**Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.**

**Ans:**

To create an ER diagram for given business scenario that includes entities, relationships, attributes, and cardinality, we will follow the following steps .

**Step 1:** Identify Entities and Their Attributes

Entities:

**Books**

BookID (Primary Key)

Title

ISBN

PublicationYear

**Authors**

AuthorID (Primary Key)

AuthorFirstname

AuthorLastname

**Patron**

PatronID (Primary Key)

PatronName

PatronAddress

PatronPhone

**Notification**

NotificationID (Primary Key)

PatronID (Foreign Key)

Message

DateSent

**Inventory**

InventoryID (Primary Key)

BookID (Foreign Key)

Quantity

**Checkout**

CheckoutID (Primary Key)

PatronID (Foreign Key)

BookID (Foreign Key)

CheckoutDate

ReturnDate

**Step 2:** Define Relationships and Cardinality

**Book to Author:** A book can have multiple authors, and an author can write multiple books. (Many-to-Many)

**Book to Publisher:** A book is published by one publisher, and a publisher can publish multiple books. (1-to-Many)

**Book to Inventory:** A book can be listed in multiple inventories. (1-to-Many)

**Patron to Notification:** A patron can receive multiple notifications. (1-to-Many)

**Patron to Checkout:** A patron can check out multiple books, and each checkout is linked to one patron. (1-to-Many)

**Book to Checkout:** A book can be checked out multiple times. (1-to-Many)

**Step 3:** Normalize the Data (up to 3NF)

**First Normal Form (1NF):**

Ensure each column has atomic values and each record is unique.

**Second Normal Form (2NF****):**Ensure that all non-key attributes are fully functional dependent on the primary key.

**Third Normal Form (3NF):** Ensure that all the attributes are not only fully functional dependent on the primary key but also non-transitively dependent.

**Step 4:** Draw the ER Diagram

Here's the detailed ER diagram:



Books

Authors

Inventory

Patron

Notification

Checkout

**Detailed Explanation of the Diagram**

**Entities and Attributes:**

Book: Identified by BookID, with attributes Title, ISBN, PublisherID (as a foreign key), and PublicationYear.

Author: Identified by AuthorID, with the attribute AuthorName.

Publisher: Identified by PublisherID, with attributes PublisherName and PublisherAddress.

Patron: Identified by PatronID, with attributes PatronName, PatronAddress, and PatronPhone.

Notification: Identified by NotificationID, with attributes PatronID (as a foreign key), Message, and DateSent.

Inventory: Identified by InventoryID, with attributes BookID (as a foreign key) and Quantity.

Checkout: Identified by CheckoutID, with attributes PatronID (as a foreign key), BookID (as a foreign key), CheckoutDate, and ReturnDate.

Book\_Author: Composite entity for the many-to-many relationship between Book and Author.

**Relationships:**

Book to Author: A many-to-many relationship implemented via the Book\_Author composite entity.

Book to Publisher: A Book is published by one Publisher (1:N).

Book to Inventory: A Book can be in multiple Inventories (1:N).

Patron to Notification: A Patron can receive multiple Notifications (1:N).

Patron to Checkout: A Patron can check out multiple Books, each Checkout is linked to one Patron (1:N).

Book to Checkout: A Book can be checked out multiple times (1:N).

**Review and Finalization**

The final ER diagram meets the requirements of 3NF, ensuring no redundancy and that all attributes are fully and non-transitively dependent on their respective primary keys. This diagram should be reviewed with business users to confirm that it accurately represents the library system and meets all their requirements.

**Assignment 2: Design a database schema for a library system, including tables, fields,and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.**

**Ans:**

**Database schema for a Library System:**

Books

Book\_Id: number Primary Key

Title: varchar Not Null

Isbn: number(10) Not null

Publication\_Year: number

Authors

Author\_Id: number Primary Key

First\_Name: varchar(20)

Not Null

Last\_Name: varchar(20)

