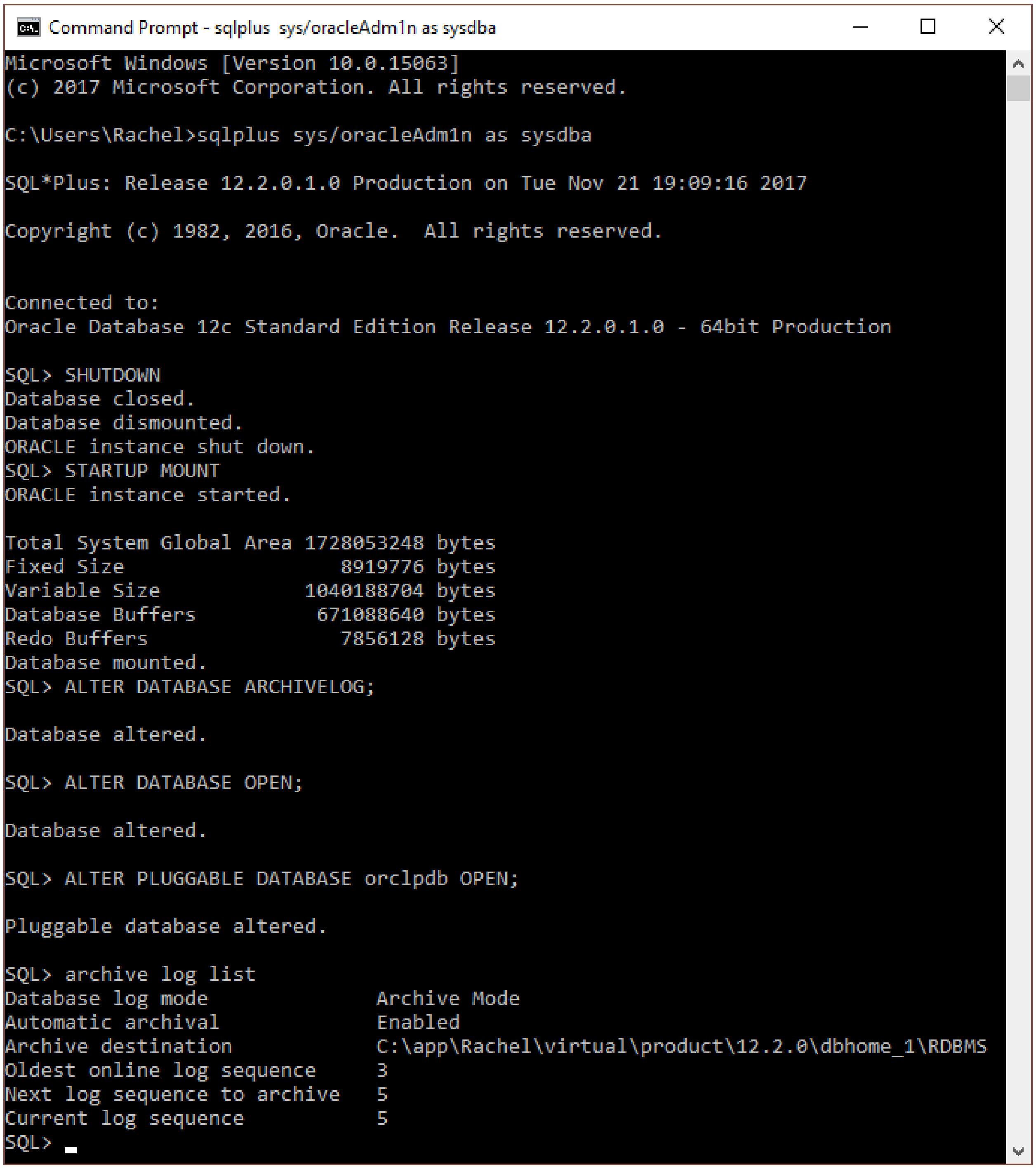
The `archive log list` command can be used again to verify that the destination has been set.

