

AutoZone

(A National retail and service store)

Name: Naga Venkata Kanakalakshmi

UNT ID: 11725119

Project -Part 3 (Queries, updates, and deletes)

Group Number: 8

Group Members:

Name	UNT ID	Mail ID
Naga Venkata kanakalakshmi	11725119	nagavenkatakanakalmurikipudi@my.unt.edu
Vishnu Vardhan Reddy Sudireddy	11642773	vishnusudiredy@my.unt.edu
Srinivas Sankula	11667743	srinivassankula@my.unt.edu
Sai Sindhu Rudraraju	11644475	saisindhurudraraju@my.unt.edu

Introduction:

AutoZone, a national retailer of automotive parts and accessories Company has requirement to store the information of the inventory, employees, insurance, servicing, payment plans, supplier details and customer details. AutoZone also want to store the information about locations, jobs done by the employees and automotive retailer branch details.

Description:

- Each retailer in the AutoZone chain needs to be identified based on the retailer id. It can have the basic information like contact, business hour and each AutoZone retailer can have a manager and the website details related to the that location. Multiple employees can work in the one automotive retailer. Each retailer can have in house inventory and external inventory. Automotive retailer address/location needs to be stored. Each automotive retailer can afford multiple jobs or services.
- Inventory product entity serves as a major component and uniquely identified in this system. It manages the various automotive products. It includes the attributes like name, quantity, price, automotive_retailer_id, automobile_id, address_id. It holds the details of the automotive retailer, automobile data. It establishes the relation with suppliers, automotive retailers, part service, automobile, address and bill. It serves as a central component for efficient inventory management, procurement, and tracking within the system.
- Employee entity holds the data of the employees working in AutoZone. Each employee is uniquely identified with their ID. Along with the ID this entity will also have the attributes first name, last name, date of birth, phone number, email address, annual salary, ssn, address, hire date and the automotive retailer ID. An employee can create many bills for the customers and so the employee entity will form a one-to-many relation with the bill entity. More than one employee can have the same address as there is a chance of 2 employees living in the same location. So, employee will form a many to one relationship with the address entity. Many employees can be part of a job and one employee can be part of many jobs. So, employee entity will form many to many relationships with the job entity. As Many employees work in one location, employee entity will form a many to one relationship with the automotive retailer

entity. Employees may receive multiple paychecks over a period. So, Employee entity will form a one-to-many relationship with employee payroll entity.

- Employee payroll entity has the record of paychecks given to employees. Each pay given to an employee is identified by a unique id. This entity has the attributes hours worked (number of hours worked by employee during the pay cycle), start date (start date of the pay cycle), end date (end date of the pay cycle), pay (amount paid to the employee) and employee id (id of the employee that is used to uniquely identify an employee). The employee payroll entity will form a many to one relationship with the employee as one employee may receive many pays over a period.
- AutoZone can have suppliers to inventory who supply the products that stored or used during the services. An inventory can have multiple suppliers and vice versa. Suppliers' information like contact and address needs to be stored. Suppliers can uniquely identified by their id.
- Address entity serves as a global component within the system, defines location information. It is uniquely identified and includes the attributes like apartment number, street, city, state, country, and ZIP code, all of them are mandatory. This entity forms relationships with other entities in various ways: many employees may have a single address, inventory products are associated with specific addresses, automotive retailers and suppliers have distinct addresses, and multiple customers can be linked to one address.
- Bill entity is crucial for recording financial transactions. It is uniquely defined and includes attributes like date, payment mode, insurance, customer, job, employee, sale type (can be an online or offline), and payment plan. It forms relationships with various entities: multiple bills can be linked to one employee, have many-to-many connections with inventory products, and maintain multiple bills of each association with insurance, customers, and payment plans. Additionally, every job has a specific bill.
- Job entity is essential for managing automotive service tasks. It is uniquely defining and includes attributes like date, description, customers, automotive retailers, VIN numbers, and automobiles. All of which are mandatory. "Job" forms key relationships: It establishes a direct link with billing records, multiple jobs are associated with automobiles and automotive retailers, Customers. Various jobs can be done by different employees. Whereas Single job can be performed using multiple part services. This entity serves as a central hub for tracking and managing automotive service jobs.
- Part service like during the service which are automotive parts we are using to get the job done for a vehicle. It may have the information like job details, name/description of the service and it can have quantity of the parts which are using for that service. Part service can be identified by their id. Part service can have type like is it only service or any parts used to get that service done. In this many parts service can be related to one job and it have information related to the inventory products to know the parts related to the which inventory.
- Customer entity represents individuals in the system and is identified uniquely. It includes essential attributes like first name, last name, birthdate, driver's license, phone, and address. Multiple customers may opt for multiple insurance policies. A Customer can be associated with multiple jobs which has multiple bills. Many customers may have the single address. This entity enables efficient management of customer data, service history, and financial transactions.
- Insurance is like if customer wants to utilize his insurance plan for the payment, we need to store the information of that insurance. It can have policy type, provider and claim percentage. The particular insurance plan can be identified by plan id. A customer can have multiple insurance and vice versa. we would also like to include insurance id in the bill, it will be like one insurance plan can be included in many bills.

- Payment plan like if customer interested in the paying in instalments, he can use this option. This payment plan will have the information like plan name, number of instalments and the interest rate related to the payment plan. Each payment can be identified by plan id and multiple bills can have single payment plan.
- Would like to store the information about automobile. So that when the job is performed we can use the parts related to the specific automobile model. It can have the information like manufacture, name, variant, year of the build and color of the vehicle. Each automobile model can be identified by automobile id. An automobile can have multiple products in the inventory and would like to keep the automobile information in the jobs performed. It will be like multiple jobs can be performed to an automobile.
- Inventory-product-supplier is a relation table. Which has a primary and foreign key as inventory_product_id, supplier_id. It has a single attribute quantity. Multiple suppliers will provide multiple products. In this we maintain the product_id, supplier_id and the product quantity provided by the supplier.
- Bill_inventory_product is a relation table. Which has a primary and foreign key as bill_id, inventory_product_id. It has a single attribute quantity. Multiple products will have the multiple bills. In this we maintain the bill_id, product_id and the product quantity in each generated bill.
- Job_employee is a relation table. Which has the primary and foreign key as job_id and employee_id. There are multiple employees, working for the on the different jobs to maintains the those details this table helps to track the respective details. It's a join of employee and job and the job table.
- Customer_insurance is a relation table. Which has the primary and foreign key as customer_id and insurance_id. It has a single attribute status. Many customers can opt for multiple insurances. This table helps to track the insurance opted by the customers and their status.
- In the automotive_retailer a name attribute is newly added. The name identifier helps to track the name of specific automotive_retailer.

Note: No changes were made to the description as part of project part 3, with a focus on resolving queries.

Assumptions:

- A product can be available both in store and in warehouse. If the product has automotive_retailer_id same as address id, that means the product is in store. And if automotive_retailer_id is null, then the inventory is not in store.
- Automobile entity will have the generic information about the model, manufacturer and make of the automobile available in the market.
- A job will be created if a customer wants to get his automobile serviced in the store and the bill will be generated for the job.
- A bill can be generated without a job when a customer purchases products from the store without getting the service done in store where this data is stored in bill_inventory table.

- Employee can receive multiple payments over a period. This is maintained in employee_payroll table.
- Employee can be part of the job and he can generate bill for the job. And employee can sell the products to the customer who is not seeking service (not part of the job) from the retailer.
- Address is global table where the addresses of suppliers, customers, employees, inventory and retailer are stored.
- customer can pay the bill directly or he can opt for payment plan. Customer can select the payment plan form the payment_plan table.
- Job will be created if a customer opts services from the retailer. services are work done by the employees on the automobile to fix issues or install accessories.
- Part_services have the information about the products used in the job. One Job can have multiple products.
- Insurance table has different insurance policies that are available in the market.
- more than one person (customer or employee) can have same address. but no 2 suppliers and retailers can have the same address.
- product id in part service can be null as labor chargers of a job can also be clocked in part service where no inventory will be tagged to that service.
- The total amount of the bill of a customer who opted for a payment plan will be recorded as a transaction in bill table. And the installments are also saved as transactions in the same table.
- A phone number must consist of 10 digits and should only contain numerical characters, with no special symbols or spaces.
- Each email address must be unique, meaning no two users can share the same email ID.
- In the bill table, the sale type can be categorized as either online or offline.
- The vin_number in the job table must have a fixed length of 17 characters.
- The quantity of one item in a bill cannot exceed 10, as customers are limited to purchasing a maximum of 10 identical items.
- The status of insurance in the customer_insurance table can be either active or inactive.
- Zip codes are required to have a fixed length of 5 characters.
- Employees cannot work for more than 100 hours within a two-week period.
- The mode of payment in the bill table can be either cash or card.

Along with the above assumptions we have added few more new assumptions as part of project part 3:

- Sales cannot occur across different AutoZone locations. It is limited to few locations.
- When an automotive retailer is deleted from a location, the suppliers associated with that location are also removed.

Note: In Project Part-3, no modifications were made to the existing data or relationships between tables. To verify the correctness of the Entity-Relationship (ER) diagram, please refer to the Part 2 of the project.

Analysis					
Table	Attributes	Domain Constraints	Constraints	Constraint Name	Relations
automotive_retailer	id	RAW(16)	PRIMARY KEY	NA	automotive_retailer has one to many relationship with employee, automotive_retailer has one to many relationship with inventory_product, automotive_retailer has one to one relationship with address, automotive_retailer has one to many relationship with job
	phone	VARCHAR2(20)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (phone, '^\\(\\d{3}\\)\\d{3}-\\d{4}\$'))	invalid_automotive_retailer_phone	
	email	VARCHAR2(255)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (email, '[A-Za-z0-9. %+-]+@[A-Za-z0-9.-] +\\.[A-Za-z]{2,}\$'))	invalid_automotive_retailer_email	
	website	VARCHAR2(255)	NOT NULL, UNIQUE	NA	
	business_hours	VARCHAR2(20)	NOT NULL	NA	
	manager_id	RAW(16)	NOT NULL, FOREIGN KEY (manager_id) REFERENCES employee(id)	NA	
	address_id	RAW(16)	NOT NULL, FOREIGN KEY (address_id) REFERENCES address(id)	NA	
inventory_product	id	RAW(16)	PRIMARY KEY	NA	inventory_product has many to many relationship with supplier, inventory_product has many to one realationship with automotive_retailer, inventory_product has many to many relationship with bill, inventory_product has one to many relationship with part_service, inventory_product has many to one relationship with automobile, inventory_product has many to one relationship with address
	name	VARCHAR2(100)	NOT NULL	NA	
	quantity	INTEGER	NOT NULL	NA	
	price	INTEGER	NOT NULL	NA	
	automotive_retailer_id	RAW(16)	NOT NULL, FOREIGN KEY (automotive_retailer_id) REFERENCES automotive_retailer(id)	NA	
	automobile_id	RAW(16)	"NOT NULL, FOREIGN KEY (automobile_id) REFERENCES automobile(id)"	NA	
	address_id	RAW(16)	NOT NULL, FOREIGN KEY (address_id) REFERENCES address(id)	NA	
automobile	id	RAW(16)	PRIMARY KEY	NA	automobile has one to many relationship with inventory_product, automobile has one to many relationship with job
	manufacturer	VARCHAR2(100)	NOT NULL	NA	
	name	VARCHAR2(100)	NOT NULL	NA	
	variant	VARCHAR2(100)	NOT NULL	NA	
	year	VARCHAR2(10)	NOT NULL	NA	
	color	VARCHAR2(100)	NOT NULL	NA	
supplier	id	RAW(16)	PRIMARY KEY	NA	supplier has many to many relationship with inventory_product, supplier has one to one relationship with address
	name	VARCHAR2(100)	NOT NULL	NA	
	phone	VARCHAR2(20)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (phone, '^\\(\\d{3}\\)\\d{3}-\\d{4}\$'))	invalid_supplier_phone	

Analysis					
Table	Attributes	Domain Contraints	Constraints	Constraint Name	Relations
	address_id	RAW(16)	NOT NULL, FOREIGN KEY (address_id) REFERENCES address(id)	NA	
Employee	id	RAW(16)	PRIMARY KEY	NA	employee has one to many relationship with employee_payroll, employee has many to one relationship with automotive_retailer, employee has one to many relationship with bill, employee has many to one relationship with address, employee has many to many relationship with job
	first_name	VARCHAR2(100)	NOT NULL	NA	
	last_name	VARCHAR2(100)	NOT NULL	NA	
	dob	TIMESTAMP	NOT NULL	NA	
	phone	VARCHAR2(20)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (phone, '^\\(\\d{3}\\)\\d{3}-\\d{4}\$'))	invalid_employee_phone	
	email	VARCHAR2(255)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (email, '^@[A-Za-z0-9. _%+-]+@[A-Za-z0-9.-] +\\. [A-Za-z]{2,}\$'))	invalid_employee_email	
	annual_salary	INTEGER	NOT NULL	NA	
	ssn	CHAR(9)	NOT NULL, UNIQUE	NA	
	automotive_retailer_id	RAW(16)	NOT NULL, FOREIGN KEY (automotive_retailer_id) REFERENCES automotive_retailer(id)	NA	
	address_id	RAW(16)	NOT NULL, FOREIGN KEY (address_id) REFERENCES address(id)	NA	
	hire_date	TIMESTAMP	NOT NULL	NA	
employee_payroll	id	RAW(16)	PRIMARY KEY	NA	employee_payroll has many to one relationship with employee
	hours_worked	NUMBER	NOT NULL	NA	
	start_date	TIMESTAMP	NOT NULL	NA	
	end_date	TIMESTAMP	NOT NULL	NA	
	pay	NUMBER	NOT NULL	NA	
	employee_id	RAW(16)	NOT NULL, FOREIGN KEY (employee_id) REFERENCES employee(id)	NA	
address	id	RAW(16)	PRIMARY KEY	NA	address has one to many relationship with employee, address has one to many relationship with inventory_product, address has one to one relationship with automotive_retailer, address has one to one relationship with supplier, address has one to many relationship with customer
	apartment_no	NUMBER	NOT NULL	NA	
	street	VARCHAR2(100)	NOT NULL	NA	
	city	VARCHAR2(100)	NOT NULL	NA	
	state	VARCHAR2(50)	NOT NULL	NA	
	country	VARCHAR2(50)	NOT NULL	NA	
	zip	CHAR(5)	NOT NULL	NA	
	id	RAW(16)	PRIMARY KEY	NA	

Analysis					
Table	Attributes	Domain Contraints	Constraints	Constraint Name	Relations
bill	date	TIMESTAMP	NOT NULL	NA	bill has many to one relationship with employee, bill has many to many relationship with inventory_product, bill has many to one relationship with insurance, bill has many to one relationship with customer, bill has one to one relationship with job, bill has many to one relationship with payment_plan
	mode_of_payment	CHAR(4)	NOT NULL, CHECK (mode_of_payment IN ('CARD', 'CASH'))	invalid_mode_of_payment	
	insurance_id	RAW(16)	NOT NULL, FOREIGN KEY (insurance_id) REFERENCES insurance(id)	NA	
	customer_id	RAW(16)	NOT NULL, FOREIGN KEY (customer_id) REFERENCES customer(id)	NA	
	job_id	RAW(16)	NOT NULL, FOREIGN KEY (job_id) REFERENCES job(id)	NA	
	employee_id	RAW(16)	NOT NULL, FOREIGN KEY (employee_id) REFERENCES employee(id)	NA	
	sale_type	CHAR(7)	NOT NULL, CHECK (mode_of_payment IN ('ONLINE', 'OFFLINE'))	invalid_sale_type	
	payment_plan_id	RAW(16)	NOT NULL, FOREIGN KEY (payment_plan_id) REFERENCES payment_plan(id)	NA	
job	id	RAW(16)	PRIMARY KEY	NA	job has one to one realtionship with bill, job has many to one realtionship with automobile, job has many to one realtionship with automotive_retailer, job has many to many relationship with employee, job has one to many realtionship with part_service, job has many to one realtionship with customer
	date	TIMESTAMP	NOT NULL	NA	
	description	VARCHAR2(255)	NOT NULL	NA	
	customer_id	RAW(16)	NOT NULL, FOREIGN KEY (customer_id) REFERENCES customer(id)	NA	
	automotive_retailer_id	RAW(16)	NOT NULL, FOREIGN KEY (automotive_retailer_id) REFERENCES automotive_retailer(id)	NA	
	vin_number	VARCHAR2(100)	NOT NULL, CHECK()	NA	
	automobile_id	RAW(16)	NOT NULL, FOREIGN KEY (automobile_id) REFERENCES automobile(id)	NA	
part_service	id	RAW(16)	PRIMARY KEY	NA	part_service has many to one relationship with job
	name	VARCHAR2(100)	NOT NULL	NA	
	quantity	INTEGER	NOT NULL	NA	
	job_id	RAW(16)	NOT NULL, FOREIGN KEY (job_id) REFERENCES job(id)	NA	

Analysis					
Table	Attributes	Domain Constraints	Constraints	Constraint Name	Relations
part_service			NOT NULL, FOREIGN KEY (inventory_product_id) REFERENCES inventory_product(id)		part_service has many to one relationship with job, part_service has one to one relationship with inventory_product
	inventory_product_id	RAW(16)	NOT NULL, CHECK (mode_of_payment IN ('SERVICE', 'PART'))	NA	
	type	CHAR(7)		invalid_part_service_type	
payment_plan	id	RAW(16)	PRIMARY KEY	NA	payment_plan has one to many relationship with bill
	name	VARCHAR2(100)	NOT NULL	NA	
	installments	INTEGER	NOT NULL	NA	
	interest	INTEGER	NOT NULL	NA	
customer	id	RAW(16)	PRIMARY KEY	NA	customer has many to many relationship with insurance, customer has one to many relationship with job, customer has many to one relationship with address, customer has one to many relationship with bills
	first_name	VARCHAR2(100)	NOT NULL	NA	
	last_name	VARCHAR2(100)	NOT NULL	NA	
	dob	TIMESTAMP	NOT NULL	NA	
	driverlicense	VARCHAR2(50)	UNIQUE, NOT NULL	NA	
	phone	VARCHAR2(20)	NOT NULL, UNIQUE, CHECK (REGEXP_LIKE (phone, '^\\(\\d{3}\\)\\d{3}-\\d{4}\$'))	invalid_customer_phone	
	address_id	RAW(16)	NOT NULL, FOREIGN KEY (address_id) REFERENCES address(id)	NA	
insurance	id	RAW(16)	PRIMARY KEY	NA	insurance has many to many relationship with customer, insurance has one to many relationship with bill
	policy_type	VARCHAR2(100)	NOT NULL	NA	
	provider	VARCHAR2(100)	NOT NULL	NA	
	claim_percentage	INTEGER	NOT NULL	NA	
inventory_product_supplier					
	inventory_product_id	RAW(16)	PRIMARY KEY, NOT NULL, FOREIGN KEY (inventory_product_id) REFERENCES inventory_product(id)"	NA	NA
	supplier_id	RAW(16)	PRIMARY KEY, NOT NULL, FOREIGN KEY (supplier_id) REFERENCES supplier(id)	NA	
	quantity	INTEGER	NOT NULL	NA	
	bill_id	RAW(16)	PRIMARY KEY, NOT NULL, FOREIGN KEY (bill_id) REFERENCES bill(id)"	NA	

Analysis					
Table	Attributes	Domain Constraints	Constraints	Constraint Name	Relations
bill_inventory_product			PRIMARY KEY, NOT NULL, FOREIGN KEY (inventory_product_id) REFERENCES inventory_product(id)"		NA
	inventory_product_id	RAW(16)		NA	
	quantity	INTEGER	NOT NULL	NA	
job_employee	job_id	RAW(16)	PRIMARY KEY, NOT NULL, FOREIGN KEY (job_id) REFERENCES job(id)"	NA	NA
			PRIMARY KEY, NOT NULL, FOREIGN KEY (employee_id) REFERENCES employee(id)"	NA	
	employee_id	RAW(16)		NA	
customer_insurance	customer_id	RAW(16)	PRIMARY KEY, NOT NULL, FOREIGN KEY (customer_id) REFERENCES customer(id)"	NA	NA
			PRIMARY KEY, NOT NULL, FOREIGN KEY (insurance_id) REFERENCES insurance(id)"	NA	
	status	CHAR(8)	NOT NULL	NA	

The cloud account users for Group-8 are as follows.