Not Null

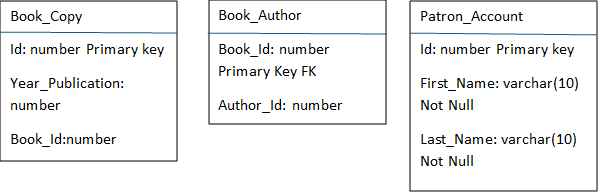
Inventory

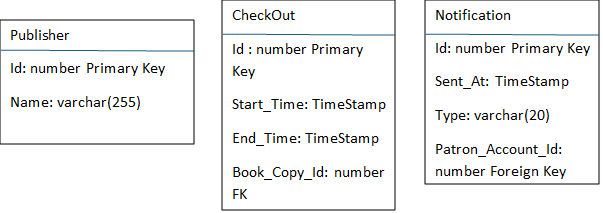
Inventory\_Id: number Primary Key

Book\_Id: number Foriegn Key references Books(Book\_Id);

Forrmat: Check(Format in

('E-book','AudioBook'))





Books:

book\_id number PRIMARY KEY,

title VARCHAR(255) NOT NULL,

isbn VARCHAR(13) UNIQUE NOT NULL,

publication\_year YEAR NOT NULL,

Authors:

author\_id number PRIMARY KEY ,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

Book\_Author:

book\_id number NOT NULL,

author\_id number NOT NULL,

PRIMARY KEY (book\_id, author\_id),

FOREIGN KEY (book\_id) REFERENCES Books(book\_id),

FOREIGN KEY (author\_id) REFERENCES Authors(author\_id),

Patrons:

patron\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

Notifications:

notification\_id INT PRIMARY KEY,

date\_sent DATETIME NOT NULL’

Type VARCHAR (20),

Publishers:

publisher\_id number PRIMARY KEY,

name VARCHAR(255) NOT NULL,

Inventory:

inventory\_id INT PRIMARY KEY AUTO\_INCREMENT

book\_id INT NOT NULL,

format Check(format in (‘E-book’, Audiobook’),

FOREIGN KEY (book\_id) REFERENCES Books(book\_id).

**Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.**

Ans:

ACID properties are a set of guarantees that ensure the integrity and consistency of data during transactions in a database system. Here's a breakdown of each property in simpler terms:

1. **Atomicity:** All the operations within a transaction are treated as a single unit. Either all the operations succeed, or none of them do. This prevents partial updates and ensures data consistency. Imagine transferring money between two accounts. ACID guarantees that either both accounts are updated successfully, or neither is affected.
2. **Consistency:** A transaction transforms the database from one valid state to another. It enforces predefined business rules to maintain data integrity. Think of it like a recipe. Each step (operation) needs to be completed for the final dish (data state) to be correct.
3. **Isolation:** Concurrent transactions executing at the same time are isolated from each other. This prevents conflicts and ensures data consistency. Imagine two people editing the same document simultaneously. Isolation ensures their changes don't interfere with each other.
4. **Durability:** Once a transaction is committed, the changes are permanently stored and survive system failures. This guarantees that data is not lost even in case of crashes or power outages. It's like baking a cake. Once it's taken out of the oven (committed), it's a finished product that won't disappear.

**Simulating a Transaction with Locking(Oracle)**

Here's an example transaction to transfer funds between two accounts (simplified for demonstration):

**Table:** Accounts (account\_id, balance)

**Transaction:** Transfer $500 from account\_id 1 to account\_id 2 SQL

START TRANSACTION;

SELECT \* FROM Accounts WHERE account\_id IN (1, 2) FOR UPDATE;

UPDATE Accounts SET balance = balance - 500 WHERE account\_id = 1;

UPDATE Accounts SET balance = balance + 500 WHERE account\_id = 2;

COMMIT;

This transaction uses FOR UPDATE clause during the SELECT statement to acquire exclusive locks on both accounts. This prevents other transactions from modifying them until the current transaction commits.

**Demonstrating Isolation Levels(Oracle)**

Oracle provides various isolation levels that control how transactions see changes made by other concurrent transactions. Here are two common examples:

1. **Read Committed (Default):** Transactions can only see changes committed by other transactions before they started reading. Uncommitted changes are invisible.

**Scenario:**

Transaction 1 reads the balance of account 1. Transaction 2 debits account 1 but hasn't committed yet. Transaction 1 will see the original balance of account 1, unaware of the pending debit.

1. **Serializable:** Transactions are serialized, meaning they are executed one after another, even if submitted concurrently. This ensures the highest level of consistency but can impact performance.