Enable Archive Mode

The following code should be run in Command Prompt

Login to SQL Plus as `SYS` user:

```
sqlplus sys/oracleAdm1n as sysdba
SQL> SHUTDOWN
SQL> STARTUP MOUNT
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
SQL> ALTER PLUGGABLE DATABASE orclpdb OPEN;
SQL> archive log list
```



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Rachel>sqlplus sys/oracleAdm1n as sysdba

SQL*Plus: Release 12.2.0.1.0 Production on Tue Nov 21 19:09:16 2017

Copyright (c) 1982, 2016, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Standard Edition Release 12.2.0.1.0 - 64bit Production

SQL> SHUTDOWN
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP MOUNT
ORACLE instance started.

Total System Global Area 1728053248 bytes
Fixed Size                  8919776 bytes
Variable Size            1040188704 bytes
Database Buffers          671088640 bytes
Redo Buffers                7856128 bytes
Database mounted.
SQL> ALTER DATABASE ARCHIVELOG;

Database altered.

SQL> ALTER DATABASE OPEN;

Database altered.

SQL> ALTER PLUGGABLE DATABASE orclpdb OPEN;

Pluggable database altered.

SQL> archive log list
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            C:\app\Rachel\virtual\product\12.2.0\dbhome_1\RDBMS
Oldest online log sequence     3
Next log sequence to archive   5
Current log sequence           5
SQL>
```

The screenshot above shows the steps taken to enable ARCHIVELOG mode in Oracle. Notice that the database log mode says “Archive Mode” after all steps are taken.

Perform Database Backup

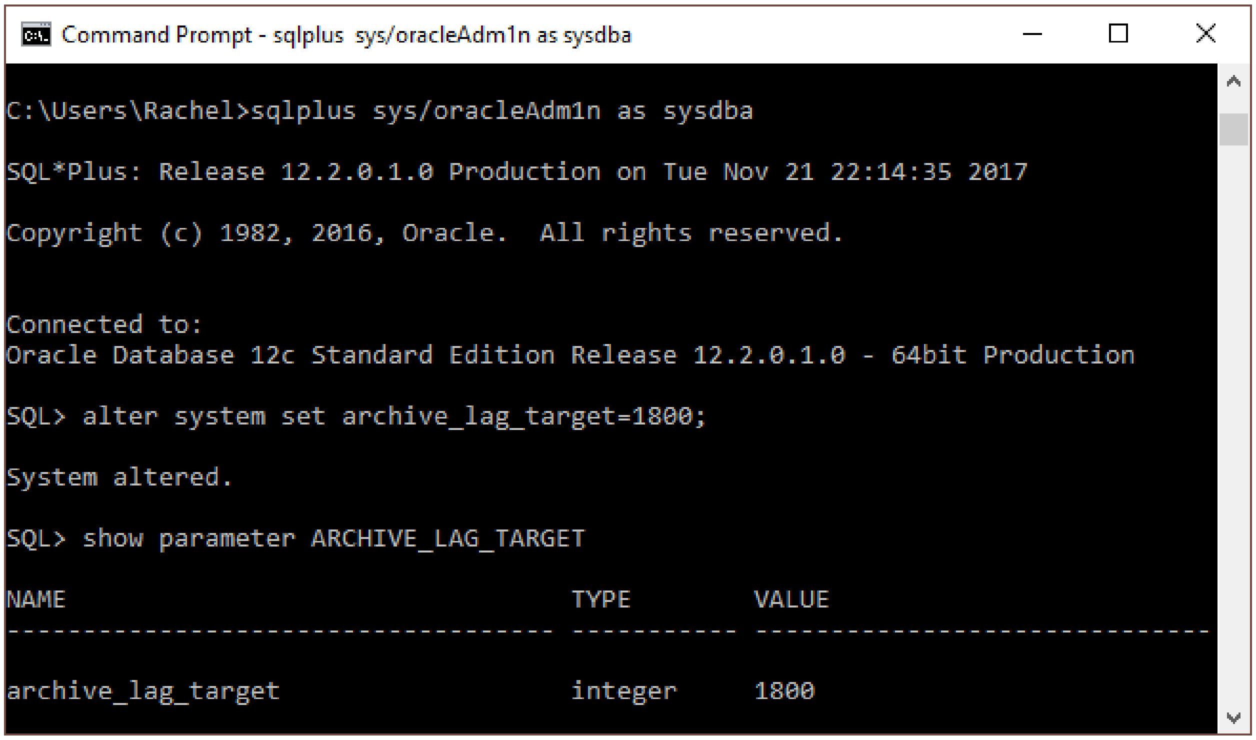
ABC Health’s Product Management System will use Oracle’s Recovery Manager (RMAN) function to assist with backup tasks. Use the following commands to connect RMAN to the target database, configure the default backup format and location, and perform backups. Note: backups must be conducted by a user with `SYSBACKUP` privileges. User `c##bgartrell` will be conducting full CDB backups (which includes PDBs) at ABC, but for purposes of this prototype, backups are done by user `SYS`.

The following code should be run in Command Prompt