ADMIN – Kanakalakshmi Murikipudi

V_11642773 – Vishnu Suidreddy - https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/v_11642773/_sdw/

SR_11667743 – Srinivas - https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/

S116644475 – Sindhu - https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/s116644475/_sdw/

The screenshot shows the Oracle Database Actions | Database Users interface. The users listed are:

- ADMIN**: ORDS Alias: admin, Password Expires in 359 days, REST Enabled. URL: https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/v_11642773/_sdw/
- BOAT_ADMIN**: (empty row)
- GGADMIN**: (empty row)
- RMAN\$VPC**: (empty row)
- S116644475**: REST Enabled, ORDS Alias: s116644475, Password Expires in 359 days. URL: https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/s116644475/_sdw/
- SR_11667743**: REST Enabled, ORDS Alias: sr_11667743, Password Expires in 359 days. URL: https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/
- V_11642773**: REST Enabled, ORDS Alias: v_11642773, Password Expires in 359 days. URL: https://gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/v_11642773/_sdw/

Page Size: 20

Powered by ORDS

Queries of Project Part-3

Query-1: List the total number of sales on December 24, 2022 group by state.

SQL Query:

```
select a.state, count(b.id) as total_sales
from SR_11667743.bill b
join SR_11667743.bill_inventory_product bip on b.id = bip.bill_id
join inventory_product ip on bip.inventory_product_id = ip.id
join SR_11667743.address a on ip.address_id = a.id
where trunc(b.bill_date) = to_date('2022-12-24', 'YYYY-MM-DD')
group by a.state;
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. On the left, the Navigator pane displays the schema structure under the ADMIN user, including tables like AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The central workspace contains an SQL query:

```
1 select a.state, count(b.id) as total_sales
2 from SR_11667743.bill b
3 join SR_11667743.bill_inventory_product bip on b.id = bip.bill_id
4 join inventory_product ip on bip.inventory_product_id = ip.id
5 join SR_11667743.address a on ip.address_id = a.id
6 where trunc(b.bill_date) = to_date('2022-12-24', 'YYYY-MM-DD')
7 group by a.state;
```

The Query Result tab shows the execution results:

STATE	TOTAL_SALES
California	2
Illinois	1
New York	1

Execution time: 0.015 seconds

Powered by ORDS

Explanation: In the SQL query, we need to list the total number of sales on December 24, 2022 group by state. To achieve this, the query filters sales data by the bill_date column, specifically looking for sales on December 24, 2022. It then groups the results by the state in which the sales occurred, calculating the total number of sales for each state.

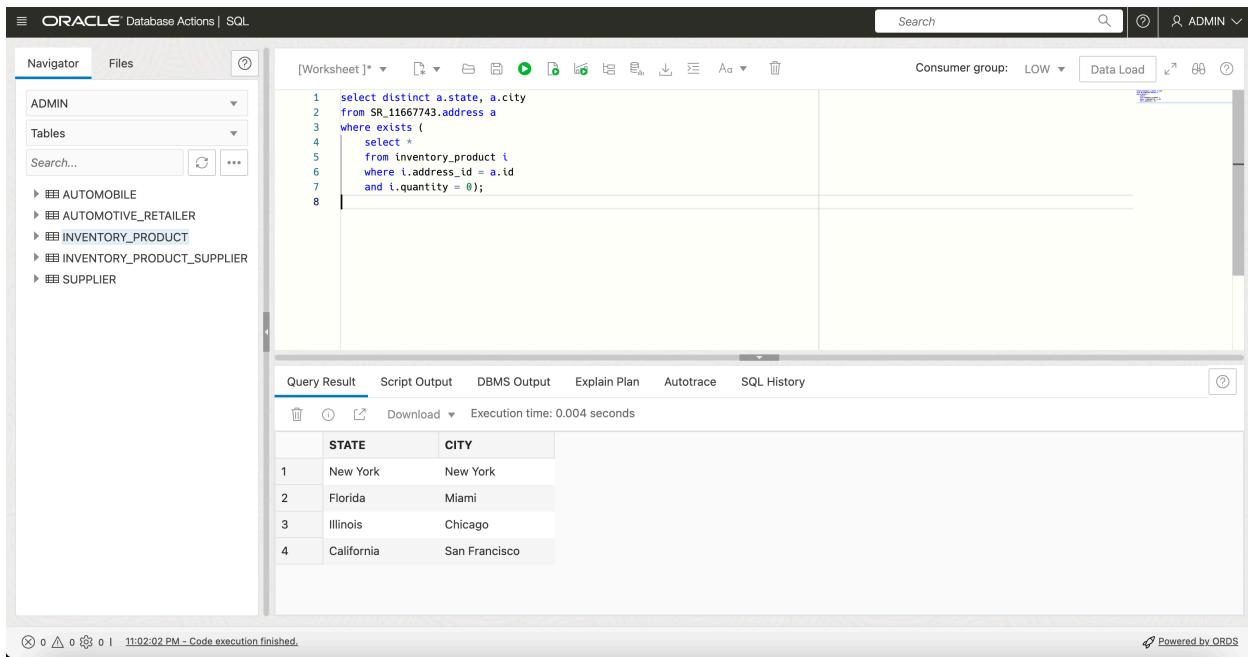
As a result of execution, we got three states in which sales happened along with the total number of sales on 24th December 2022. Those are California, Illinois and New York.

Query-2: Find locations with inventories that list at least one missing product (a product that has quantity of zero in the inventory)

SQL Query:

```
select distinct a.state, a.city
from SR_11667743.address a
where exists (
    select *
    from inventory_product i
    where i.address_id = a.id
    and i.quantity = 0);
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query window contains the SQL code provided above. The results window displays a table with four rows of data:

	STATE	CITY
1	New York	New York
2	Florida	Miami
3	Illinois	Chicago
4	California	San Francisco

Explanation: We need to retrieve locations (which includes states and cities) with inventories that contain at least one missing product, which is defined as a product with a quantity of zero in the inventory. So, this query retrieves the unique state and city pairs from the "address" table for locations where there is at least one missing product. This is achieved by linking the "address" table with the "inventory_product" table and filtering for products with zero quantity at each location.

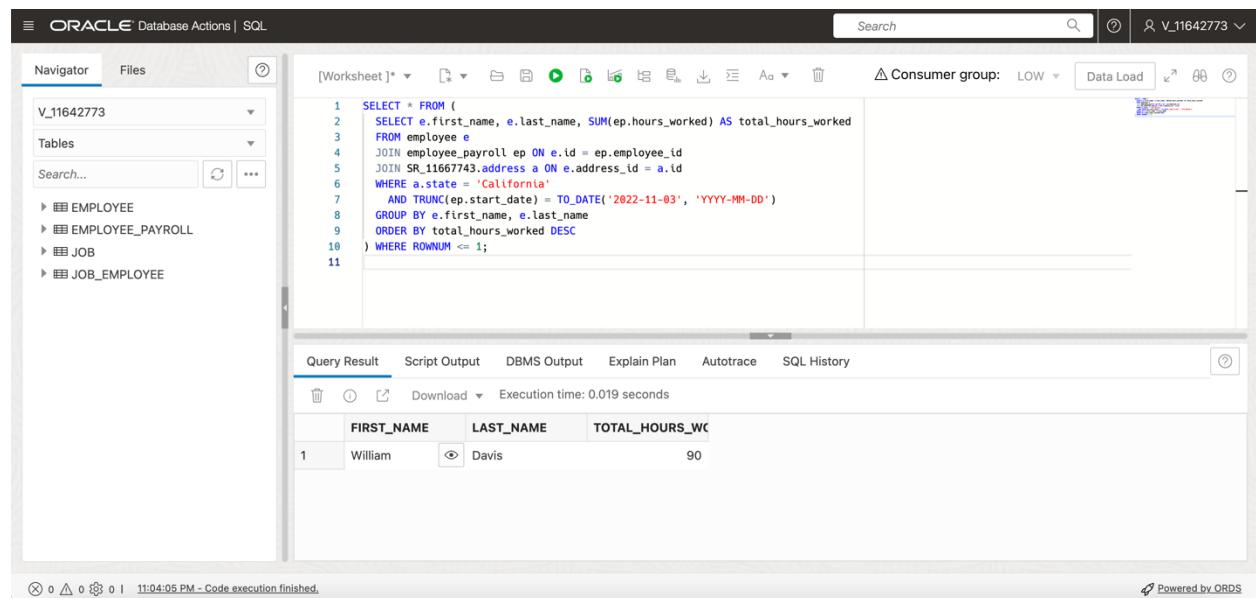
We got 4 records of locations with missing inventory product.

Query-3: Find the name of the employee(s) from California that had worked the most hours on November 3, 2022

SQL Query:

```
SELECT * FROM (
  SELECT e.first_name, e.last_name, SUM(ep.hours_worked) AS total_hours_worked
  FROM employee e
  JOIN employee_payroll ep ON e.id = ep.employee_id
  JOIN SR_11667743.address a ON e.address_id = a.id
  WHERE a.state = 'California'
  AND TRUNC(ep.start_date) = TO_DATE('2022-11-03', 'YYYY-MM-DD')
  GROUP BY e.first_name, e.last_name
  ORDER BY total_hours_worked DESC
) WHERE ROWNUM <= 1;
```

Screenshot:



The screenshot shows the Oracle Database Actions interface with a SQL worksheet. The code in the worksheet is identical to the one above. The results pane shows a single row of data:

FIRST_NAME	LAST_NAME	TOTAL_HOURS_WK
William	Davis	90

Explanation: We need to retrieve the names of all employees from California who worked the most hours on November 3, 2022. We retrieved this information by joining the employee table with the employee_payroll and address tables, filtering for the state 'California', and then grouping and ordering the results by total hours worked, returning the employee(s) with the highest hours worked on the specified date.

Query-4: List the items that currently have the least quantity on inventory.

SQL Query:

```
select ip.name, ip.quantity
from admin.inventory_product ip
where ip.quantity = (select MIN(quantity) from admin.inventory_product);
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the ADMIN schema selected, showing tables like AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The main workspace contains the SQL query:

```
1 select ip.name, ip.quantity
2   from admin.inventory_product ip
3  where ip.quantity = (select MIN(quantity) from admin.inventory_product);
```

The results tab shows the output of the query:

	NAME	QUANTITY
1	Product B	0
2	Product G	0
3	Product H	0
4	Product J	0

At the bottom, a message indicates "Execution time: 0.08 seconds".

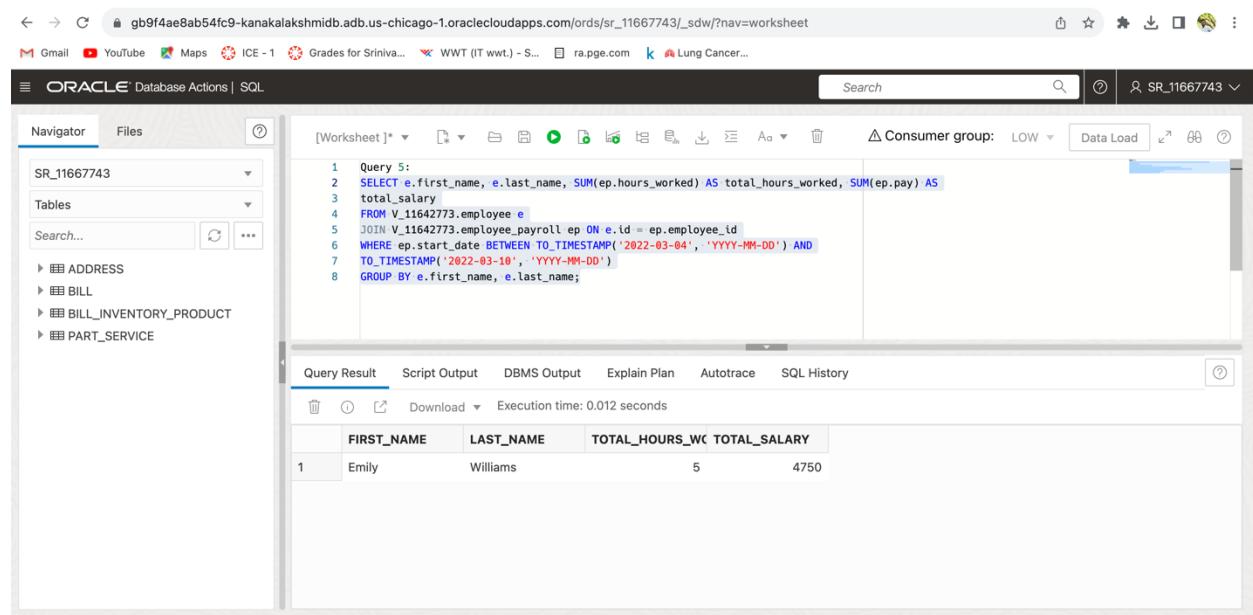
Explanation: We need to retrieves the items with the least quantity in the inventory_product table. This query finds and lists the items in the inventory with the smallest quantity. It first figures out what the minimum quantity is in the inventory, and then it selects and shows you the items that have exactly that minimum amount in quantity.

Query-5: Print the payroll from March 4, 2022 to March 10, 2022 displaying employee name, hours worked and total salary for all employees.

SQL Query:

```
SELECT e.first_name, e.last_name, SUM(ep.hours_worked) AS total_hours_worked, SUM(ep.pay) AS
total_salary
FROM V_11642773.employee e
JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
WHERE ep.start_date BETWEEN TO_TIMESTAMP('2022-03-04', 'YYYY-MM-DD') AND
TO_TIMESTAMP('2022-03-10', 'YYYY-MM-DD')
GROUP BY e.first_name, e.last_name;
```

Screenshot:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, there are links for Gmail, YouTube, Maps, ICE - 1, Grades for Sriniva..., WWT (IT wwt.) - S..., ra.page.com, and Lung Cancer... A search bar and a session identifier 'SR_11667743' are also present. The main area is titled '[Worksheet]'. On the left, the Navigator pane shows a tree view with 'SR_11667743' expanded, revealing 'Tables' and 'Search...'. Below that are collapsed sections for ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE. The central workspace contains the SQL query. The 'Query Result' tab is selected, showing a single row of data in a table:

	FIRST_NAME	LAST_NAME	TOTAL_HOURS_WK	TOTAL_SALARY
1	Emily	Williams	5	4750

The execution time is listed as 0.012 seconds.

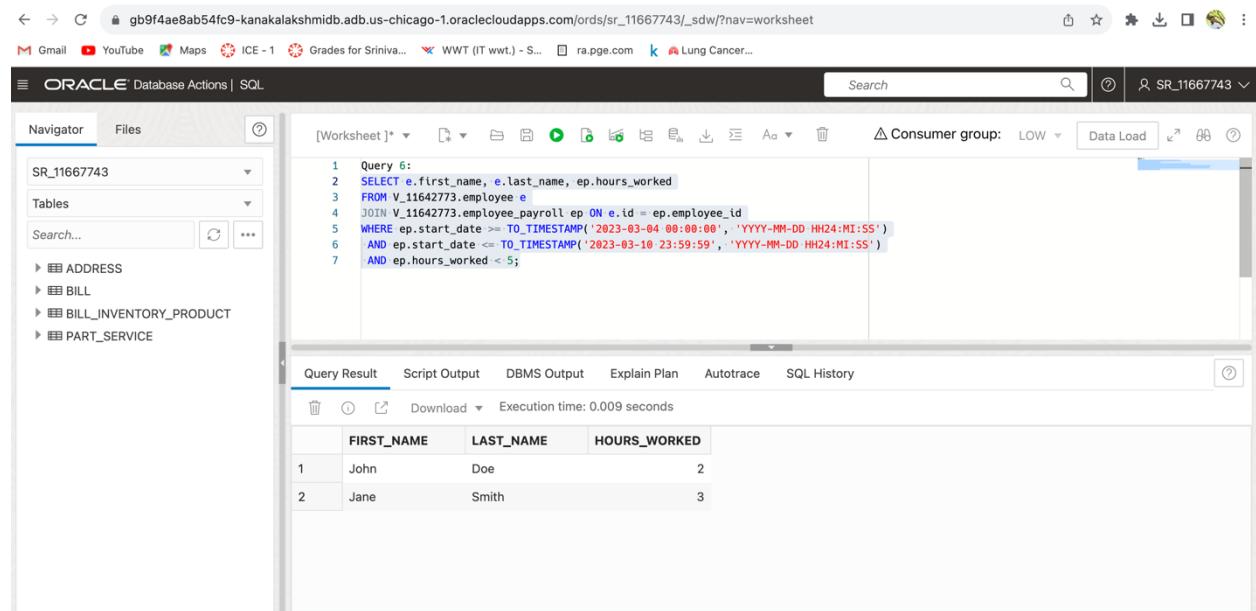
Explanation: In query, we need to show payroll data i.e total hours worked and the total salary they received for employees who worked from March 4, 2022, to March 10, 2022. It combines information from the "employee" and "employee_payroll" tables, showing each employee's first name, last name, total hours worked during that week, and their overall earnings. To achieve this, it connects the two tables by employee ID, filters the results for the specified date range, and groups the data by first name and last name to avoid duplicates.

Query-6: Design a delete statement to delete employees working less than 5 hours from March 4, 2023 to March 10, 2023.

SQL Query Before Delete:

```
SELECT e.first_name, e.last_name, ep.hours_worked
FROM V_11642773.employee e
JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
WHERE ep.start_date >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
AND ep.start_date <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
AND ep.hours_worked < 5;
```

Screenshot-1:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is `gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oracleclouddatabases.com/ords/sr_11667743/_sdw/?nav=worksheet`. Below the URL, there are several browser tabs and icons. The main area is titled "Database Actions | SQL". On the left, the "Navigator" pane shows a database named "SR_11667743" with tables like ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE. The central workspace contains the SQL query:

```
1  Query 6:
2  SELECT e.first_name, e.last_name, ep.hours_worked
3  FROM V_11642773.employee e
4  JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
5  WHERE ep.start_date >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
6  AND ep.start_date <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
7  AND ep.hours_worked < 5;
```

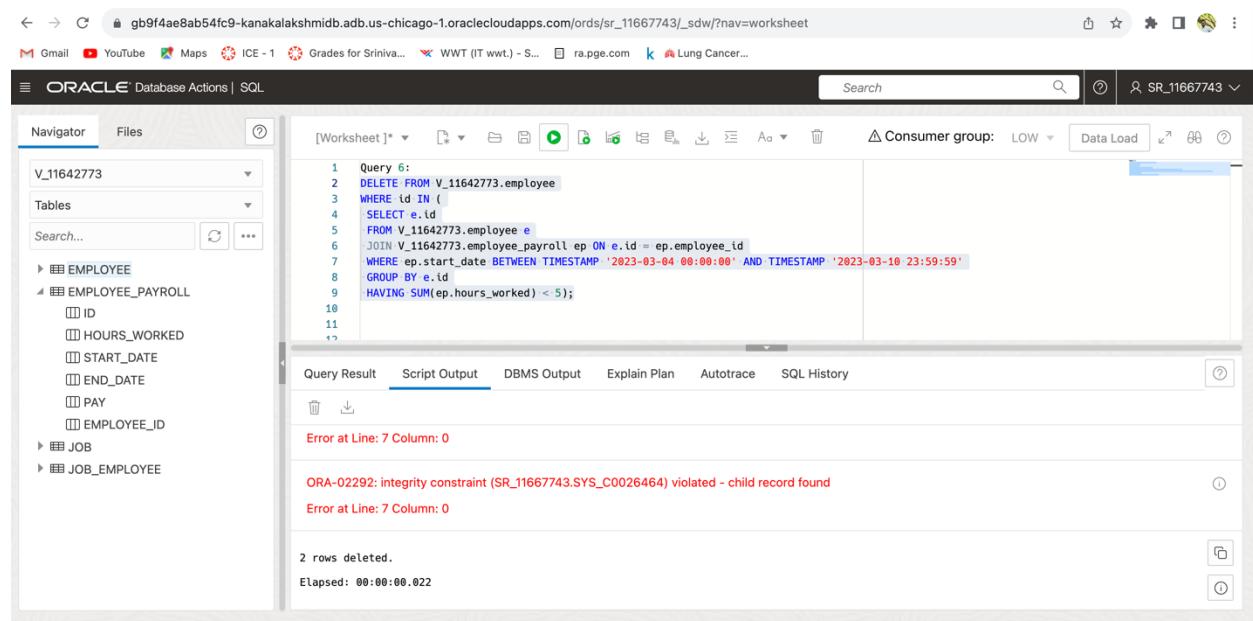
Below the query, the "Query Result" tab is selected, showing the execution time as 0.009 seconds. The result table contains two rows:

	FIRST_NAME	LAST_NAME	HOURS_WORKED
1	John	Doe	2
2	Jane	Smith	3

SQL Query to Delete:

```
DELETE FROM V_11642773.employee
WHERE id IN (
    SELECT e.id
    FROM V_11642773.employee e
    JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
    WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10 23:59:59'
    GROUP BY e.id
    HAVING SUM(ep.hours_worked) < 5);
```

Screenshot-2:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is `gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/?nav=worksheet`. Below the URL, there are several browser tabs and links. The main workspace is titled [Worksheet]. It contains the following SQL code:

```
1  Query 6:
2  DELETE FROM V_11642773.employee
3  WHERE id IN (
4    SELECT e.id
5    FROM V_11642773.employee e
6    JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
7    WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10 23:59:59'
8    GROUP BY e.id
9    HAVING SUM(ep.hours_worked) < 5);
10
11
```

The "Script Output" tab is selected. The output shows the following errors:

- Error at Line: 7 Column: 0
- ORA-02292: integrity constraint (SR_11667743.SYS_C0026464) violated - child record found
- Error at Line: 7 Column: 0

At the bottom of the output, it says "2 rows deleted." and "Elapsed: 00:00:00.022".

Screenshot-3: cross validation after deleting the records.

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables for the schema V_11642773. The main area contains a SQL query and its results.

```
1 Query 6:
2 SELECT * FROM V_11642773.employee
3 WHERE id IN (
4   SELECT e.id
5   FROM V_11642773.employee e
6   JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
7   WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10 23:59:59'
8   GROUP BY e.id
9   HAVING SUM(ep.hours_worked) < 5);
10
11
```

The results table shows the following columns: ID, FIRST_NAME, LAST_NAME, DOB, PHONE, EMAIL, ANNUAL_SALARY, and HIRE_DATE. The message "No rows selected" is displayed.

ID	FIRST_NAME	LAST_NAME	DOB	PHONE	EMAIL	ANNUAL_SALARY	HIRE_DATE
No rows selected							

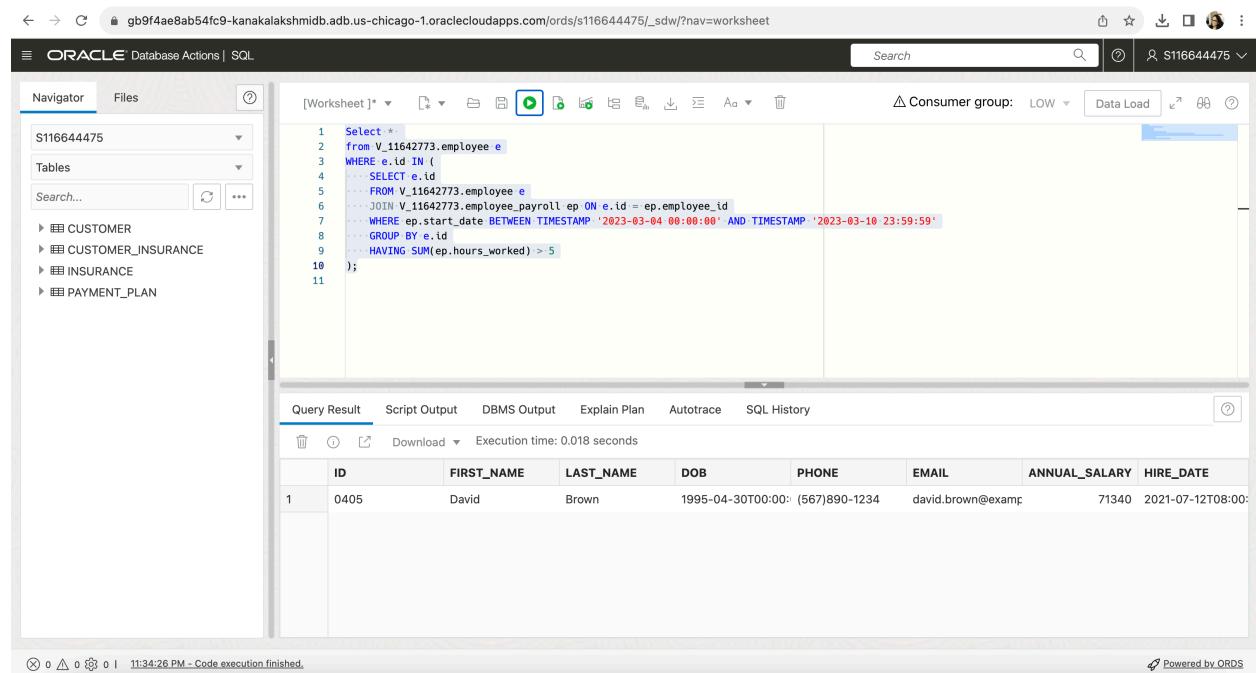
Explanation: This query is to find employees who worked less than 5 hours from March 4, 2023, to March 10, 2023. It provides with the employee details like their first names, last names, and the hours they worked during this period from table 'employee', joining it with another table employee_payroll using the employee ID as the common key. This information can be useful for monitoring and managing employee attendance and work hours.

Query-7: Design an update statement to give a 23% salary raise to employees working more than 5 hours from March 4, 2023 to March 10, 2023.

Query Before Update:

```
Select *
from V_11642773.employee e
WHERE e.id IN (
    SELECT e.id
    FROM V_11642773.employee e
    JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
    WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10
23:59:59'
    GROUP BY e.id
    HAVING SUM(ep.hours_worked) > 5
);
```

Screenshot-1:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The top navigation bar includes links for Navigator, Files, and a search bar. The main workspace displays the SQL query provided above. Below the query, the 'Query Result' tab is selected, showing a single row of data from the employee table. The data is as follows:

ID	FIRST_NAME	LAST_NAME	DOB	PHONE	EMAIL	ANNUAL_SALARY	HIRE_DATE	
1	0405	David	Brown	1995-04-30T00:00:00	(567)890-1234	david.brown@example.com	71340	2021-07-12T08:00:00

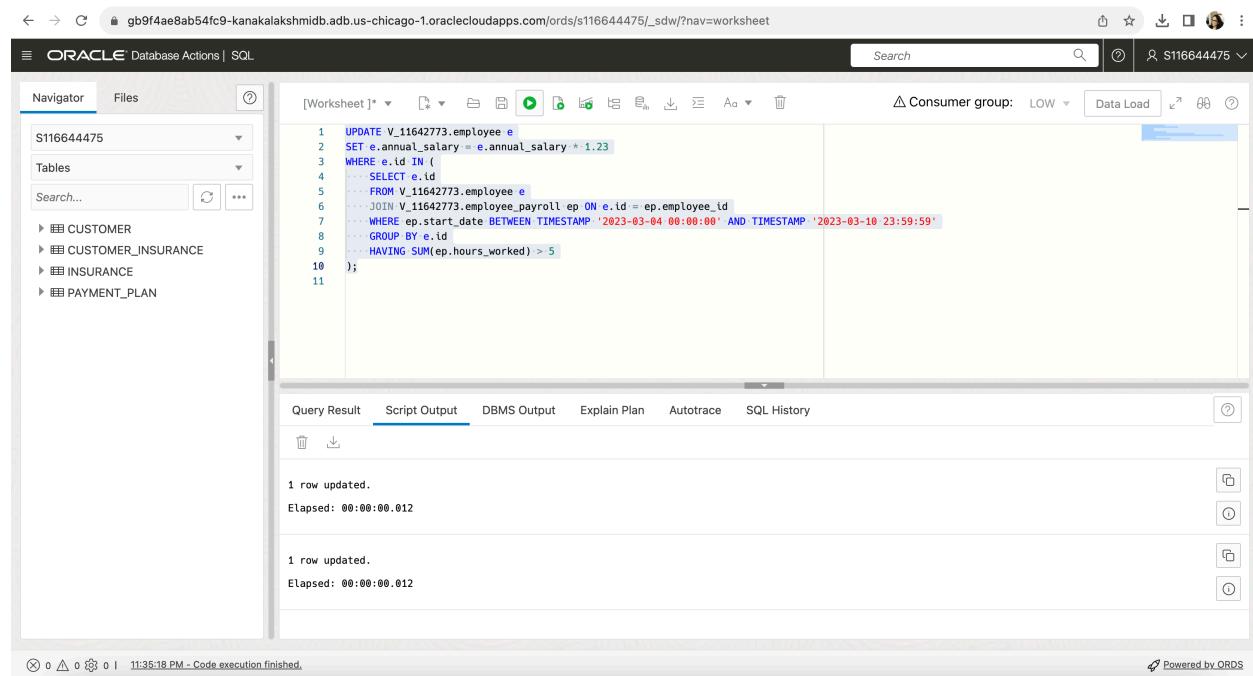
The bottom status bar indicates the code execution finished at 11:34:26 PM.

Explanation: As we have to design an update statement to give a 23% salary raise to employees working more than 5 hours from March 4, 2023 to March 10, 2023, first we are selecting all the employees who worked more than 5 hours from March 4, 2023, to March 10, 2023 by analyzing employee records and their corresponding payroll data, filtering out those who meet the specified working hours criteria.

SQL Query to update:

```
UPDATE V_11642773.employee e
SET e.annual_salary = e.annual_salary * 1.23
WHERE e.id IN (
    SELECT e.id
    FROM V_11642773.employee e
    JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
    WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10
23:59:59'
    GROUP BY e.id
    HAVING SUM(ep.hours_worked) > 5
);
```

Screenshot-2:



The screenshot shows the Oracle Database Actions SQL worksheet interface. The query is pasted into the worksheet area, and the 'Script Output' tab is selected at the bottom. The output shows two rows updated, each taking approximately 0.012 seconds.

```
1 UPDATE V_11642773.employee e
2 SET e.annual_salary = e.annual_salary * 1.23
3 WHERE e.id IN (
4     SELECT e.id
5     FROM V_11642773.employee e
6     JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
7     WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10 23:59:59'
8     GROUP BY e.id
9     HAVING SUM(ep.hours_worked) > 5
10 );
11
```

Query Result Script Output DBMS Output Explain Plan Autotrace SQL History

1 row updated.
Elapsed: 00:00:00.012

1 row updated.
Elapsed: 00:00:00.012

Explanation: This query is to update 23% salary raise to employees who worked more than 5 hours between March 4, 2023, and March 10, 2023. It operates on a database with two tables: "employee" and "employee_payroll." It first checks if an employee worked more than 5 hours between March 4, 2023, and March 10, 2023, by examining records in the "employee_payroll" table. Then, for each eligible employee, it multiplies their annual salary by 1.23 to grant them a 23% salary raise.

SQL Query After update:

```
Select *
from V_11642773.employee e
WHERE e.id IN (
    SELECT e.id
    FROM V_11642773.employee e
    JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
    WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10
23:59:59'
    GROUP BY e.id
    HAVING SUM(ep.hours_worked) > 5
);
```

Screenshot-3:

The screenshot shows the Oracle Database Actions SQL worksheet interface. The top navigation bar includes links for Navigator, Files, and a search bar. The main workspace has tabs for Worksheet, Consumer group: LOW, Data Load, and a toolbar with various icons. The Worksheet tab is active, displaying the SQL query. The code is as follows:

```
1 Select *
2 from V_11642773.employee e
3 WHERE e.id IN (
4     SELECT e.id
5     FROM V_11642773.employee e
6     JOIN V_11642773.employee_payroll ep ON e.id = ep.employee_id
7     WHERE ep.start_date BETWEEN TIMESTAMP '2023-03-04 00:00:00' AND TIMESTAMP '2023-03-10 23:59:59'
8     GROUP BY e.id
9     HAVING SUM(ep.hours_worked) > 5
10 );
11
```

Below the code, the Query Result tab is selected, showing the execution time (0.005 seconds). The result set is a table with columns: ID, FIRST_NAME, LAST_NAME, DOB, PHONE, EMAIL, ANNUAL_SALARY, and HIRE_DATE. One row is displayed:

ID	FIRST_NAME	LAST_NAME	DOB	PHONE	EMAIL	ANNUAL_SALARY	HIRE_DATE	
1	0405	David	Brown	1995-04-30T00:00:00	(567)890-1234	david.brown@example.com	87748	2021-07-12T08:00

At the bottom left, there are status icons for errors and warnings, and a message: "11:35:48 PM - Code execution finished." At the bottom right, it says "Powered by ORDS".

Explanation: Here we are testing the previous update. This query fetches details about employees who have worked for more than 5 hours between March 4, 2023, and March 10, 2023. It's designed to provide a list of employees who have put in significant hours during that period. When compared the before update data and after update data, the salary got increased.

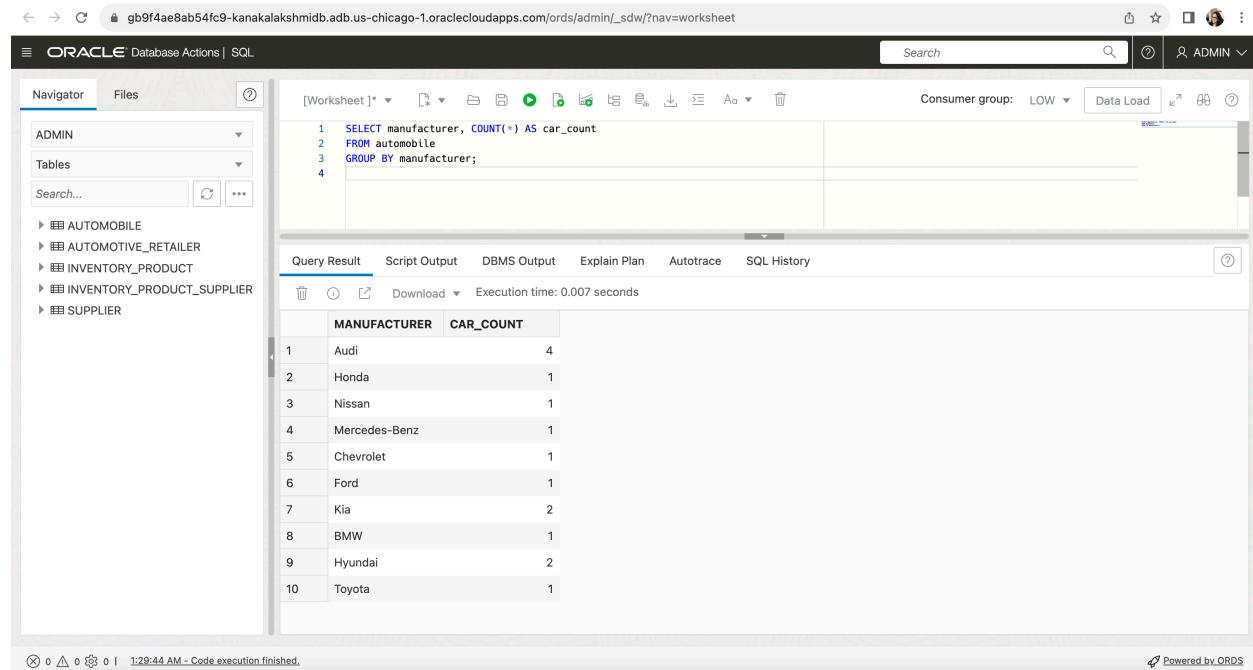
10 Select Queries:

Query-1: List manufacturer, count of each manufacturer produce.

SQL Query:

```
SELECT manufacturer, COUNT(*) AS car_count
FROM automobile
GROUP BY manufacturer;
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. On the left, the Navigator pane displays database objects like ADMIN, Tables, and various schema tables such as AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The central workspace contains the SQL query:

```
1 SELECT manufacturer, COUNT(*) AS car_count
2 FROM automobile
3 GROUP BY manufacturer;
4
```

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

MANUFACTURER	CAR_COUNT
Audi	4
Honda	1
Nissan	1
Mercedes-Benz	1
Chevrolet	1
Ford	1
Kia	2
BMW	1
Hyundai	2
Toyota	1

Execution time: 0.007 seconds

Explanation: The query retrieves data from a table called "automobile." It calculates the number of cars produced by each manufacturer and groups the results by the "manufacturer" column. The query uses the COUNT(*) function to count the number of rows (representing cars) for each distinct manufacturer. The results are presented with two columns: "manufacturer" and "car_count," showing the name of the manufacturer and the total count of cars they have produced, respectively.