**Scenario:**

Transaction 1 and Transaction 2 both try to update the same account. Oracle will execute them sequentially, ensuring no conflicts arise.

By adjusting the isolation level, you can achieve a balance between data consistency and concurrency based on your application's needs.

**Assignment 4: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.**

**Ans:**

**CREATE a new database:**

**Query:**

Create database LibraryDB;

Use LibraryDB;

**Create Book Table:**

CREATE TABLE Books (

book\_id number PRIMARY KEY,

title VARCHAR(255) NOT NULL,

isbn VARCHAR(13) NOT NULL UNIQUE,

publication\_year number NOT NULL,

FOREIGN KEY (publisher\_id) REFERENCES Publisher(publisher\_id)

);

**Create Authors table:**

CREATE TABLE Authors (

author\_id number PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL

);

**Create Publisher table:**

CREATE TABLE Publisher (

publisher\_id number PRIMARY KEY,

name VARCHAR(100) NOT NULL

);

**Create Inventory table:**

CREATE TABLE Inventory (

inventory\_id number PRIMARY KEY,

book\_id number NOT NULL,

FOREIGN KEY (book\_id) REFERENCES Books(book\_id)

);

**Create Book\_Author table:**

CREATE TABLE Book\_Author (

book\_id number NOT NULL,

author\_id number NOT NULL,

PRIMARY KEY (book\_id, author\_id),

FOREIGN KEY (book\_id) REFERENCES Books(book\_id),

FOREIGN KEY (author\_id) REFERENCES Authors(author\_id));

**Create Patron\_Account table:**

CREATE TABLE Patron\_Account (

patron\_id number PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE,

phone VARCHAR(15) NOT NULL,

address VARCHAR(255) NOT NULL

);

**Create Checkout table:**

CREATE TABLE Checkout (

checkout\_id number PRIMARY KEY,

patron\_id number NOT NULL,

book\_id number NOT NULL,

checkout\_date DATE NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE,

FOREIGN KEY (patron\_id) REFERENCES Patron\_Account(patron\_id),

FOREIGN KEY (book\_id) REFERENCES Books(book\_id)

);

**Create Notification table:**

CREATE TABLE Notification (

notification\_id number PRIMARY KEY,

patron\_id number NOT NULL,

sent\_date DATE NOT NULL,

FOREIGN KEY (patron\_id) REFERENCES Patron\_Account(patron\_id));

**Add a middle\_name column to PatronAccount table:**

ALTER TABLE Patron\_Account ADD COLUMN middle\_name VARCHAR(50);

**Add a genre column to Books table:**

ALTER TABLE Books ADD COLUMN genre VARCHAR(50);

**Create a redundant Old\_Inventory table (for demonstration purposes):**

CREATE TABLE Old\_Inventory (

old\_inventory\_id number PRIMARY KEY,

book\_id number NOT NULL,

location VARCHAR(100) NOT NULL,

book\_id REFERENCES Books(book\_id)

);

**Drop the redundant Old\_Inventory table:**

DROP TABLE Old\_Inventory;

**Assignment 5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.**

**Ans:**

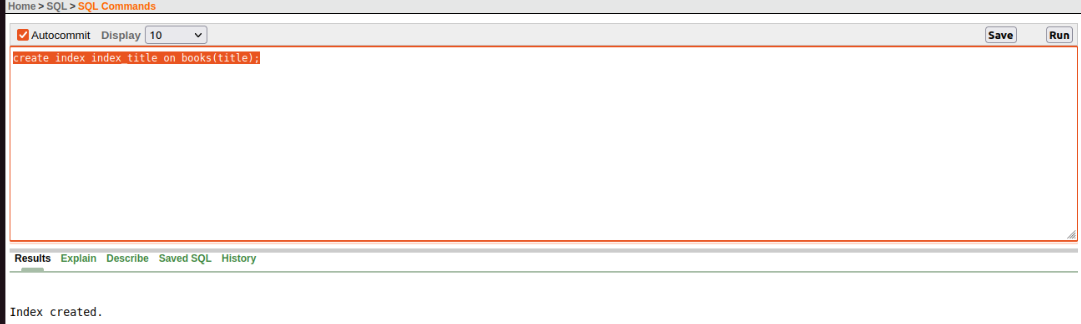
Consider a Books table with columns like book\_id, title, author\_id, and publication\_year. We frequently perform queries that filter books based on their title.