```
RMAN TARGET '"SYS as sysbackup"' --connect to target database
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '<path to disk
location>'
/* specifies the channel, or pathway to disk location, for backup.
Replace <disk location> with valid path to SAN */
RMAN> BACKUP INCREMENTAL LEVEL 0 DATABASE PLUS ARCHIVELOG TAG
'BACKUP_0';
/* Level 0 incremental backups plus archive log to be performed every
Saturday night (weekly) at 6:00 p.m. */
RMAN> BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE TAG 'BACKUP_1';
/* Level 1 cumulative incremental backups to be performed every night
(daily) at 6:00 p.m. */
```

Furthermore, ABC can only afford to lose 30 minutes of data (RPO). Thus, all redo logs can be forced to archive at a set interval (in seconds) using the `ARCHIVE_LAG_TARGET` parameter. For example:

```
SQL> ALTER SYSTEM SET ARCHIVE_LAG_TARGET = 1800;
```



```
C:\Users\Rachel>sqlplus sys/oracleAdm1n as sysdba

SQL*Plus: Release 12.2.0.1.0 Production on Tue Nov 21 22:14:35 2017

Copyright (c) 1982, 2016, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Standard Edition Release 12.2.0.1.0 - 64bit Production

SQL> alter system set archive_lag_target=1800;

System altered.

SQL> show parameter ARCHIVE_LAG_TARGET

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
archive_lag_target                   integer     1800
```

In this screenshot, the log switch time is set to 1800 seconds (30 minutes). ABC Health's Product Management System has a primary node and a secondary failover node. Setting this parameter forces the primary node to archive the redo log every 30 minutes to limit the lag in the data between the two nodes.

Finally, full backups will also be written to tape. The following scripts should be run in RMAN:

```
RUN
```

```
{
  ALLOCATE CHANNEL c1
    DEVICE TYPE sbt --for tape device type
    PARMS 'ENV=(<instructions to media manager>)';
  /* enter instructions to media manager into the PARMS parameter. For
example, PARMS 'ENV=(OB_MEDIA_FAMILY=archive_backup)' instructs the
media manager to backup data to 'Archive Backup' media family. */
    BACKUP DATABASE
    TAG weekly_backup
    KEEP UNTIL TIME 'SYSDATE+84';
}
/* script should be run to archive full weekly backups for 84 days (12
weeks). After 12 weeks, the tapes are rotated and overwritten with new
data */

RUN
{
  ALLOCATE CHANNEL c1
    DEVICE TYPE sbt
    PARMS 'ENV=(<instructions to media manager>)';
    BACKUP DATABASE
    TAG quarterly_backup
    KEEP FOREVER
    RESTORE POINT FY17Q4;
}
/* script should be run to permanently archive quarterly backups */
```

Configured Audits

Per the Auditing Requirements described in Deliverable 3, the Oracle database will use unified auditing to improve system security. Use the following command to determine whether the database is using unified auditing:

```
select * from v$option where PARAMETER = 'Unified Auditing';
```

If the query returns `FALSE`, use the following instructions to enable unified auditing. If the query returns `TRUE`, proceed to "Create Object Action Unified Audit Policy."

Enable Unified Auditing

The following code should be run in Command Prompt

Login to SQL Plus as `SYS` user:

```
sqlplus sys/oracleAdm1n as sysdba
SQL> SHUTDOWN IMMEDIATE
SQL> EXIT
net stop OracleServiceabcorcl --stop the Oracle service
lsnrctl stop --stop the Oracle listener
```

Navigate to the Oracle Home directory. The Oracle Home directory for this prototype is:

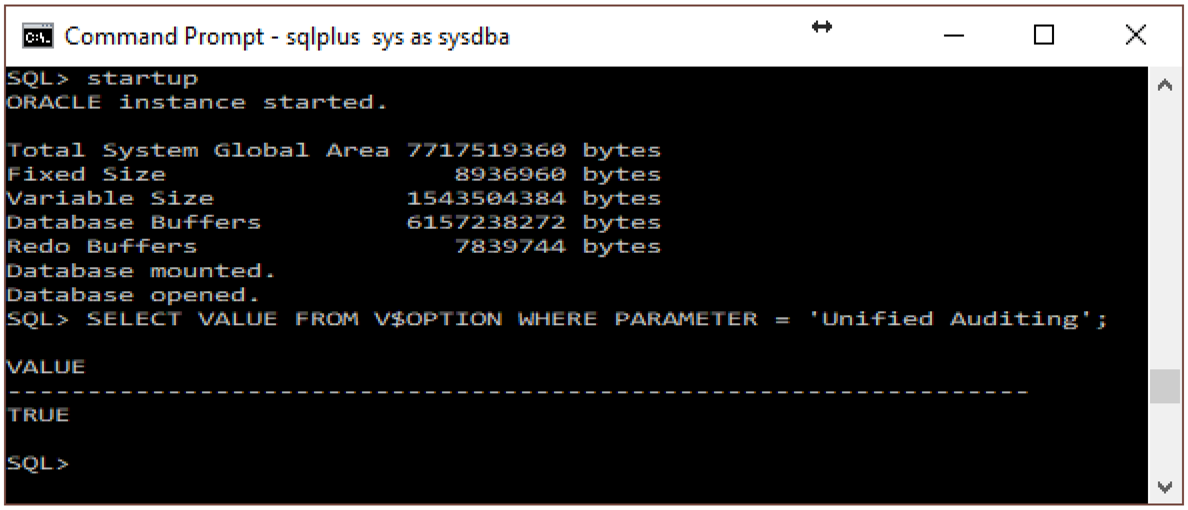
```
C:\app\Rachel\virtual\product\12.2.0\dbhome_1
```

Enable the unified auditing executable by renaming the `orauniaud12.dll.option` to `orauniaud12.dll`. In the case of this prototype, rename

`C:\app\Rachel\virtual\product\12.2.0\dbhome_1\bin\orauniaud12.dll.dbl` to `C:\app\Rachel\virtual\product\12.2.0\dbhome_1\bin\orauniaud12.dll`.

Restart the listener, Oracle service, and Oracle database:

```
lsnrctl start
net start OracleServiceabcorcl
sqlplus sys/oracleAdm1n as sysdba
SQL> STARTUP
SQL> ALTER pluggable database orclpdb open;
SQL> select name, open_mode from v$pdbs;
--ensure PDB is in READ/WRITE mode, not just MOUNTED
```



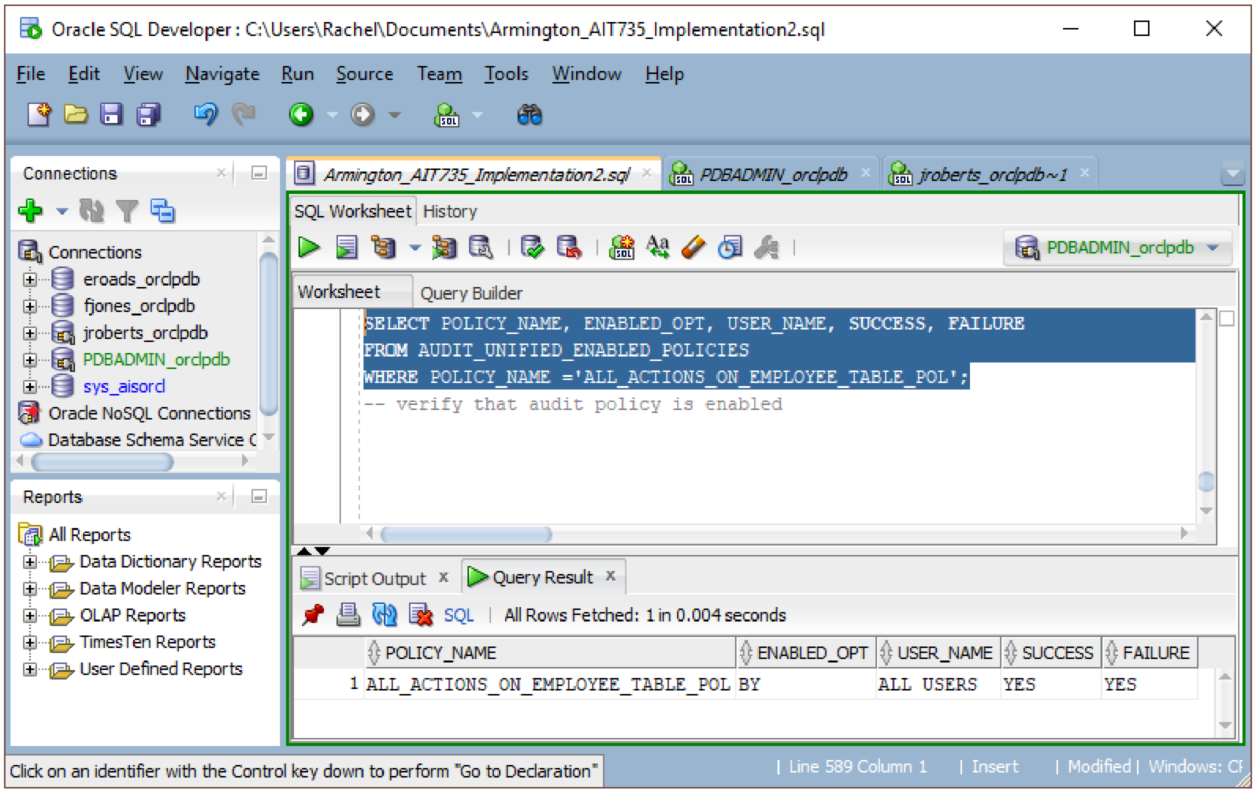
```
SQL> startup
ORACLE instance started.