Query-2: list automobiles from the year 2022 are gray in color

SQL Query: SQL Query - SELECT * FROM automobile WHERE year = '2022' AND color = 'Gray';

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query entered is:

```
1 --list automobiles from the year 2022 are gray in color
2 SELECT * FROM automobile WHERE year = '2022' AND color = 'Gray';
```

The results are displayed in a table:

ID	MANUFACTURER	NAME	VARIANT	YEAR	COLOR
11	Audi	Q5	Premium	2022	Gray
12	Kia	Q5	Premium	2022	Gray
13	Audi	Q5	Premium	2022	Gray
14	Audi	Q5	Premium	2022	Gray
15	Hyundai	Q5	Premium	2022	Gray
10	Audi	Q5	Premium	2022	Gray

Execution time: 0.007 seconds

Explanation: The SQL query provided is designed to retrieve information about automobiles from the year 2022 that have a gray color. It uses the "SELECT" statement to specify the columns to be retrieved, the "FROM" clause to specify the table named "automobile," and the "WHERE" clause to filter the results based on two conditions: the year being 2022 and the color being gray.

Query-3: list the addresses of all the automotive retailers.

SQL Query:

```
SELECT r.name AS retailer_name, a.apartment_no, a.street, a.city, a.state, a.country, a.zip  
FROM automotive_retailer r  
JOIN SR_11667743.address a ON r.address_id = a.id;
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, it says "ORACLE Database Actions | SQL". The main area has a "Worksheet" tab open. The code entered is:

```
1 --list the addresses of all the automotive retailers  
2  
3 SELECT r.name AS retailer_name, a.apartment_no, a.street, a.city, a.state, a.country, a.zip  
4 FROM automotive_retailer r  
5 JOIN SR_11667743.address a ON r.address_id = a.id;
```

Below the code, the "Query Result" tab is selected, showing the execution time was 0.01 seconds. The results are displayed in a table:

	RETAILER_NAME	APARTMENT_NO	STREET	CITY	STATE	COUNTRY	ZIP
1	autozone retailer1	101	123 Main St	Los Angeles	California	USA	90001
2	autozone retailer2	202	456 Elm St	New York	New York	USA	10001
3	autozone retailer3	303	789 Oak St	San Francisco	California	USA	94101
4	autozone retailer4	404	101 Pine St	Chicago	Illinois	USA	60601
5	autozone retailer5	505	567 Maple St	Miami	Florida	USA	33101
6	autozone retailer6	606	202 Cedar St	Houston	Texas	USA	77001
7	autozone retailer7	707	111 Birch St	Seazle	Washington	USA	98101
8	autozone retailer8	808	777 Spruce St	Los Angeles	California	USA	02201
9	autozone retailer9	909	404 Redwood St	Phoenix	Arizona	USA	85001
10	autozone retailer10	1010	999 Oak St	San Francisco	California	USA	80201

At the bottom of the worksheet, it says "11:52:13 PM - Code execution finished." and "Powered by ORDS".

Explanation: The query is to obtain a list of automotive retailers and their respective addresses from the database. It accomplishes this by connecting the "automotive_retailer" table with the "address" table through a shared address ID. This way, we can compile a comprehensive list of automotive retailer names along with address details such as apartment number, street, city, state, country, and ZIP code for each retailer's location.

Query-4: Write a Query to retrieves retailers located in the state of 'California'.

SQL Query:

```
SELECT name, phone, email, website  
FROM admin.automotive_retailer  
WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE state = 'California');
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. On the left, the Navigator pane displays database objects like CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, PAYMENT_PLAN, and various SYS_C0026341 through 413. The central workspace contains the SQL query:

```
--query retrieves retailers located in the state of 'California'  
SELECT name, phone, email, website  
FROM admin.automotive_retailer  
WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE state = 'California');
```

The bottom pane shows the Query Result table with four rows of data:

	NAME	PHONE	EMAIL	WEBSITE
1	autozone retailer1	(123)456-7890	retailer1@example.c	www.retailer1.com
2	autozone retailer3	(345)678-9012	retailer3@example.c	www.retailer3.com
3	autozone retailer8	(890)123-4567	retailer8@example.c	www.retailer8.com
4	autozone retailer10	(012)345-6789	retailer10@example.c	www.retailer10.com

Execution time: 0.008 seconds

Explanation: This query fetches details of automotive retailers in California. It retrieves their names, phone numbers, emails, and websites. To identify retailers in California, it first finds the address IDs of those in the state from the 'address' table, and then it retrieves the relevant information from the 'automotive_retailer' table.

Query-5: List the names of retailers who have products with the highest prices.

SQL Query:

```
SELECT r.name AS retailer_name
FROM admin.automotive_retailer r
JOIN admin.inventory_product p ON r.id = p.automotive_retailer_id
WHERE p.price = (SELECT MAX(price) FROM admin.inventory_product);
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The top navigation bar includes 'Database Actions | SQL', 'Search', and a session ID 'S116644475'. The main area has tabs for 'Navigator' (selected), 'Files', and a toolbar with various icons. The 'Worksheet' tab is active, displaying the following SQL code:

```
1 --List the names of retailers who have products with the highest prices.
2
3 SELECT r.name AS retailer_name
4 FROM admin.automotive_retailer r
5 JOIN admin.inventory_product p ON r.id = p.automotive_retailer_id
6 WHERE p.price = (SELECT MAX(price) FROM admin.inventory_product);
```

Below the code, the 'Query Result' tab is selected, showing the output:

RETAILER_NAME
1 autozone retailer2
2 autozone retailer10
3 autozone retailer3

Execution time: 0.025 seconds

At the bottom left, there are status icons: a red circle with an 'X', a green triangle, a yellow star, and a blue circle. The message '11:56:52 PM - Code execution finished.' is displayed. On the right, it says 'Powered by ORDS'.

Explanation: The query identifies and lists the names of retailers who sell products with the highest prices within the "automotive" category. It achieves this by linking retailer and product information and then selecting the retailers whose product prices are equal to the maximum price found in the inventory. This helps to identify which retailers offer the most expensive products in the automotive category.

Query-6: list Retailers and Their Business Hours Sorted.

SQL Query:

```
SELECT r.name AS retailer_name, r.business_hours  
FROM admin.automotive_retailer r  
ORDER BY TO_TIMESTAMP(SUBSTR(r.business_hours, 1, 8), 'HH:MI AM') ASC;
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, it says "ORACLE Database Actions | SQL". On the left, there's a "Navigator" pane showing database objects like "CUSTOMER", "CUSTOMER_INSURANCE", "INSURANCE", and "PAYMENT_PLAN". The main workspace contains a "Worksheet" tab with the following SQL code:

```
1 --list Retailers and Their Business Hours Sorted by Opening Time  
2  
3 SELECT r.name AS retailer_name, r.business_hours  
4 FROM admin.automotive_retailer r  
5 ORDER BY TO_TIMESTAMP(SUBSTR(r.business_hours, 1, 8), 'HH:MI AM') ASC;
```

Below the code, the "Query Result" tab is selected, showing the output:

	RETAILER_NAME	BUSINESS_HOURS
1	autozone retailer5	8:00 AM - 8:00 PM
2	autozone retailer9	8:00 AM - 7:00 PM
3	autozone retailer6	8:30 AM - 6:30 PM
4	autozone retailer2	8:30 AM - 7:00 PM
5	autozone retailer7	9:00 AM - 7:30 PM
6	autozone retailer1	9:00 AM - 6:00 PM
7	autozone retailer4	9:30 AM - 6:30 PM
8	autozone retailer10	9:30 AM - 5:30 PM
9	autozone retailer3	10:00 AM - 5:30 PM
10	autozone retailer8	10:00 AM - 6:00 PM

At the bottom of the interface, it says "Powered by ORDS".

Explanation: The SQL query retrieves data from a table named "automotive_retailer" in a database schema named "admin." It selects two columns, "name" (aliased as "retailer_name") and "business_hours" from the table. The query then orders the results in ascending order based on the business hours after converting them to a timestamp format.

Query-7: list the names and quantities of products in stock for each automotive retailer

SQL Query:

```
SELECT r.name AS retailer_name, p.name AS product_name, p.quantity
FROM admin.automotive_retailer r
JOIN admin.inventory_product p ON r.id = p.automotive_retailer_id
WHERE p.quantity > 0
ORDER BY r.name, p.name;
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query window contains the SQL code for listing products in stock. The results window displays a table with columns RETAILER_NAME, PRODUCT_NAME, and QUANTITY, showing data for eight different retailers and their respective products and quantities.

	RETAILER_NAME	PRODUCT_NAME	QUANTITY
1	autozone retailer	Product A	100
2	autozone retailer10	Product J	95
3	autozone retailer2	Product H	70
4	autozone retailer2	Product I	130
5	autozone retailer3	Product C	120
6	autozone retailer4	Product D	90
7	autozone retailer5	Product E	10
8	autozone retailer6	Product F	80

Explanation: This query provides a clear overview of the available products for each automotive retailer. It lists the retailer names along with the names of products they have in stock, along with the respective quantities. So, It filters the results to only include rows where the product quantity is greater than 0 and orders the results first by retailer name and then by product name. This information is helpful for inventory management and allows retailers to monitor their stock levels effectively.

Query-8: List the first and last names of all employees whose first names start with 'J'

SQL Query:

```
SELECT first_name, last_name  
FROM Employee  
WHERE first_name LIKE 'J%';
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the connection name S116644475 and tables CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The main area has tabs for Worksheet, Script Output, DBMS Output, Explain Plan, Autotrace, and SQL History. The Worksheet tab contains the following SQL code:

```
-- 8. List the first and last names of all employees whose first names start with 'J'  
SELECT first_name, last_name  
FROM Employee  
WHERE first_name LIKE 'J%';
```

The Query Result tab shows the execution time as 0.006 seconds and displays the results in a table:

	FIRST_NAME	LAST_NAME
1	James	Garcia

At the bottom, a status message indicates "11:47:49 PM - Code execution finished."

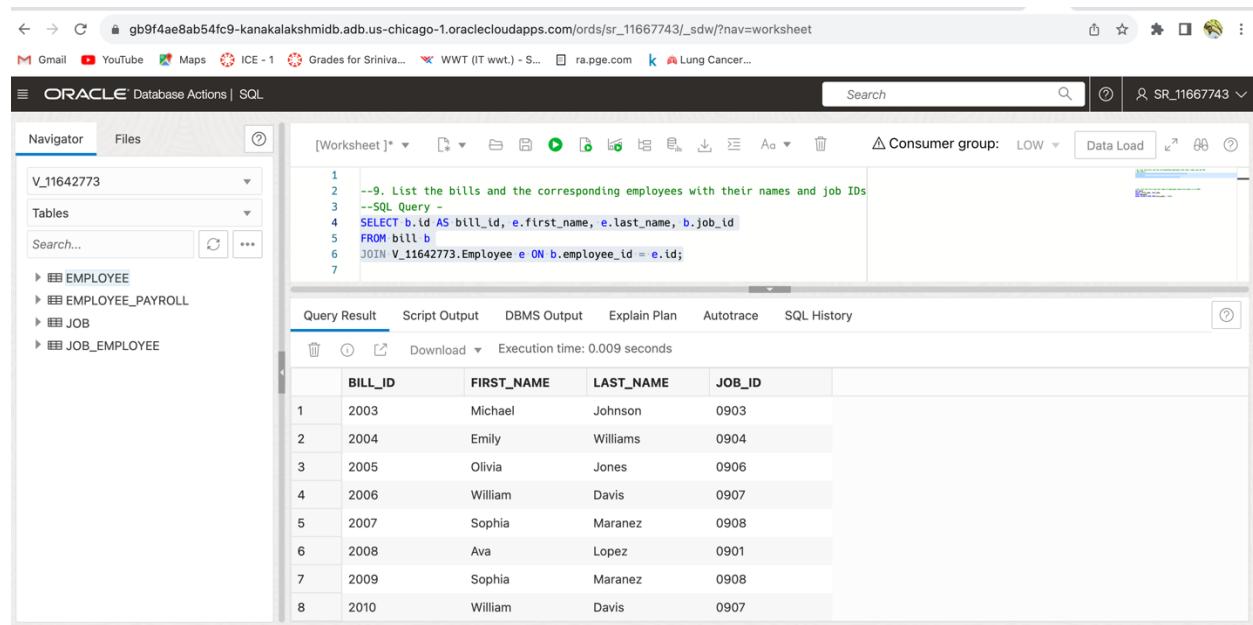
Explanation: This query is like a filter for employees with names that start with 'J.' It finds and lists their first and last names from the Employee table. It's a handy way to quickly identify all the employees whose first names begin with 'J'. We can identify all employees alphabetically by changing the character.

Query-9: List the bills and the corresponding employees with their names and job IDs

SQL Query:

```
SELECT b.id AS bill_id, e.first_name, e.last_name, b.job_id  
FROM bill b  
JOIN V_11642773.Employee e ON b.employee_id = e.id;
```

Screenshot:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is `gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/?nav=worksheet`. Below the URL, there are several browser tabs and icons. The main window has a title bar "ORACLE Database Actions | SQL". On the left, a "Navigator" panel shows a database named "V_11642773" with tables "EMPLOYEE", "EMPLOYEE_PAYROLL", "JOB", and "JOB_EMPLOYEE". The central workspace contains a SQL worksheet with the following content:

```
[Worksheet]*  
1 --9. List the bills and the corresponding employees with their names and job IDs  
2 --SQL Query -  
3 SELECT b.id AS bill_id, e.first_name, e.last_name, b.job_id  
4 FROM bill b  
5 JOIN V_11642773.Employee e ON b.employee_id = e.id;  
6  
7
```

Below the worksheet, a "Query Result" table displays the following data:

BILL_ID	FIRST_NAME	LAST_NAME	JOB_ID
1 2003	Michael	Johnson	0903
2 2004	Emily	Williams	0904
3 2005	Olivia	Jones	0906
4 2006	William	Davis	0907
5 2007	Sophia	Maranez	0908
6 2008	Ava	Lopez	0901
7 2009	Sophia	Maranez	0908
8 2010	William	Davis	0907

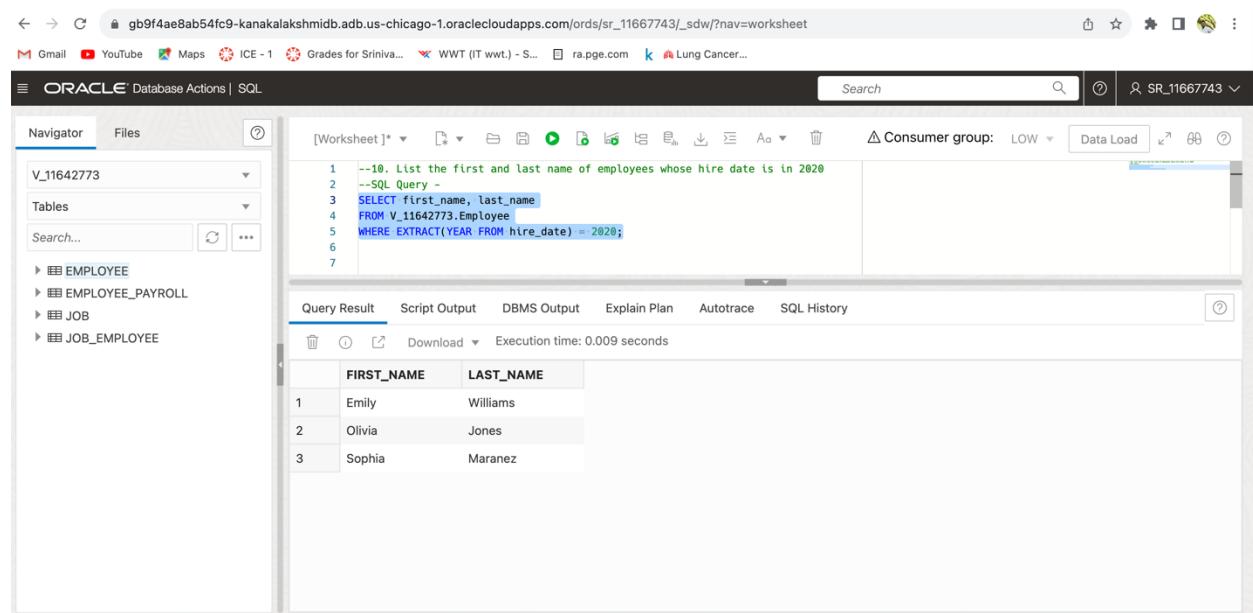
Explanation: The query combines information from two tables, bills, and employees. It links bills to the employees responsible for them by matching their employee IDs. The result includes a list of bills with their IDs, the first and last names of the employees handling them, and their respective job IDs. This query helps provide insight into which employees are associated with specific bills.