**1. Create an Index:**

An index acts like a pre-sorted catalog for specific columns, speeding up searches. Here's how to create an index named index\_title on the title column of the Books table:

**Query:**

CREATE INDEX index\_title on Books(title);



**2. Analyze Performance Impact:**

Now, let's compare the execution time of two queries:

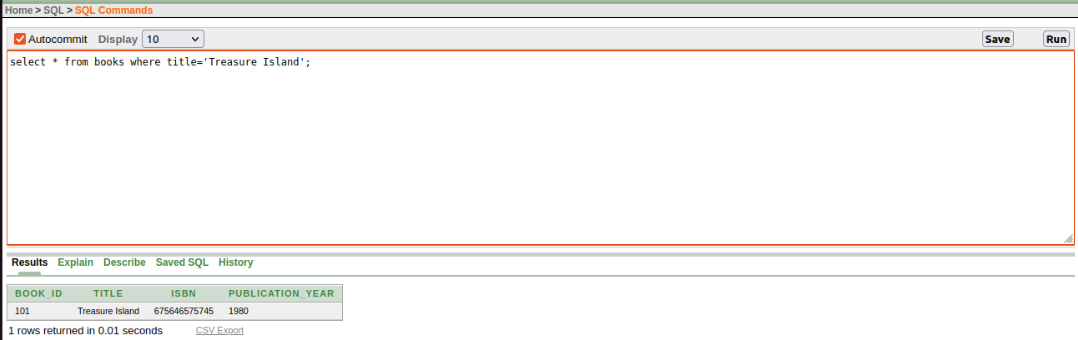
**Query 1 (Without Index):** Selects all books with a specific title.

**Query:**

SELECT \* FROM Books WHERE title = 'Treasure Island';

**Query 2 (With Index):** Same as Query 1, but after creating the index.

Run both queries and compare their execution times. You should see a significant improvement in speed for Query 2 when using the index.



**Explanation:**

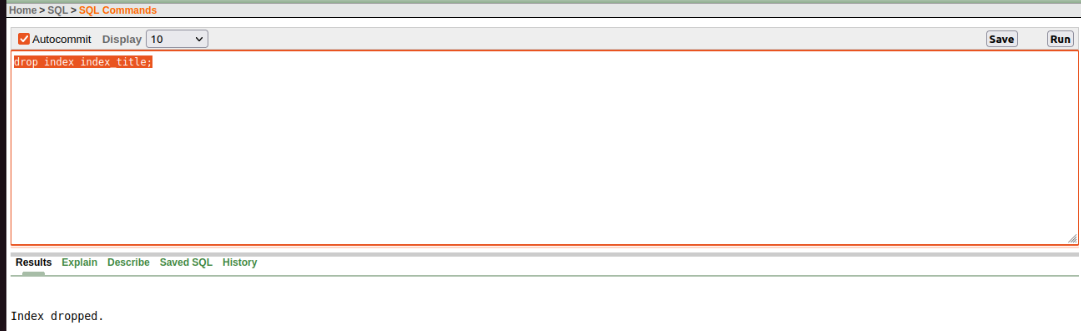
Without an index, Oracle needs to scan every row in the Books table to find books with the specified title. With the index, Oracle can efficiently locate the relevant entries using the pre-sorted structure of the index, significantly reducing the number of rows to examine.

**3. Drop the Index and Analyze:**

Let's see what happens when the index is removed. Use the following statement to drop the index\_title index:

**Query:**

DROP INDEX index\_title;



Run Query 1 again and observe the execution time. You should notice a performance regression compared to when the index was present. This reinforces the importance of indexes for speeding up specific queries.

**Remember:**

Indexes are beneficial for frequently used WHERE clause conditions on specific columns.

They add overhead during data insertion and updates due to index maintenance.

Analyze the trade-off between query performance improvement and data manipulation overhead before creating indexes.

This exercise demonstrates how indexes can significantly enhance query performance in Oracle by optimizing data retrieval. However, it's crucial to strategically place indexes considering factors like query patterns and data modification frequency.