Total System Global Area 7717519360 bytes
Fixed Size                  8936960 bytes
Variable Size            1543504384 bytes
Database Buffers         6157238272 bytes
Redo Buffers                7839744 bytes
Database mounted.
Database opened.
SQL> SELECT VALUE FROM V$OPTION WHERE PARAMETER = 'Unified Auditing';

VALUE
----------------------------------------------------------------
TRUE

SQL>
```

This screenshot shows that Oracle database is now using unified auditing because the Value equals `TRUE`.

Create Object Action Unified Audit Policy

In ABC Health’s Product Management System, the only table that contains extremely confidential information is the Employee table (stores employees’ social security numbers). Therefore, a unified audit policy should be created to log all actions on the Employee table and to improve security.

```
CREATE AUDIT POLICY all_actions_on_employee_table_pol
      ACTIONS ALL ON PDBADMIN.EMPLOYEE;
AUDIT POLICY all_actions_on_employee_table_pol;
SELECT POLICY_NAME, ENABLED_OPT, USER_NAME, SUCCESS, FAILURE
      FROM AUDIT_UNIFIED_ENABLED_POLICIES
      WHERE POLICY_NAME ='ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL';
      --verify that audit policy is enabled
```



```
SELECT POLICY_NAME, ENABLED_OPT, USER_NAME, SUCCESS, FAILURE
FROM AUDIT_UNIFIED_ENABLED_POLICIES
WHERE POLICY_NAME ='ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL';
-- verify that audit policy is enabled
```

```
  POLICY_NAME                          ENABLED_OPT USER_NAME SUCCESS FAILURE
1 ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL    BY          ALL USERS YES     YES
```

This screenshot shows that the `all_actions_on_employee_table_pol` audit policy is enabled. The unified audit trail log will record all users' actions (successes and failures) on the Employee table.

This audit policy can be tested by logging in to the database as various users and attempting to select rows from the Employee table. For example,