Query-10: List the first and last name of employees whose hire date is in 2020

SQL Query:

```
SELECT first_name, last_name  
FROM V_11642773.Employee  
WHERE EXTRACT(YEAR FROM hire_date) = 2020;
```

Screenshot:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is `gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw?nav=worksheet`. Below the URL, there are several browser tabs and icons. The main window is titled "ORACLE Database Actions | SQL". On the left, the "Navigator" pane shows a database connection named "V_11642773" with tables like EMPLOYEE, EMPLOYEE_PAYROLL, JOB, and JOB_EMPLOYEE listed. The central workspace contains a SQL query in the "Worksheet" tab:

```
--18. List the first and last name of employees whose hire date is in 2020  
--SQL Query -  
3 SELECT first_name, last_name  
4 FROM V_11642773.Employee  
5 WHERE EXTRACT(YEAR FROM hire_date) = 2020;  
6  
7
```

The "Query Result" tab shows the output of the query:

	FIRST_NAME	LAST_NAME
1	Emily	Williams
2	Olivia	Jones
3	Sophia	Maranez

Explanation: This query selects the first name and last name of employees from a table named "Employee." It filters the results based on the hire date of employees. Specifically, it retrieves employees whose hire date is in the year 2020. It does this by looking at the hire date for each employee and filtering out those who joined the company during that specific year. The query utilizes the EXTRACT function to extract the year from the hire date and then compares it to the value 2020.

7 Update Queries:

Query-1: Update the annual salary of an employee to 75000 with the first name 'John' and last name 'Doe' in the "employee" table.

SQL Query Before update: Select Query Before Update: select * from V_11642773.employee;

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, with 'Tables' expanded to show 'AUTOMOBILE', 'AUTOMOTIVE_RETAILER', 'INVENTORY_PRODUCT', 'INVENTORY_PRODUCT_SUPPLIER', and 'SUPPLIER'. The main workspace contains a SQL script and its execution results.

[Worksheet] * | Consumer group: LOW | Data Load

Search | ADMIN

```
1 -- Update the annual salary of an employee with the first name 'John' and last name 'Doe' in the "employee" table
2
3 select * from V_11642773.employee;
```

Query Result | Script Output | DBMS Output | Explain Plan | Autotrace | SQL History

ID	FIRST_NAME	LAST_NAME	DOB	PHONE	EMAIL	ANNUAL_SALARY	HIRE_DATE	
1	0403	Michael	Johnson	1992-01-10T00:00:00+00:00	(345)678-9012	michael.johnson@example.com	55000	2021-03-05T10:15:00+00:00
2	0404	Emily	Williams	1988-09-05T00:00:00+00:00	(456)789-0123	emily.williams@example.com	70000	2020-11-18T09:45:00+00:00
3	0405	David	Brown	1995-04-30T00:00:00+00:00	(567)890-1234	david.brown@example.com	87748	2021-07-12T08:00:00+00:00
4	0406	Olivia	Jones	1987-12-25T00:00:00+00:00	(678)901-2345	olivia.jones@example.com	72000	2020-02-15T10:30:00+00:00
5	0407	William	Davis	1991-06-18T00:00:00+00:00	(789)012-3456	william.davis@example.com	62000	2019-04-05T09:15:00+00:00
6	0408	Sophia	Maranez	1989-10-12T00:00:00+00:00	(890)123-4567	sophia.maranez@example.com	68000	2020-09-20T08:45:00+00:00
7	0409	James	Garcia	1994-08-07T00:00:00+00:00	(901)234-5678	james.garcia@example.com	59000	2021-12-03T09:30:00+00:00
8	0410	Ava	Lopez	1986-02-28T00:00:00+00:00	(012)345-6789	ava.lopez@example.com	75000	2018-06-22T08:00:00+00:00

Execution time: 0.005 seconds

Powered by ORDS

Explanation: Before updating the table, this is the data in employees table.

SQL Query to Update:

```
UPDATE V_11642773.employee  
SET annual_salary = 75000  
WHERE first_name = 'Michael' AND last_name = 'Johnson';
```

Screenshot:

The screenshot shows the Oracle Database Actions interface with the SQL tab selected. The Navigator panel on the left lists various schema objects like ADMIN, Tables, and several fact tables (AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, SUPPLIER). The central workspace contains the following SQL code:

```
-- Update the annual salary of an employee with the first name 'John' and last name 'Doe' in the "employee" table  
UPDATE V_11642773.employee  
SET annual_salary = 75000  
WHERE first_name = 'Michael' AND last_name = 'Johnson';
```

Below the code, the Script Output tab is active, showing the results of the execution:

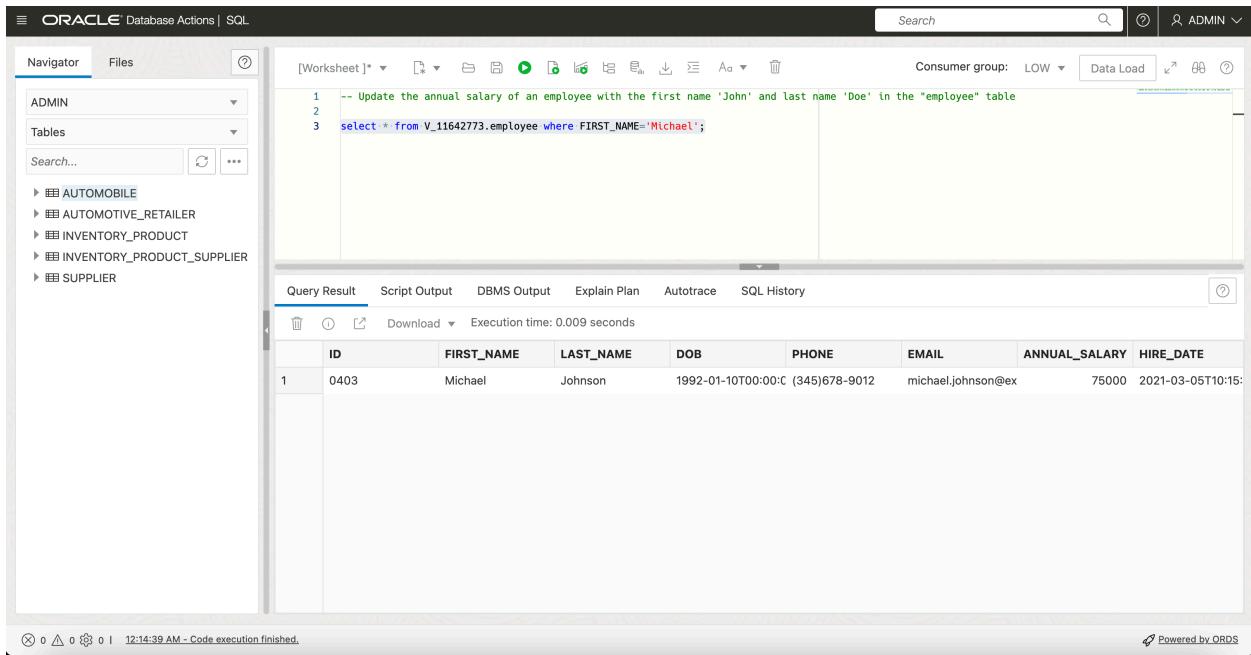
```
1 row updated.  
Elapsed: 00:00:00.006
```

At the bottom of the interface, a status bar indicates "Powered by ORDS".

Explanation: Through this SQL query, we are changing the annual salary of an employee named Michael Johnson to \$75,000 in a database called "V_11642773." It's a way to update the salary information for a specific person in the employee database.

SQL Query After Update: select * from V_11642773.employee where FIRST_NAME='Michael';

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, with the ADMIN table selected. The main workspace contains the following SQL code:

```
1 -- Update the annual salary of an employee with the first name 'John' and last name 'Doe' in the "employee" table
2
3 select * from V_11642773.employee where FIRST_NAME='Michael';
```

The results pane shows the output of the query:

ID	FIRST_NAME	LAST_NAME	DOB	PHONE	EMAIL	ANNUAL_SALARY	HIRE_DATE	
1	0403	Michael	Johnson	1992-01-10T00:00:00+00:00	(345)678-9012	michael.johnson@ex	75000	2021-03-05T10:15:00+00:00

Below the results, a message indicates the execution was completed successfully: "12:14:39 AM - Code execution finished."

Explanation: Checking whether the data is updated or not after performing update.

Query-2: Update the policy claim percentage of Disability Insurance to 70

SQL Query Before Update: select * from S116644475.insurance

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is gb9f4ae8ab54fc9-kankakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/?nav=worksheet. The title bar says "ORACLE Database Actions | SQL". The left sidebar shows a Navigator with "SR_11667743" selected, and a list of tables: ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE. The main area has a "Worksheet" tab open. The code in the worksheet is:

```
1 --2. Update the policy claim percentage of Disability Insurance to 70
2 --SQL Query-
3
4 --Before update
5 select * from S116644475.insurance
6
7
8
```

Below the code, the "Query Result" tab is selected, showing the following data:

ID	POLICY_TYPE	PROVIDER	CLAIM_PERCENTAGE
04	Home Insurance	Shield Insurance Sol	75
05	Travel Insurance	Global Insure	70
06	Pet Insurance	Pawsurance	95
07	Business Insurance	Corporate Insurers Ir	60
08	Dental Insurance	SmileCare Insurance	85
09	Property Insurance	Secure Properties In	70
10	Disability Insurance	AbilitySure Insurance	80

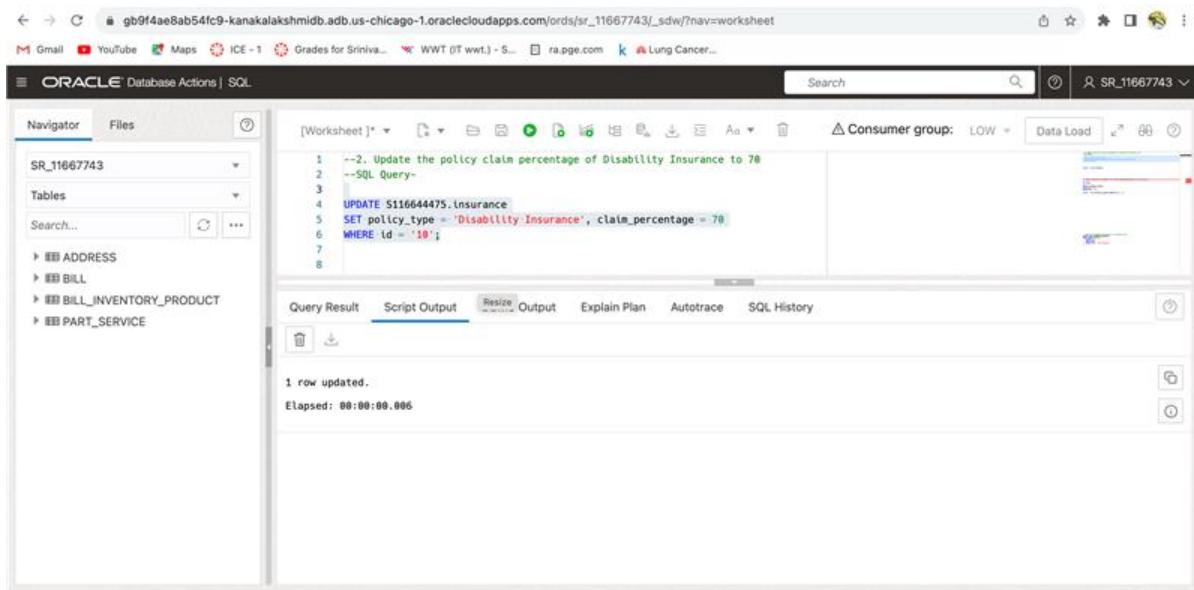
Execution time: 0.005 seconds.

Explanation: This query is to display the information in insurance table. This data is before updating the table.

SQL Query to Update:

```
UPDATE S116644475.insurance
SET policy_type = 'Disability Insurance', claim_percentage = 70
WHERE id = '10';
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL interface. In the Navigator pane, the schema 'SR_11667743' is selected. The Tables section lists 'ADDRESS', 'BILL', 'BILL_INVENTORY_PRODUCT', and 'PART_SERVICE'. The main workspace contains a SQL query window with the following content:

```
--2. Update the policy claim percentage of Disability Insurance to 70
--SQL Query-
UPDATE S116644475.insurance
SET policy_type = 'Disability Insurance', claim_percentage = 70
WHERE id = '10';
```

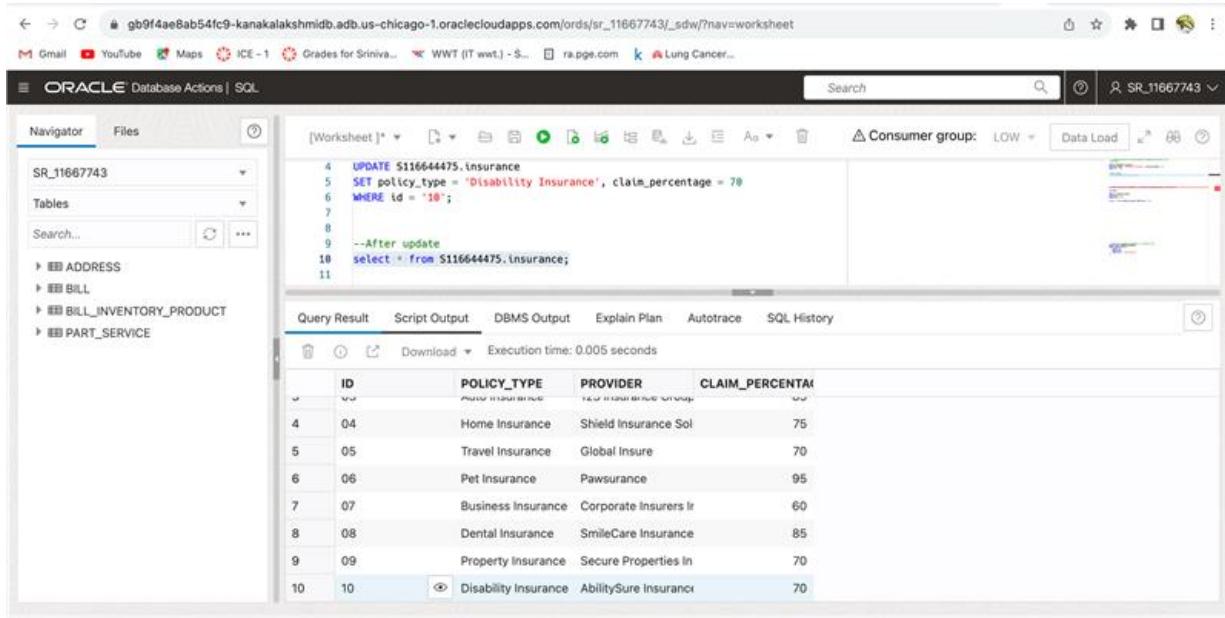
The 'Query Result' tab is selected, showing the output:

```
1 row updated.
Elapsed: 00:00:00.006
```

Explanation: This query updates an entry in the insurance database. It changes the insurance policy type to 'Disability Insurance' and adjusts the claim percentage to 70% for the policy with ID 10. This could be important for managing and customizing insurance policy details in a database.

SQL Query After Update: select * from S116644475.insurance;

Screenshot:



The screenshot shows the Oracle Database Actions SQL interface after running the update query. The Navigator pane shows the schema 'SR_11667743' and tables 'ADDRESS', 'BILL', 'BILL_INVENTORY_PRODUCT', and 'PART_SERVICE'. The main workspace contains a SQL query window with the following content:

```
4 UPDATE S116644475.insurance
5 SET policy_type = 'Disability Insurance', claim_percentage = 70
6 WHERE id = '10';
7
8
9 --After update
10 select * from S116644475.insurance;
11
```

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

```
Execution time: 0.005 seconds
```

ID	POLICY_TYPE	PROVIDER	CLAIM_PERCENTAGE
04	Home Insurance	Shield Insurance Sol	75
05	Travel Insurance	Global Insure	70
06	Pet Insurance	Pawsurance	95
07	Business Insurance	Corporate Insurers Ir	60
08	Dental Insurance	SmileCare Insurance	85
09	Property Insurance	Secure Properties In	70
10	Disability Insurance	AbilitySure Insurance	70

Explanation: The data in insurance table after update.

Query-3: update the price of product to 50 where the existing price of products is less than 50.

SQL Query Before Update: select * from admin.inventory_product order by PRICE;

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query window contains the following code:

```
1 --3. update the price of products to 50 where the existing price is less than 50
2
3 --SQL Query-
4
5 --Before update
6
7 select * from admin.inventory_product order by PRICE;
8
```

The results tab displays the data from the inventory_product table before the update:

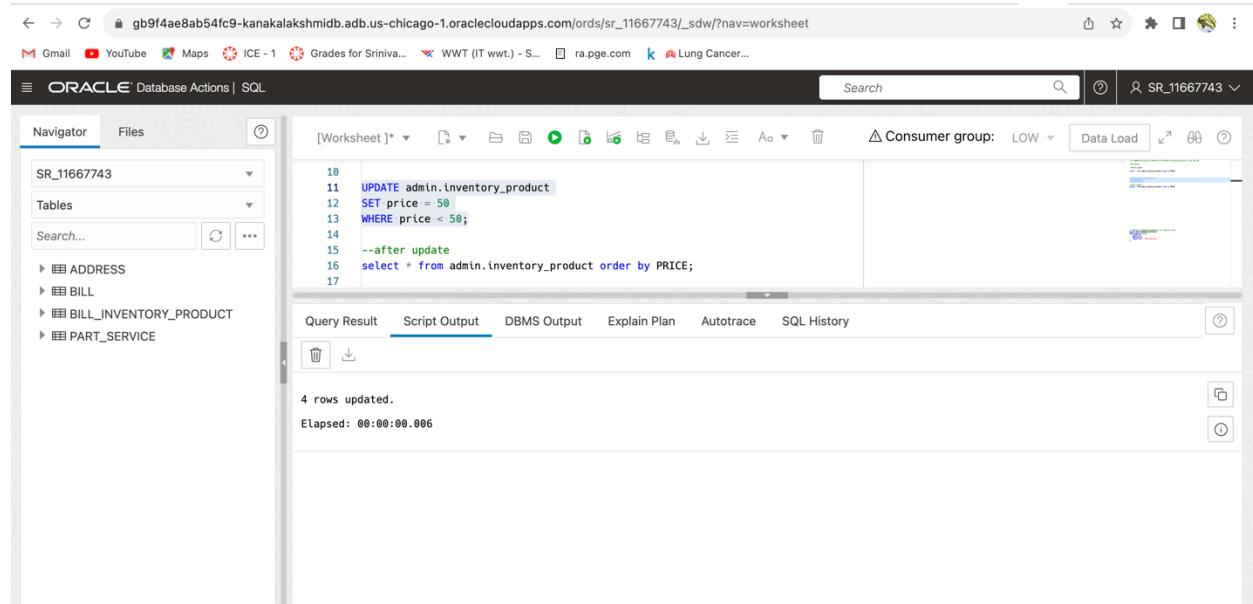
ID	NAME	QUANTITY	PRICE	AUTOMOTIVE_RET	AUTOMOBILE_ID	ADDRESS_ID	
1	0512	Product I	130	35	0302	01	0204
2	0509	Product I	130	35	0308	02	0201
3	0503	Product C	120	40	0303	03	0203
4	0505	Product E	10	45	0305	01	0201
5	0501	Product A	100	50	0301	01	0201
6	0506	Product F	80	55	0306	02	0202
7	0504	Product D	90	60	0304	02	0204
8	0507	Product G	0	65	0307	02	0202

Explanation: The data in inventory product table and price table before updating.

SQL Query to Update:

```
UPDATE admin.inventory_product  
SET price = 50  
WHERE price < 50;
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query window contains the following SQL code:

```
10  
11 UPDATE admin.inventory_product  
12 SET price = 50  
13 WHERE price < 50;  
14  
15 --after update  
16 select * from admin.inventory_product order by PRICE;  
17
```

The "Script Output" tab is selected, showing the results of the query execution:

```
4 rows updated.  
Elapsed: 00:00:00.006
```

Explanation: This SQL query is like instructing a database to go through a table "inventory_product" and find all the products with a price less than 50. For each of those products, it changes their price to 50. In simple terms, it's updating the prices of products that cost less than 50 to make them all cost 50.

SQL Query After update: select * from admin.inventory_product order by PRICE;

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. On the left, the Navigator pane lists tables: ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE. The main area displays a SQL script and its execution results.

Script:

```
10
11 UPDATE admin.inventory_product
12 SET price = 50
13 WHERE price < 50;
14
15 --after update
16 select * from admin.inventory_product order by PRICE;
17
```

Query Result:

ID	NAME	QUANTITY	PRICE	AUTOMOTIVE_RET	AUTOMOBILE_ID	ADDRESS_ID
1	Product A	100	50	0301	01	0201
2	Product I	130	50	0302	01	0204
3	Product C	120	50	0303	03	0203
4	Product E	10	50	0305	01	0201
5	Product I	130	50	0308	02	0201
6	Product F	80	55	0306	02	0202
7	Product D	90	60	0304	02	0204
8	Product G	0	65	0307	00	0202

Explanation: After updating, no product price is less than 50.

Query-4: Update Country for a specific Address Id

SQL Query Before update: select * from SR_11667743.address where COUNTRY='USA';

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables for the schema SR_11667743, which includes ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE tables. The main workspace contains a SQL editor with the following code:

```
1
2
3
4 -- selecting the records before updating.
5 select * from SR_11667743.address where id = '0201';
```

Below the editor is a table titled "Query Result" showing the results of the query:

ID	APARTMENT_NO	STREET	CITY	STATE	COUNTRY	ZIP
1	0201	101 123 Main St	Los Angeles	California	USA	90001

The status bar at the bottom indicates "12:17:00 AM - Code execution finished."

Explanation: This query is to display the rows where the country is listed as 'USA.' It's like asking for a list of addresses specifically in the United States.

SQL Query to Update:

```
UPDATE SR_11667743.address  
SET country = 'United States'  
WHERE id = '0201';
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. On the left is a Navigator pane with a dropdown set to 'SR_11667743' and sections for Tables, Search..., ADDRESS, BILL, BILL_INVENTORY_PRODUCT, and PART_SERVICE. The main area is a 'Worksheet' tab showing the following SQL code:

```
7 -- 4. Update Country for Address Id '1000'  
8 UPDATE SR_11667743.address  
9 SET country = 'United States'  
10 WHERE id = '0201';
```

Below the code, the 'Script Output' tab is selected, displaying the results:

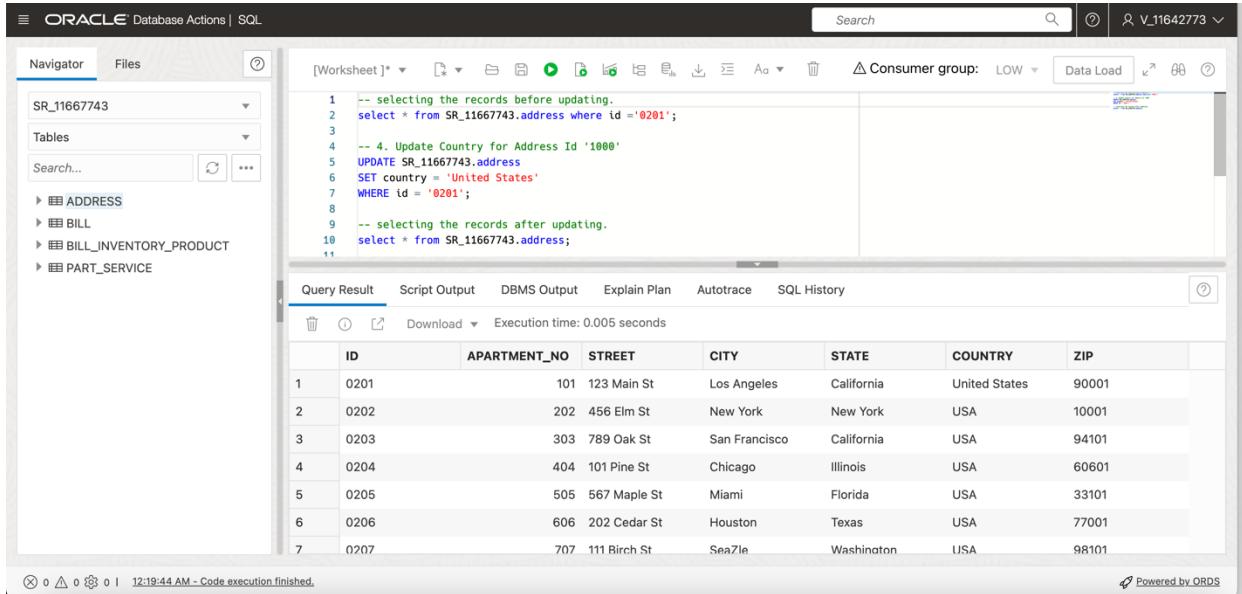
```
1 row updated.  
Elapsed: 00:00:00.006
```

At the bottom, a status bar indicates: '0 ▲ 0 ⚡ 0 | 12:18:32 AM - Code execution finished.' and 'Powered by ORDS'.

Explanation: This SQL query updates a database table called "address" with a specific ID of '0201' and changes the country value to 'United States.' It essentially edits the country information for a particular address entry in the table.

SQL Query After update: select * from SR_11667743.address;

Screenshot:



The screenshot shows the Oracle Database Actions interface with a SQL worksheet. The code executed is:

```
1 -- selecting the records before updating.
2 select * from SR_11667743.address where id ='0201';
3
4 -- 4. Update Country for Address Id '1000'
5 UPDATE SR_11667743.address
6 SET country = 'United States'
7 WHERE id = '0201';
8
9 -- selecting the records after updating.
10 select * from SR_11667743.address;
11
```

The results of the query are displayed in a table:

ID	APARTMENT_NO	STREET	CITY	STATE	COUNTRY	ZIP	
1	0201	101	123 Main St	Los Angeles	California	United States	90001
2	0202	202	456 Elm St	New York	New York	USA	10001
3	0203	303	789 Oak St	San Francisco	California	USA	94101
4	0204	404	101 Pine St	Chicago	Illinois	USA	60601
5	0205	505	567 Maple St	Miami	Florida	USA	33101
6	0206	606	202 Cedar St	Houston	Texas	USA	77001
7	0207	707	111 Birch St	Seattle	Washington	USA	98101

Execution time: 0.005 seconds

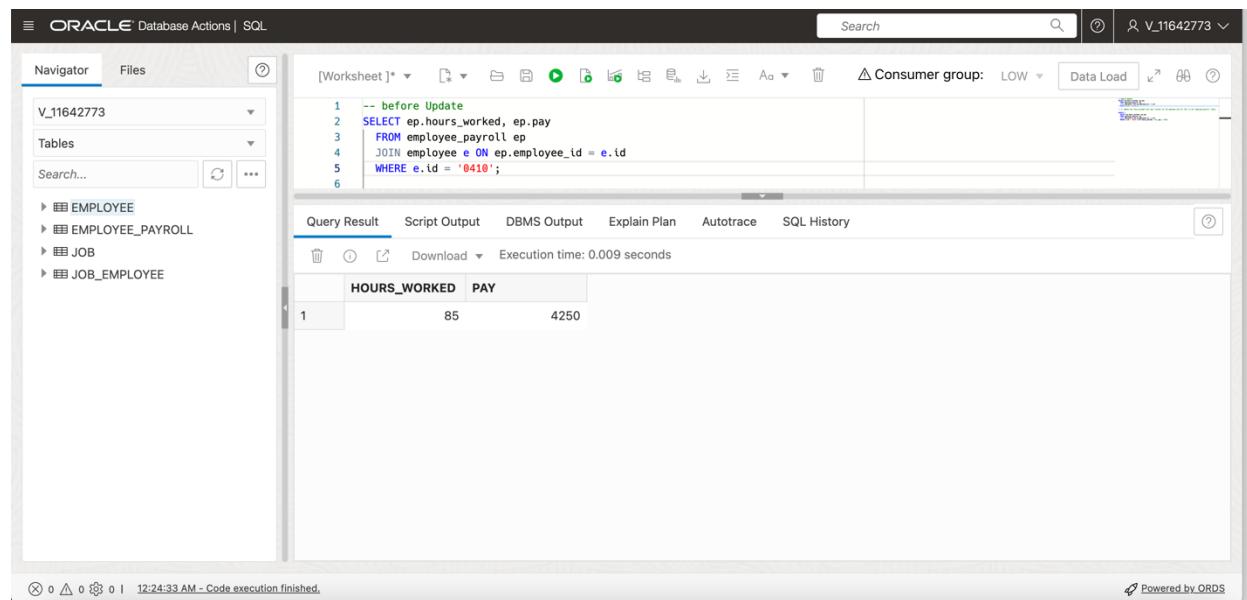
Explanation: The country of id 201 is updated to United States.

Query-5: update the "hours_worked" and "pay" columns for the employee for a specific ID in the "employee_payroll" table.

SQL Query Before update:

```
SELECT ep.hours_worked, ep.pay
FROM employee_payroll ep
JOIN employee e ON ep.employee_id = e.id
WHERE e.id = '0410';
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, with tables like EMPLOYEE, EMPLOYEE_PAYROLL, JOB, and JOB_EMPLOYEE listed. The main workspace contains the following SQL code:

```
-- before Update
SELECT ep.hours_worked, ep.pay
FROM employee_payroll ep
JOIN employee e ON ep.employee_id = e.id
WHERE e.id = '0410';
```

The 'Query Result' tab is selected, showing the output of the query:

	HOURS_WORKED	PAY
1	85	4250

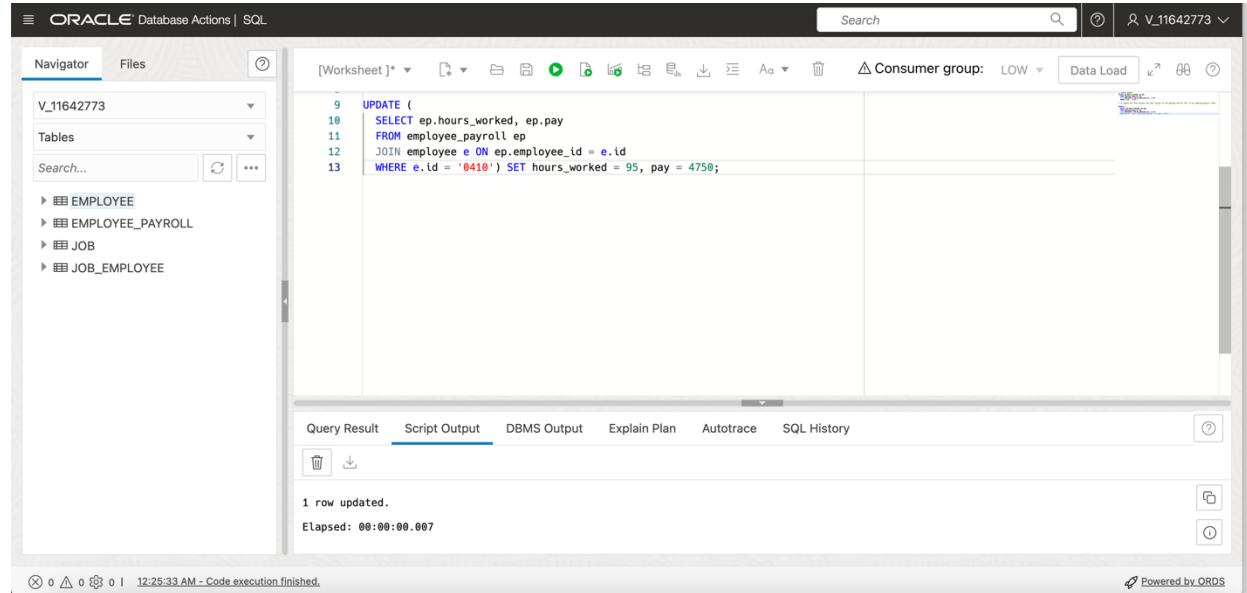
Execution time: 0.009 seconds

Explanation: This SQL query fetches the hours worked and pay details for an employee identified by the ID '0410' from a database that manages employee payroll. It combines data from two tables, 'employee_payroll' and 'employee,' linking them by the employee ID, and then narrows down the results to show information about the specific employee with that ID. This is performed to check the data before updating.

SQL Query to Update:

```
UPDATE(
SELECT ep.hours_worked, ep.pay
FROM employee_payroll ep
JOIN employee e ON ep.employee_id = e.id
WHERE e.id = '0410') SET hours_worked = 95, pay = 4750;
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, listing tables like EMPLOYEE, EMPLOYEE_PAYROLL, JOB, and JOB_EMPLOYEE. The main workspace contains the following SQL code:

```
9 UPDATE (
10   SELECT ep.hours_worked, ep.pay
11   FROM employee_payroll ep
12   JOIN employee e ON ep.employee_id = e.id
13   WHERE e.id = '0410') SET hours_worked = 95, pay = 4750;
```

Below the code, the "Script Output" tab is selected, showing the results of the query execution:

```
1 row updated.
Elapsed: 00:00:00.007
```

At the bottom of the interface, a status bar indicates "Powered by ORDS".

Explanation: The query is meant to change the recorded working hours and pay for an employee with the employee ID '0410' in the database. It finds this employee's data in the "employee_payroll" table, and then updates their "hours_worked" to 95 and "pay" to 4750. Essentially, it's adjusting the payroll information for that specific employee.

SQL Query After update:

```
SELECT ep.hours_worked, ep.pay
FROM employee_payroll ep
JOIN employee e ON ep.employee_id = e.id
WHERE e.id = '0410';
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with tables: EMPLOYEE, EMPLOYEE_PAYROLL, JOB, and JOB_EMPLOYEE. The main area contains two SQL statements. The first statement is an update query:

```
10  SELECT ep.hours_worked, ep.pay
11  FROM employee_payroll ep
12  JOIN employee e ON ep.employee_id = e.id
13  WHERE e.id = '0410' ) SET hours_worked = 95, pay = 4750;
14
15  -- after Update
16  SELECT ep.hours_worked, ep.pay
17  FROM employee_payroll ep
18  JOIN employee e ON ep.employee_id = e.id
19  WHERE e.id = '0410';
```

The second statement is a select query that retrieves the updated data. Below the code, the 'Query Result' tab is selected, showing the output:

	HOURS_WORKED	PAY
1	95	4750

At the bottom of the interface, a status bar indicates "12:26:28 AM - Code execution finished."

Explanation: This is the updated data.

Query-6: SQL query to update the payment_plan_id for an employee with the first name 'Ava' in the bill table.