**Assignment 6: Create a new database user with specific privileges using the CREATEUSER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.**

**Ans:**

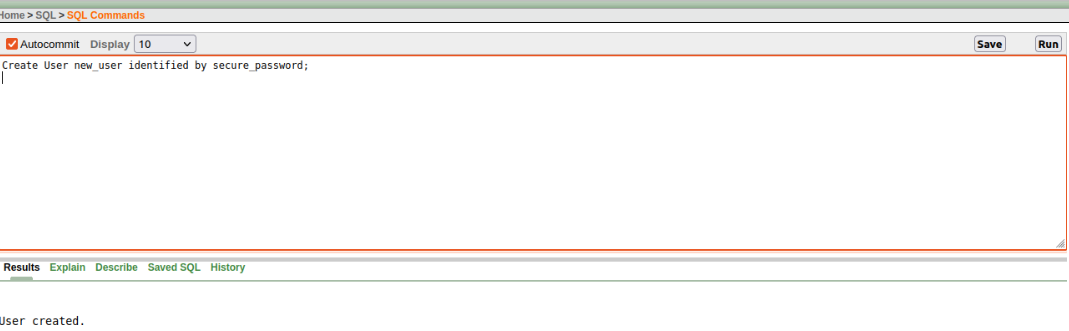
**Step 1:** Create a New Database User

First, let's create a new user in Oracle. This is done using the CREATE USER statement.

**Query:**

Create a new database user

CREATE USER new\_user IDENTIFIED BY secure\_password;

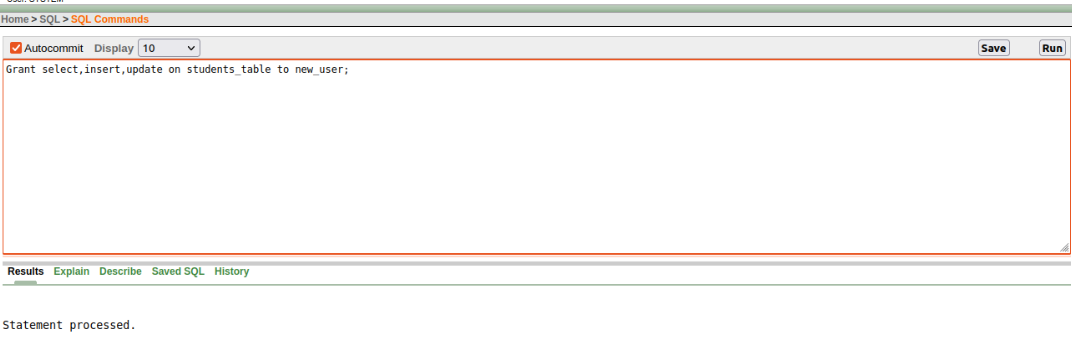


**Step 2:** Assign Specific Privileges to the User

Next, we need to grant certain privileges to the newly created user. For example, we can give the user SELECT, INSERT, and UPDATE privileges on a particular table.

**Query:**

GRANT SELECT, INSERT, UPDATE ON students\_table TO new\_user;

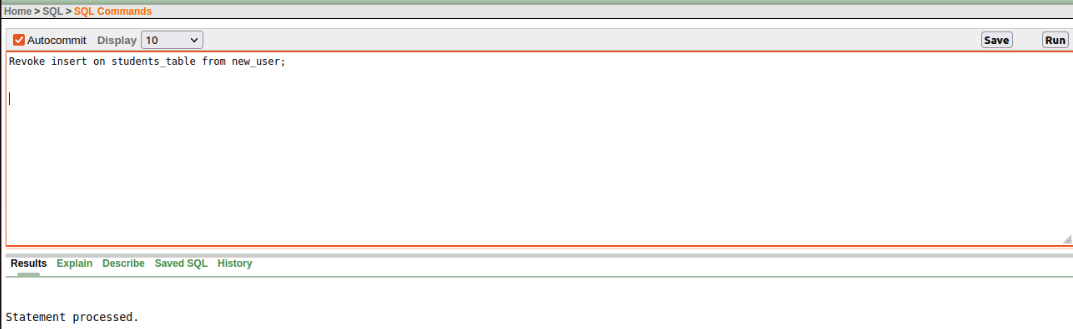


**Step 3:** Revoke Certain Privileges

Now, we'll write a script to revoke some of the previously granted privileges. For instance, we might want to revoke the INSERT privilege.