```
SELECT empId, EmpSSN FROM PDBADMIN.EMPLOYEE;
```

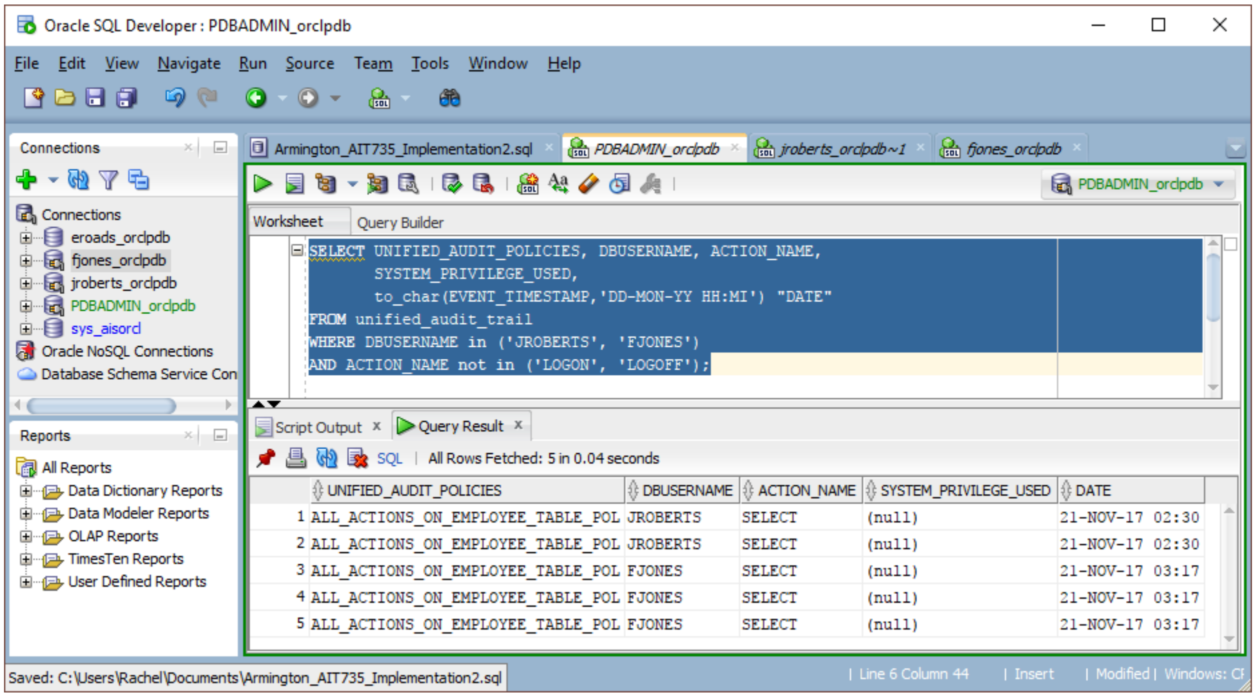
Performing this action as user `jroberts`, an employee in the Human Resources department, will yield a successful result because `jroberts` was assigned the HR role, which has `SELECT` privileges on the Employee table. On the other hand, performing this action as user `fjones`, an employee in the Customer Relations department, will produce an error and fail because this user has insufficient privileges to select from the Employee table.

To view the audit log, login as `PDBADMIN`. If the audited data is still in memory, it must be flushed to disk:

```
EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;
```

Query the audited data:

```
SELECT UNIFIED_AUDIT_POLICIES, DBUSERNAME, ACTION_NAME,
       SYSTEM_PRIVILEGE_USED,
       to_char(EVENT_TIMESTAMP,'DD-MON-YY HH:MI') "DATE"
FROM unified_audit_trail
WHERE DBUSERNAME in ('JROBERTS', 'FJONES')
AND ACTION_NAME not in ('LOGON', 'LOGOFF');
```



```
SELECT UNIFIED_AUDIT_POLICIES, DBUSERNAME, ACTION_NAME,
        SYSTEM_PRIVILEGE_USED,
        to_char(EVENT_TIMESTAMP,'DD-MON-YY HH:MI') "DATE"
FROM unified_audit_trail
WHERE DBUSERNAME in ('JROBERTS', 'FJONES')
AND ACTION_NAME not in ('LOGON', 'LOGOFF');
```

	UNIFIED_AUDIT_POLICIES	DBUSERNAME	ACTION_NAME	SYSTEM_PRIVILEGE_USED	DATE
1	ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL	JROBERTS	SELECT	(null)	21-NOV-17 02:30
2	ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL	JROBERTS	SELECT	(null)	21-NOV-17 02:30
3	ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL	FJONES	SELECT	(null)	21-NOV-17 03:17
4	ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL	FJONES	SELECT	(null)	21-NOV-17 03:17
5	ALL_ACTIONS_ON_EMPLOYEE_TABLE_POL	FJONES	SELECT	(null)	21-NOV-17 03:17

This screenshot shows that users `jroberts` and `fjones` tried to perform the `SELECT` action from the Employee table.

References

Oracle. (2017). 17 Introduction to the multitenant architecture. Retrieved from https://docs.oracle.com/database/121/CNCPT/cdbovrvw.htm#CNCPT89236