SQL Query Before update: select * from SR_11667743.bill;

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and a list of objects under database S11664475, including CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The main area contains a code editor with the following SQL query:

```
1 -- SQL query to update the payment_plan_id for an employee with the first name 'Ava' in the bill table.
2
3 select * from SR_11667743.bill;
4
```

Below the code editor is a "Query Result" table with the following data:

	BILL_DATE	MODE_OF_PAYMENT	INSURANCE_ID	CUSTOMER_ID	JOB_ID	EMPLOYEE_ID	SALE_TYPE	PAYMENT_PLAN_ID
1	2023-03-10T09:45:00	CARD	03	0803	0903	0403	ONLINE	1003
2	2023-04-05T12:00:00	CASH	04	0804	0904	0404	OFFLINE	1004
3	2023-06-08T11:45:00	CASH	06	0806	0906	0406	OFFLINE	1006
4	2023-07-23T09:15:00	CARD	07	0807	0907	0407	ONLINE	1007
5	2023-08-19T14:00:00	CASH	08	0808	0908	0408	OFFLINE	1008
6	2023-10-14T15:45:00	CASH	10	0810	0901	0410	OFFLINE	010010
7	2022-12-24T14:00:00	CASH	08	0808	0908	0408	OFFLINE	1001
8	2022-12-24T10:30:00	CARD	06	0807	0907	0407	ONLINE	1002
9	2022-12-24T15:45:00	CASH	10	0810	0910	0410	OFFLINE	1003

At the bottom of the interface, there are status icons and a note: "Powered by ORDS".

Explanation: The information of the “bill” table before updating.

SQL Query to Update:

```
UPDATE SR_11667743.bill  
SET insurance_id='5'  
WHERE employee_id IN(SELECT id FROM V_11642773.employee WHERE first_name = 'Ava');
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the schema 'S116644475' selected, showing tables like CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, PAYMENT_PLAN, and various SYS_C002634x entries. The central workspace contains the following SQL code:

```
-- SQL query to update the insurance for an employee with the first name 'Ava' in the bill table.  
UPDATE SR_11667743.bill  
SET insurance_id = '5'  
WHERE employee_id IN (SELECT id FROM V_11642773.employee WHERE first_name = 'Ava');
```

The 'Script Output' tab at the bottom is selected, showing the results of the query execution:

```
2 rows updated.  
Elapsed: 00:00:00.007
```

At the bottom left, there are status icons for errors, warnings, and information, followed by the message '12:30:21 AM - Code execution finished.' On the right, there is a note 'Powered by ORDS'.

Explanation: This query is trying to change the "insurance_id" for certain entries in the "SR_11667743.bill" table. It's supposed to update the insurance ID to '5' for employees with the first name 'Ava.' However, it has some syntax errors and might not work as intended due to missing operators in the WHERE clause.

Screenshot After update:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the schema S116644475 and various objects like CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, PAYMENT_PLAN, and several SYS objects. The main area has a toolbar with icons for file operations and a search bar. The consumer group is set to LOW. A SQL query is run:

```
-- SQL query to update the insurance for an employee with the first name 'Ava' in the bill table.  
select * from SR_11667743.bill;
```

The Query Result tab is selected, showing the execution time was 0.003 seconds. The results are displayed in a table:

	BILL_DATE	MODE_OF_PAYMENT	INSURANCE_ID	CUSTOMER_ID	JOB_ID	EMPLOYEE_ID	SALE_TYPE	PAYM
1	2023-03-10T09:45:00	CARD	03	0803	0903	0403	ONLINE	1003
2	2023-04-05T12:00:00	CASH	04	0804	0904	0404	OFFLINE	1004
3	2023-06-08T11:45:00	CASH	06	0806	0906	0406	OFFLINE	1006
4	2023-07-23T09:15:00	CARD	07	0807	0907	0407	ONLINE	1007
5	2023-08-19T14:00:00	CASH	08	0808	0908	0408	OFFLINE	1008
6	2023-10-14T15:45:00	CASH	05	0810	0901	0410	OFFLINE	01001
7	2022-12-24T14:00:00	CASH	08	0808	0908	0408	OFFLINE	1001
8	2022-12-24T10:30:00	CARD	06	0807	0907	0407	ONLINE	1002
9	2022-12-24T15:45:00	CASH	05	0810	0910	0410	OFFLINE	1003

At the bottom, it says "Powered by ORDS".

Query-7: update the address of a customer with the first name 'James' in the address

SQL Query Before update:

```
SELECT street, city, state, zip  
FROM SR_11667743.address  
WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and a list of database objects, including CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, PAYMENT_PLAN, and several SYS_C0026411 through C0026413 entries. The main workspace contains the following SQL code:

```
--update the address of a customer with the first name 'James' in the address  
1  
2  
3 SELECT street, city, state, zip  
4 FROM SR_11667743.address  
5 WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');  
6  
7  
8
```

The 'Query Result' tab is selected, showing the output of the query:

	STREET	CITY	STATE	ZIP
1	404 Redwood St	Phoenix	Arizona	85001

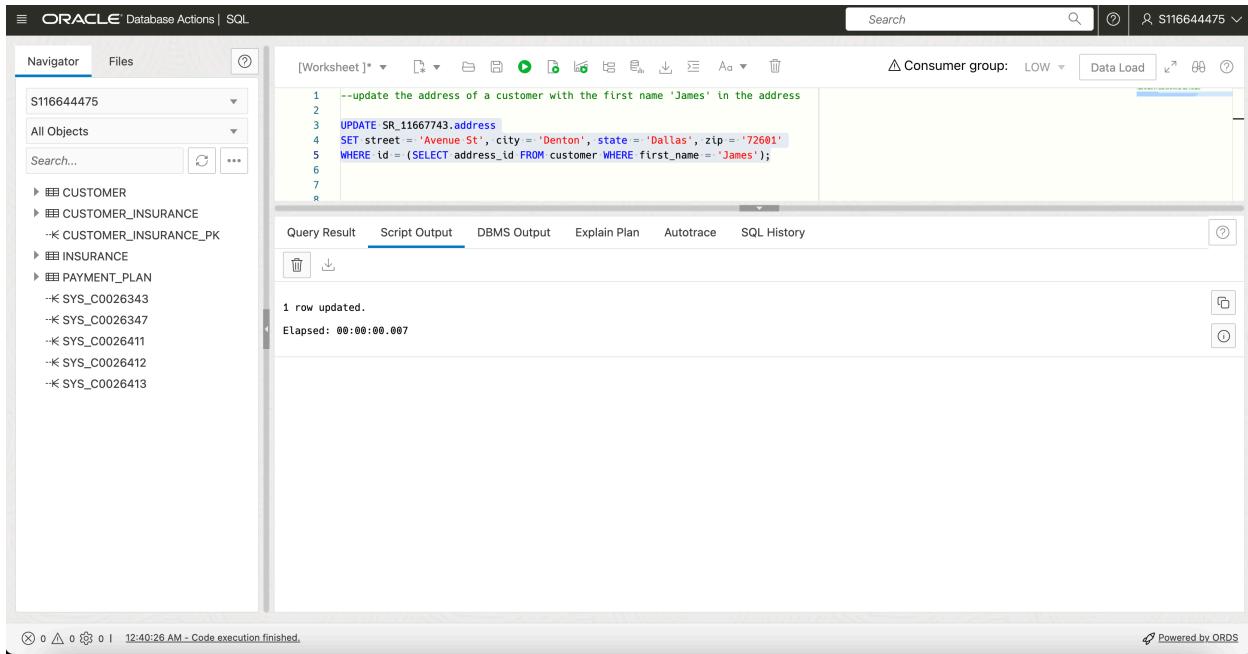
Below the table, the status bar indicates "12:38:02 AM - Code execution finished." and "Powered by ORDS".

Explanation: This SQL query fetches the street, city, state, and zip code of the address for a customer named James. It first finds James's customer ID and then uses that ID to get his address information from the database. This is performed before updation.

SQL Query to Update:

```
UPDATE SR_11667743.address
SET street = 'Avenue St', city = 'Denton', state = 'Dallas', zip = '72601'
WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the schema 'S116644475' selected. The main area contains the SQL script:

```
--update the address of a customer with the first name 'James' in the address
UPDATE SR_11667743.address
SET street = 'Avenue St', city = 'Denton', state = 'Dallas', zip = '72601'
WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');
```

The 'Script Output' tab is selected, showing the results of the query execution:

```
1 row updated.
Elapsed: 00:00:00.007
```

At the bottom left, a status bar indicates '12:40:28 AM - Code execution finished.'

Explanation: This SQL query updates the address information for a customer named James. It changes the street to 'Avenue St', the city to 'Denton', the state to 'Dallas', and the ZIP code to '72601' for James' address.

SQL Query After Update:

```
SELECT street, city, state, zip
FROM SR_11667743.address
WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the schema 'SR116644475' selected, showing tables like CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The central area contains the SQL code:

```
--update the address of a customer with the first name 'James' in the address
SELECT street, city, state, zip
FROM SR_11667743.address
WHERE id = (SELECT address_id FROM customer WHERE first_name = 'James');
```

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

	STREET	CITY	STATE	ZIP
1	Avenue St	Denton	Dallas	72601

Execution time: 0.003 seconds

At the bottom, a status bar indicates: 12:41:44 AM - Code execution finished.

Explanation: The same select statement is performed as before updating to check if the information is updated or not.

6 DELETE QUERIES:

Query-1: Delete All Automotive Retailers in a San Francisco City.

SQL Query Before Delete:

```
select * from AUTOMOTIVE_RETAILER  
where address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, it says "ORACLE Database Actions | SQL". The main area is a "Worksheet" tab. On the left, there's a "Navigator" pane showing database schema like ADMIN, AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The central workspace contains the following SQL code:

```
-- Delete All Automotive Retailers in a San Francisco City  
select * from AUTOMOTIVE_RETAILER  
where address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
```

Below the code, the "Query Result" tab is selected, showing the execution time: "Execution time: 0.136 seconds". The result table has columns: ID, NAME, PHONE, EMAIL, WEBSITE, BUSINESS_HOURS, MANAGER_ID, and ADDRESS_ID. It displays two rows of data:

ID	NAME	PHONE	EMAIL	WEBSITE	BUSINESS_HOURS	MANAGER_ID	ADDRESS_ID
1 0303	autozone retailer3	(345)678-9012	retailer3@example.c	www.retailer3.com	10:00 AM - 5:30 PM	03	0203
2 0310	autozone retailer10	(012)345-6789	retailer10@example.c	www.retailer10.com	9:30 AM - 5:30 PM	10	0210

At the bottom of the interface, there are status icons and a note: "1:11:16 AM - Code execution finished." and "Powered by ORDS".

Explanation: As we are performing delete operation, the information present before deleting is displayed through this query. It first finds the unique identifiers (IDs) of San Francisco addresses in the "address" table and then matches those IDs with the "address_id" field in the "AUTOMOTIVE_RETAILER" table. This helps identify and select all the automotive retailers located in San Francisco.

SQL Query to Delete:

```
DELETE FROM AUTOMOTIVE_RETAILER  
WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
```

Screenshot:

The screenshot shows the Oracle Database Actions interface with a SQL worksheet open. The worksheet contains the following SQL code:

```
1 -- Delete All Automotive Retailers in a San Francisco City  
2  
3 DELETE FROM AUTOMOTIVE_RETAILER  
4 WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
```

Below the code, the error message is displayed:

ORA-02292: integrity constraint (ADMIN.SYS_C0026401) violated - child record found
Error at Line: 7 Column: 0

The status bar at the bottom indicates: 0 0 0 0 1 1:20:48 AM - REST call resolved successfully.

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the central workspace, a SQL script is being run:

```

1 -- Delete All Automotive Retailers in a San Francisco City
2
3 DELETE FROM AUTOMOTIVE_RETAILER
4 WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
    
```

An error message is displayed below the script output:

ORA-02292: integrity constraint (ADMIN.SYS_C0026401) violated - child record found

In the top right corner, the "Table Properties" dialog is open for the "AUTOMOTIVE_RETAILER" table. The "Foreign Keys" tab is selected, showing three foreign key constraints:

Name	Enabled	Rely	Deferrable	Initially Immediate
SYS_C0026401	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
SYS_C0026402	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
SYS_C0026403	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

The "Associations" section shows a constraint named "SYS_C0026364" associated with the "AUTOMOTIVE_RETAILER_ID" column.

Explanation: Unable to delete the data due to integrity constraint violation, updating on Delete cascade to respective tables. *The same process is applied to below, wherever it is necessary.*

The screenshot shows the Oracle Database Actions interface. A SQL script is being run:

```

1 -- Delete All Automotive Retailers in a San Francisco City
2
3 DELETE FROM AUTOMOTIVE_RETAILER
4 WHERE address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
    
```

The output shows:

2 rows deleted.

Elapsed: 00:00:00.027

This indicates that the delete operation was successful because the WHERE clause specified a condition that excluded any rows with a foreign key constraint.

SQL Query After Delete: select * from AUTOMOTIVE_RETAILER
where address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar has 'Navigator' selected, showing tables like AUTOMOTIVE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The main area has a code editor with the following SQL script:

```
-- Delete All Automotive Retailers in a San Francisco City
select * from AUTOMOTIVE_RETAILER
where address_id IN (SELECT id FROM SR_11667743.address WHERE city = 'San Francisco');
```

Below the code editor is a 'Query Result' tab, which displays the following table header and message:

ID	NAME	PHONE	EMAIL	WEBSITE	BUSINESS_HOURS	MANAGER_ID	ADDRESS_ID
----	------	-------	-------	---------	----------------	------------	------------

No rows selected

At the bottom of the interface, it says 'Powered by ORDS'.

Explanation: The provided SQL query is to remove records from a table called "AUTOMOTIVE_RETAILER." The deletion is based on a condition that specifies removing rows where the "address_id" matches any ID found in a subquery result. The subquery retrieves IDs from the "address" table, but only those associated with the city 'San Francisco.' It specifically deletes entries that have an "address_id" linked to addresses in San Francisco. In simpler terms, it deletes information about automotive retailers in San Francisco from the database

Query-2: Deleting customers based on age less than zero can be done using a DELETE statement combined with a subquery to identify and remove the relevant customer records.

Select Query Before Delete: select * FROM customer
WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Files sections, with 'S116644475' selected. The main workspace contains a SQL script and its execution results.

Script Content:

```
1 --Deleting customers based on age less than zero can be done using a DELETE statement combined with a subquery to identify
2 -- and remove the relevant customer records.
3
4 select * FROM customer
5 WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;
6
7
8
```

Execution Results:

Query Result

ID	FIRST_NAME	LAST_NAME	DOB	DRIVERLICENSE	PHONE	ADDRESS_ID
1	0865	KIRAN	POLLARD	2024-03-15T00:00:00+01:00	(123)456-7880	0201

Execution time: 0.006 seconds

Powered by ORDS

SQL Query to Delete: DELETE FROM S116644475.customer WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;

Screenshot:

The screenshot shows the Oracle Database Actions interface with a SQL worksheet. The Navigator pane on the left lists schema objects for 'S116644475'. The Worksheet pane contains the following SQL code:

```
1 --Deleting customers based on age less than zero can be done using a DELETE statement combined with a subquery to identify
2 -- and remove the relevant customer records.
3
4 DELETE FROM S116644475.customer
5 WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;
6
7
8
```

The Script Output tab is selected, showing the results of the execution:

1 row updated.
Elapsed: 00:00:00.007

At the bottom, status indicators show 0 errors, 0 warnings, 0 informational messages, and 1 log entry, with the message "1:29:45 AM - Code execution finished." A "Powered by ORDS" link is also present.

Explanation: The SQL query identifies and deletes customer records with an age less than zero. It does this by calculating the age of each customer using their date of birth and the current date. If it finds any customers whose calculated age is less than zero, it means there's a data problem, and these erroneous customer records will be removed from the database. This query helps maintain data accuracy by eliminating entries with impossible negative ages.

SQL Query After Delete:

```
select * FROM customer  
WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator with the schema S11664475 and its objects: CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The central workspace contains the following SQL code:

```
--Deleting customers based on age less than zero can be done using a DELETE statement combined with a subquery to identify  
-- and remove the relevant customer records.  
select * FROM customer  
WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob) < 0;
```

The bottom pane shows the Query Result with the following output:

ID	FIRST_NAME	LAST_NAME	DOB	DRIVERLICENSE	PHONE	ADDRESS_ID
No rows selected						

Execution time: 0.004 seconds

At the bottom of the interface, it says "Powered by ORDS".

Explanation: This query fetches data from the "customer" table for individuals who, according to the current year, have not been born yet. It's essentially trying to find customers with birthdates in the future, based on the system's date. This is just to test whether the data got deleted or not.

Query-3: Delete All Parts or Services associated with Tire Rotation description.