**Query:**

REVOKE INSERT ON students\_table FROM new\_user;

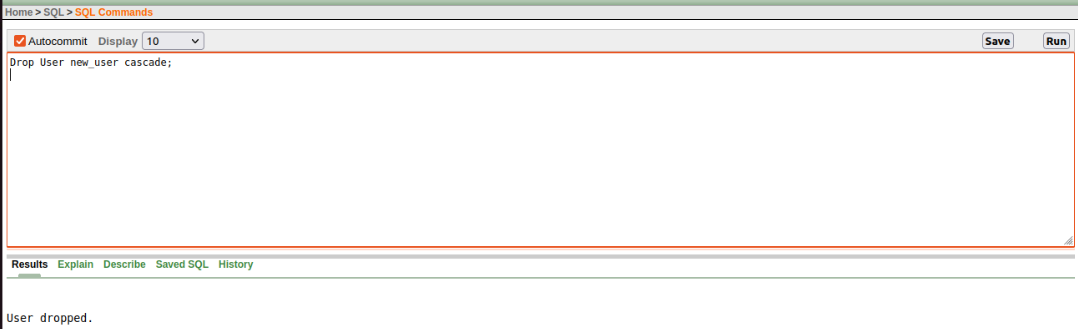


**Step 4:** Remove the User

Finally, we need to drop the user. Before doing this, ensure the user has no active sessions or owns any database objects. You might need to transfer ownership of objects the user owns and terminate their active sessions.

**Query:**

DROP USER new\_user CASCADE;



**Additional Notes:**

Reassign Ownership: If the user owns database objects, you might need to reassign ownership using commands like ALTER TABLE ... OWNER TO.

Terminate Active Sessions: If there are active sessions, identify and terminate them with ALTER SYSTEM KILL SESSION 'sid,serial#';.

By following these steps and using the provided SQL commands, you can successfully create a new Oracle database user, assign specific privileges, revoke certain privileges, and finally drop the user when necessary.

**Assignment 7: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.**

**Ans:**

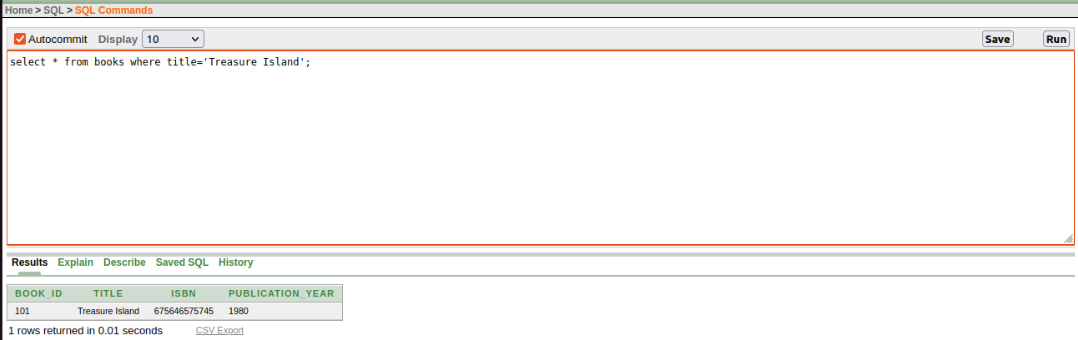
This script showcases various SQL statements for managing data in library database schema:

**1. INSERT Statements:**

**Single Row Insert:** Insert a new record into the Books table.

**Query:**

INSERT INTO Books (book\_id, title, isbn, publication\_year) VALUES (101, 'Treasure Island', '9780140439516', 1980);



**Multiple Row Insert:** Insert multiple records at once.

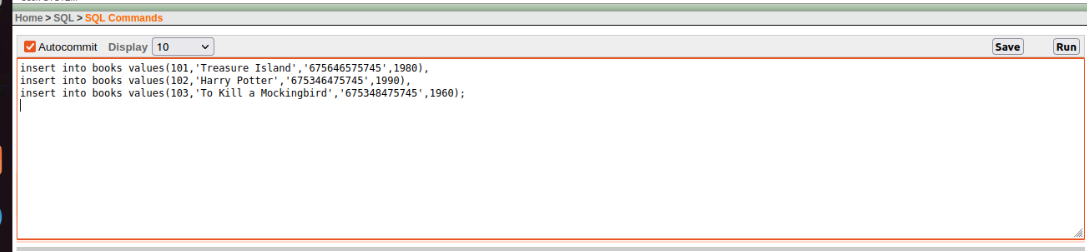
**Query:**

INSERT INTO Books (book\_id, title, isbn, publication\_year) VALUES

(101, 'Treasure Island', '9780140439516', 1980),

(102, 'Harry Potter, '9780547928225', 1990),

(103, 'To Kill a Mockingbird', '9780446310727', 1960);