SQL Query Before Delete:

```
select *
FROM part_service ps
JOIN V_11642773.job j ON ps.job_id = j.id
WHERE j.Description = 'Tire Rotation';
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The query window contains the following SQL code:

```
--3.Delete All Parts or Services associated with Tire Rotation description:
select *
FROM part_service ps
JOIN V_11642773.job j ON ps.job_id = j.id
WHERE j.Description = 'Tire Rotation';
```

The results tab displays a single row of data from the PART_SERVICE table:

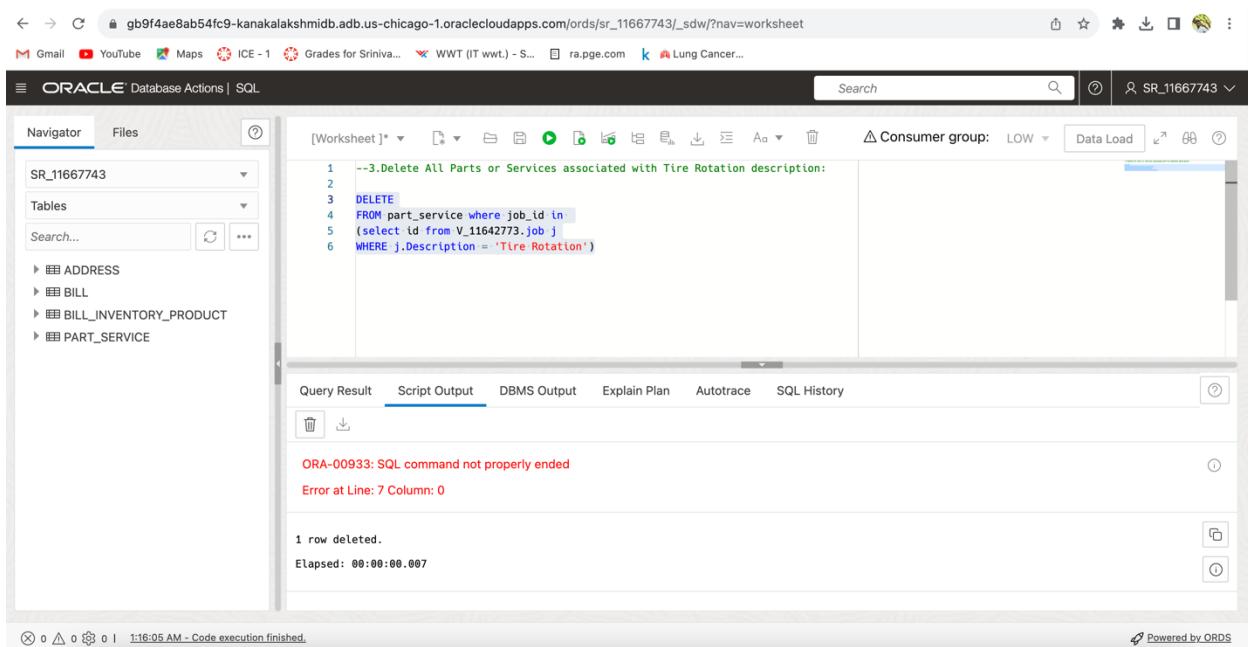
ID	NAME	QUANTITY	JOB_ID	INVENTORY_PROD	TYPE	ID	JOB_D	
1	Tire Rotation		4	0903	0503	SERVICE	0903	2023-0

Explanation: This query identifies all the parts and services that are connected to the job with the description "Tire Rotation." It retrieves all the details about these parts and services. This is the data before the delete is performed.

SQL Query to Delete:

```
DELETE
FROM part_service where job_id in
(select id from V_11642773.job j
WHERE j.Description = 'Tire Rotation')
```

Screenshot:



The screenshot shows the Oracle Database Actions | SQL interface. In the central workspace, a SQL script is displayed:

```
1 --3_Delete All Parts or Services associated with Tire Rotation description:
2
3 DELETE
4 FROM part_service where job_id in
5 (select id from V_11642773.job j
6 WHERE j.Description = 'Tire Rotation')
```

The 'Script Output' tab is selected, showing the following results:

ORA-00933: SQL command not properly ended
Error at Line: 7 Column: 0

1 row deleted.
Elapsed: 00:00:00.007

At the bottom left, there are status icons: a circle with an X, a triangle, a gear, and a hexagon, each with a count of 0. At the bottom right, it says "Powered by ORDS".

Explanation: This query is designed to delete data from a table called "part_service." It targets records related to a specific job, which has the description "Tire Rotation." So, any services associated with this particular job will be removed from the "part_service" table.

Screenshot After Delete:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The top navigation bar includes links to Gmail, YouTube, Maps, ICE - 1, Grades for Sriniva..., WWT (IT wwt.) - S..., ra.pge.com, and Lung Cancer... The search bar contains 'SR_11667743'. The main workspace displays a SQL query:

```
1 --3.Delete All Parts or Services associated with Tire Rotation description:
2
3 select *
4 FROM part_service where job_id in
5 (select id from V_11642773.job j
6 WHERE j.Description = 'Tire Rotation')
```

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

ID	NAME	QUANTITY	JOB_ID	INVENTORY_PROD	TYPE
No rows selected					

The execution time is listed as 0.004 seconds.

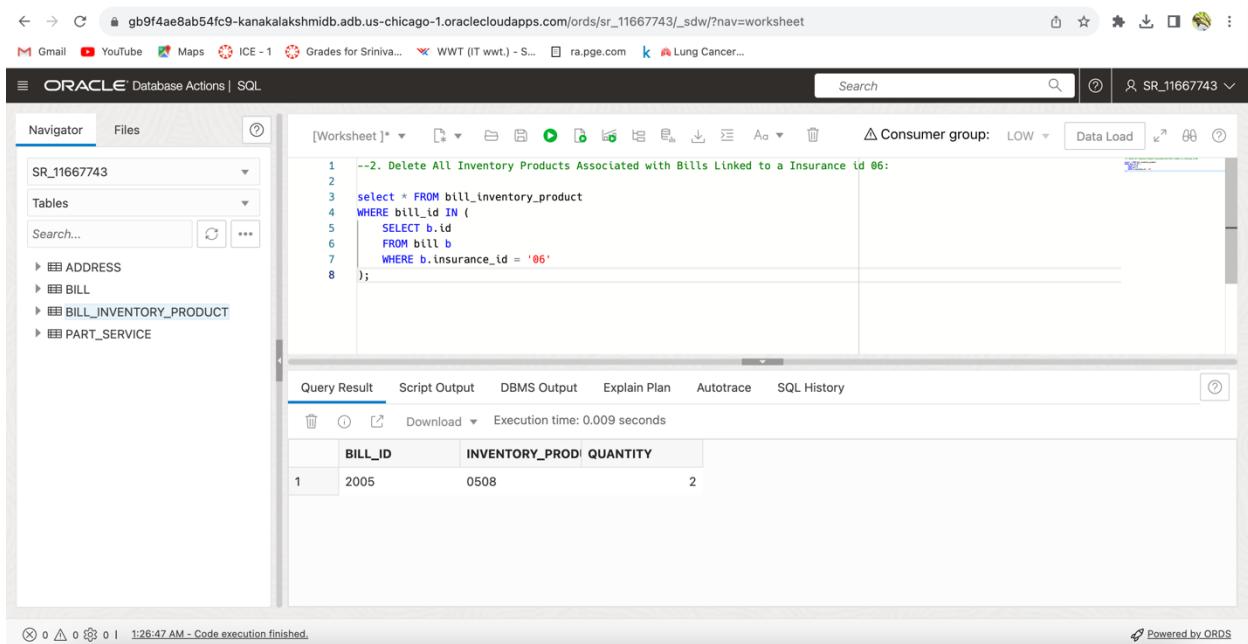
Explanation: This is the data after deleting.

Query-4: Delete All Inventory Products Associated with Bills Linked to a specific Insurance id.

Select Query Before Delete:

```
select * FROM bill_inventory_product  
WHERE bill_id IN (SELECT b.id FROM bill b WHERE b.insurance_id = '06');
```

Screenshot:



The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw/?nav=worksheet. Below the URL, there are several browser tabs and links. The main workspace is titled [Worksheet] and contains the following SQL code:

```
1 --2. Delete All Inventory Products Associated with Bills Linked to a Insurance id 06:  
2  
3 select * FROM bill_inventory_product  
4 WHERE bill_id IN (  
5     SELECT b.id  
6     FROM bill b  
7     WHERE b.insurance_id = '06'  
8 );
```

Below the code, the "Query Result" tab is selected, showing the following output:

	BILL_ID	INVENTORY_PROD	QUANTITY
1	2005	0508	2

Execution time: 0.009 seconds

At the bottom left, there are status icons: 0 errors, 0 warnings, 0 info, and 0 help. At the bottom right, it says "Powered by ORDS".

Explanation: This query fetches information about inventory products that are associated with bills having an insurance ID of '06'. It looks for these connections in the "bill_inventory_product" table and retrieves the relevant data. This is performed before performing delete on the table.

SQL Query to Delete:

Delete FROM bill_inventory_product

WHERE bill_id IN (SELECT b.id FROM bill b WHERE b.insurance_id = '06');

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, with 'BILL_INVENTORY_PRODUCT' selected. The main workspace contains the following SQL code:

```
--2. Delete All Inventory Products Associated with Bills Linked to a Insurance id 06:  
delete FROM bill_inventory_product  
WHERE bill_id IN (  
... SELECT b.id  
... FROM bill b  
... WHERE b.insurance_id = '06'  
);
```

The 'Script Output' tab is active, showing the results of the execution:

```
1 row deleted.  
Elapsed: 00:00:00.007
```

Below this, another execution shows:

```
1 row deleted.  
Elapsed: 00:00:00.007
```

Explanation: It deletes data from the "bill_inventory_product" table. The deletion is based on a condition: it removes rows where the "bill_id" matches the result of a subquery. The subquery finds "bill" records with the condition that their "insurance_id" is equal to '06'. So, it deletes records in "bill_inventory_product" associated with bills having an "insurance_id" of '06'.

Screenshot After Delete:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, the URL is `gb9f4ae8ab54fc9-kanakalakshmidb.adb.us-chicago-1.oraclecloudapps.com/ords/sr_11667743/_sdw?nav=worksheet`. The main area displays a SQL worksheet with the following code:

```
--2. Delete All Inventory Products Associated with Bills Linked to a Insurance id 06:  
select * FROM bill_inventory_product  
WHERE bill_id IN (  
...  
SELECT b.id  
...  
FROM bill b  
...  
WHERE b.insurance_id = '06'  
);
```

The "Query Result" tab is selected, showing the output:

BILL_ID	INVENTORY_PROD	QUANTITY
No rows selected		

Below the table, it says "Execution time: 0.004 seconds". At the bottom left, there is a message: "1:28:02 AM - Code execution finished." and at the bottom right, "Powered by ORDS".

Explanation: Data after performing delete.

Query-5: Delete All Jobs for Bills with an online Sale Type and description like Electrical Troubleshooting

SQL Query Before Delete:

```
select * FROM job
WHERE id IN (
    SELECT j.id
    FROM job j
    JOIN SR_11667743.bill b ON j.id = b.job_id
    WHERE b.sale_type = 'ONLINE' and j.description= 'Electrical Troubleshooting');
```

Screenshot:

The screenshot shows the Oracle Database Actions interface. In the top navigation bar, it says "ORACLE Database Actions | SQL". The main area is titled "[Worksheet]". The code entered is:

```
1 --4.Delete All Jobs for Bills with a online Sale Type and description like Electrical Troubleshooting
2
3 select * FROM job
4 WHERE id IN (
5     SELECT j.id
6     FROM job j
7     JOIN SR_11667743.bill b ON j.id = b.job_id
8     WHERE b.sale_type = 'ONLINE' and j.description= 'Electrical Troubleshooting');
9
```

Below the code, there is a "Query Result" tab. The output shows a single row of data:

ID	JOB_DATE	DESCRIPTION	CUSTOMER_ID	AUTOMOTIVE_RET	VIN_NUMBER	AUTOMOBILE_ID
1	0907	2023-04-01T13:45:00Z Electrical Troubleshooting	0807	0307	2HKRM38575H9876	07

At the bottom left, it says "Execution time: 0.015 seconds". At the bottom right, it says "Powered by ORDS".

Explanation: This query retrieves all the job records connected to bills with an online sale type and a description containing the term 'Electrical Troubleshooting.' It's a way to find and identify specific jobs related to online sales with electrical troubleshooting needs.

SQL Query to Delete:

```
DELETE FROM job
WHERE id IN (
    SELECT j.id
    FROM job j
    JOIN SR_11667743.bill b ON j.id = b.job_id
    WHERE b.sale_type = 'ONLINE' and j.description= 'Electrical Troubleshooting');
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays a Navigator with the session ID S116644475 and a list of tables: CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The main workspace contains a SQL script:

```
1 --4.Delete All Jobs for Bills with a online Sale Type and description like Electrical Troubleshooting
2
3 DELETE FROM job
4 WHERE id IN (
5   SELECT j.id
6   FROM job j
7   JOIN SR_11667743.bill b ON j.id = b.job_id
8   WHERE b.sale_type = 'ONLINE' and j.description= 'Electrical Troubleshooting');
9
```

The Query Result tab shows the output: "1 row deleted." and "Elapsed: 00:00:00.016". The status bar at the bottom indicates "1:31:18 AM - Code execution finished."

Explanation: This query will delete rows from the "job" table if they meet the conditions specified in the subquery. It specifically targets jobs with a description of 'Electrical Troubleshooting' that are associated with bills of the 'ONLINE' sale type. It joins the "job" table with another table named "bill" (aliased as "b") on the "job_id" column. It includes only rows where the "sale_type" in the "bill" table is 'ONLINE' and the "description" in the "job" table is 'Electrical Troubleshooting.'

Screenshot After Delete:

The screenshot shows the Oracle Database Actions SQL Worksheet interface after the deletion. The left sidebar is identical to the previous screenshot. The main workspace contains the same SQL script. The Query Result tab shows the output: "Execution time: 0.003 seconds" and a table with columns: ID, JOB_DATE, DESCRIPTION, CUSTOMER_ID, AUTOMOTIVE_RET, VIN_NUMBER, and AUTOMOBILE_ID. The table has a single row labeled "No rows selected". The status bar at the bottom indicates "1:32:22 AM - Code execution finished."

Query-6: Delete Suppliers Based on San Francisco as there is no retailers at San Francisco.

Select Query Before Delete:

```
select *
FROM ADMIN.supplier s where s.ADDRESS_ID in
( select a.id from SR_11667743.address a WHERE a.city = 'San Francisco');
```

Screenshot:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The top navigation bar includes 'Database Actions | SQL', 'Search', and a consumer group dropdown set to 'LOW'. The left sidebar has 'Navigator' selected, showing a database named 'S116644475' and tables: CUSTOMER, CUSTOMER_INSURANCE, INSURANCE, and PAYMENT_PLAN. The main workspace contains the following SQL code:

```
--5 Delete Suppliers Based on San Francisco since we removed retailer there :
select *
FROM ADMIN.supplier s where s.ADDRESS_ID in
( select a.id from SR_11667743.address a WHERE a.city = 'San Francisco');
```

The 'Query Result' tab is selected, displaying the following table:

	ID	NAME	PHONE	ADDRESS_ID
1	03	Supplier C	(345)678-9012	0203
2	10	Supplier J	(012)345-6789	0210

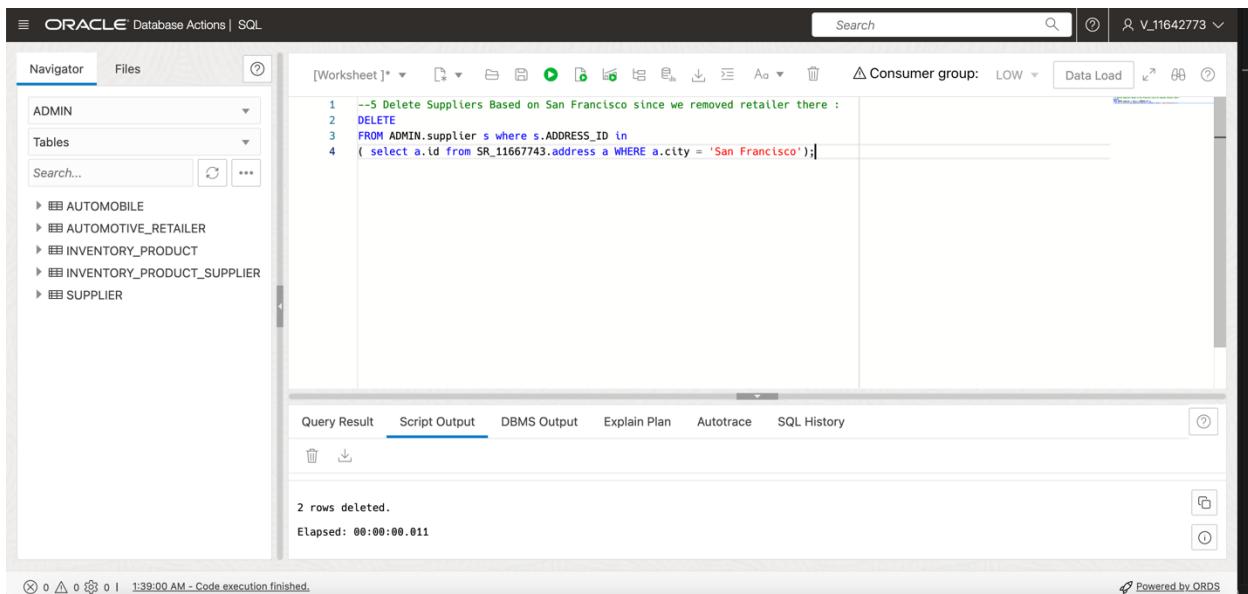
Execution time: 0.003 seconds

Explanation: This query fetches information about suppliers with addresses in San Francisco but does not perform any deletion action. The purpose of this query is to identify these suppliers, in preparation for a later deletion process.

SQL Query to Delete:

```
DELETE
FROM ADMIN.supplier s where s.ADDRESS_ID in
( select a.id from SR_11667743.address a WHERE a.city = 'San Francisco');
```

Screenshot:



The screenshot shows the Oracle Database Actions SQL Worksheet interface. The left sidebar displays the Navigator and Tables sections, with 'ADMIN' selected. The main workspace contains the following SQL code:

```
--5 Delete Suppliers Based on San Francisco since we removed retailer there :
DELETE
FROM ADMIN.supplier s where s.ADDRESS_ID in
( select a.id from SR_11667743.address a WHERE a.city = 'San Francisco');
```

The 'Script Output' tab is selected at the bottom, showing the results of the query execution:

```
2 rows deleted.
Elapsed: 00:00:00.011
```

The status bar at the bottom indicates '1:39:00 AM - Code execution finished.'

Explanation: This SQL query is designed to delete some records in the "supplier" table in the "ADMIN" schema. It removes records that have an "ADDRESS_ID" matching specific conditions. In particular, it deletes supplier records associated with addresses located in 'San Francisco.' This action helps to manage the supplier data in the database by removing entries tied to San Francisco addresses.

Screenshot After Delete:

The screenshot shows the Oracle Database Actions SQL Worksheet interface. The top navigation bar includes 'ORACLE Database Actions | SQL', a search bar, and a session identifier 'V_11642773'. The main area has tabs for 'Navigator' (selected), 'Files', and a toolbar with various icons. The 'Tables' section under 'ADMIN' lists several tables: AUTOMOBILE, AUTOMOTIVE_RETAILER, INVENTORY_PRODUCT, INVENTORY_PRODUCT_SUPPLIER, and SUPPLIER. The central workspace contains a SQL script:

```
1 --5 Delete Suppliers Based on San Francisco since we removed retailer there :
2 select * from
3 FROM ADMIN.supplier s where s.ADDRESS_ID in
4 ( select a.id from SR_11667743.address a WHERE a.city = 'San Francisco');
```

The 'Query Result' tab is selected, showing the output of the query:

ID	NAME	PHONE	ADDRESS_ID
No rows selected			

Below the table, a message indicates the execution time: 'Execution time: 0.004 seconds'. The bottom status bar shows 'Powered by ORDS'.