**2. BULK INSERT for Large Datasets:**

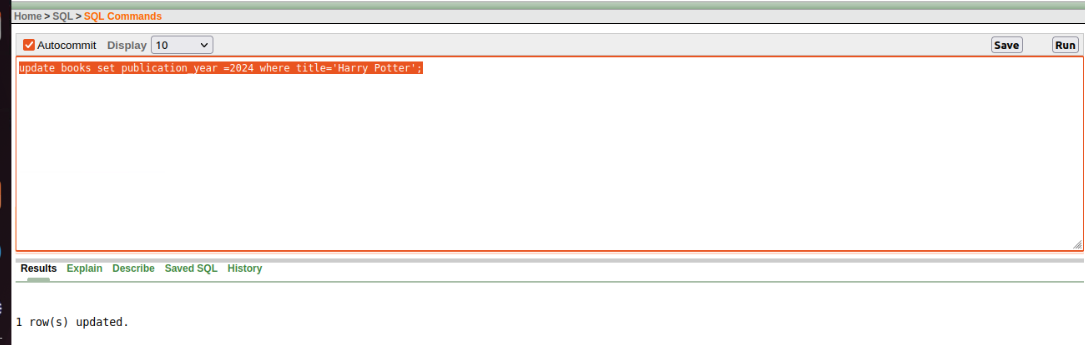
For loading a large number of records from an external file (e.g. CSV), consider using Oracle's SQL\*Loader utility or BULK COLLECT operations. Refer to Oracle documentation for detailed syntax and configuration specific to your environment.

**3. UPDATE Statements:**

Update an existing book's publication year in the Books table.

**Query:**

UPDATE Books SET publication\_year = 2024 WHERE title = 'Harry Potter';



Update Author’s information in the Authors table.

**Query:**

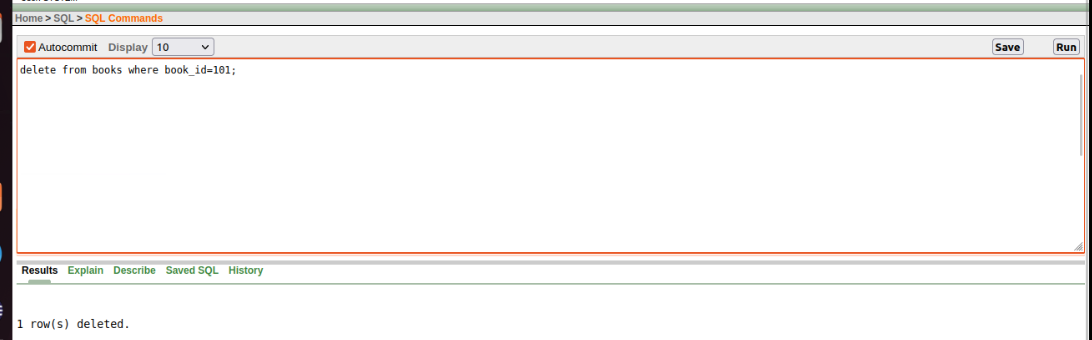
UPDATE Authors SET last\_name = 'Williams' WHERE first\_name = 'Jane';

**4. DELETE Statements:**

Delete a specific book record based on its ID.

**Query:**

DELETE FROM Books WHERE book\_id = 101;



Delete all loan records for a borrower with a specific ID.

**Query:**

DELETE FROM Authors WHERE author\_id = 5;

**Important Considerations:**

Remember to replace table and column names with your actual schema elements.

Use SEQUENCE objects (e.g., seq\_book\_id) for auto-incrementing primary keys.

Ensure data types in your INSERT statements match the corresponding table columns.

For DELETE operations, be cautious as deleted data cannot be easily recovered.