


islington college
(इस्लिङ्टन कलेज)

**Module Code & Module Title**

**CC6001NA Advanced Database Systems Development**

**Assessment Weightage & Type**

**Assignment**

**Year and Semester**

**2018-19 Autumn**

**Student Name: Rajat Shrestha**

**London Met ID: 17030954**

**College ID: np01cp4a170021**

**Assignment Due Date: 11th May 2020**

**Assignment Submission Date: 11th May 2020**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

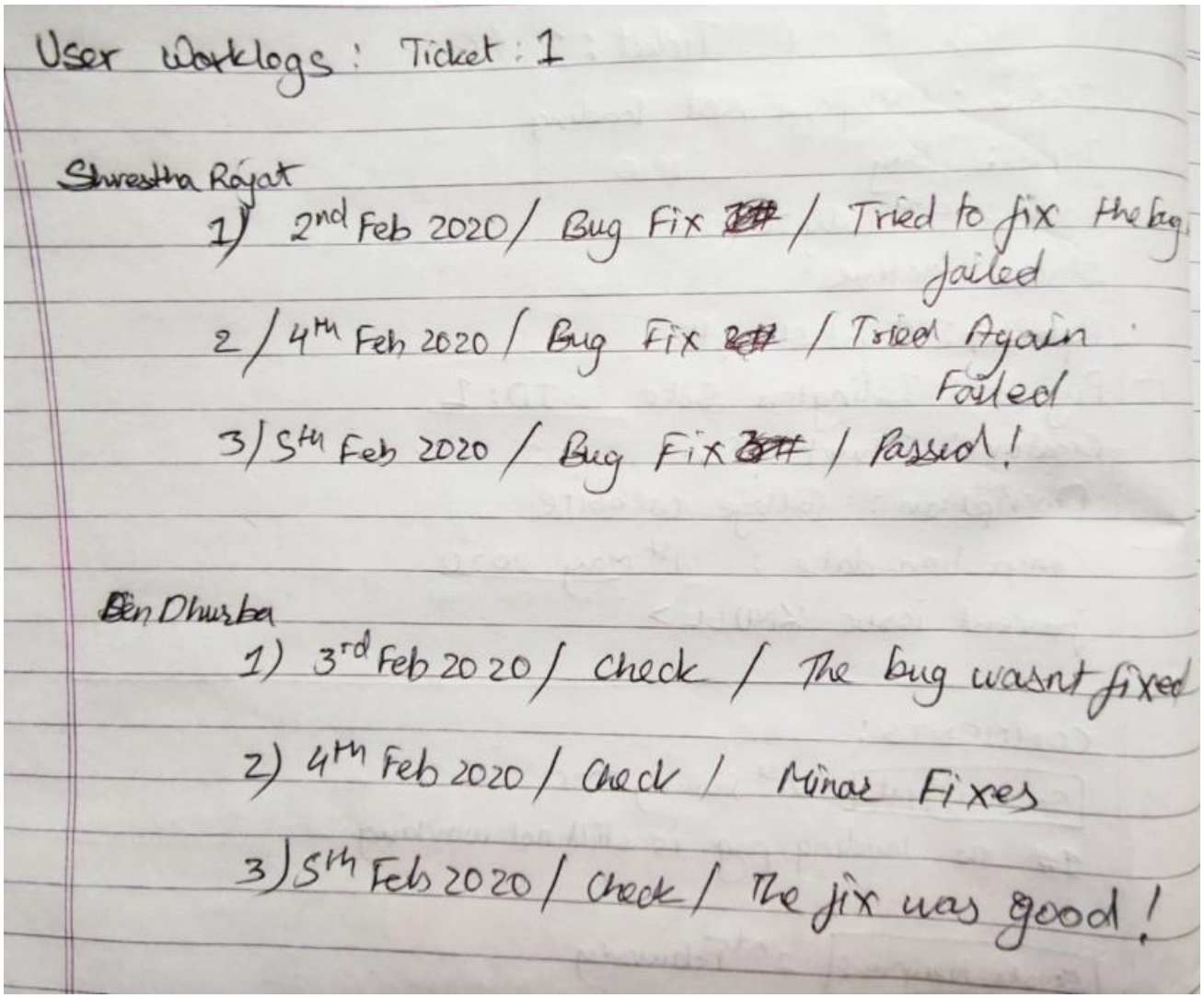
# Table of Contents

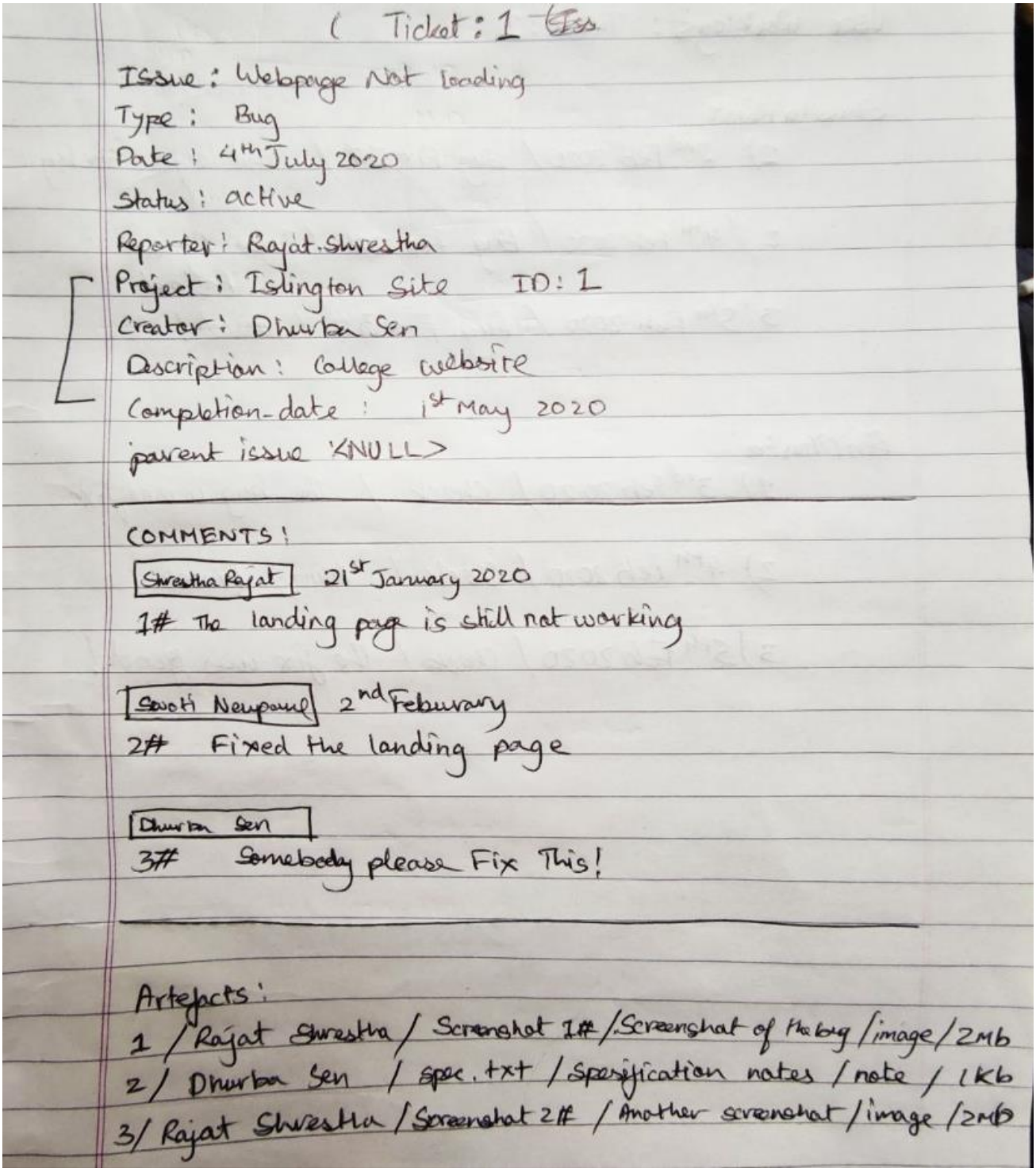
## Table of Figures:

## Table of Tables:

# 1. Introduction

The Issue Tracker Software is a fully functional propiatary software developed to track Issues of various projects where users can view, create, comment, add artefacts, assign other users, and track their worklogs in the Issue Tracker Software. The Issue Tracker is based on the Issue Tracking mechanism of popular projects such as github ang gitlab. At first the Database is modelled after a sample Ticket created with multiple comments, artefacts, assigned agents and woklogs. The Created Sample looks like the following:





( Ticket : 1 Iss

Issue : Webpage Not Loading
Type : Bug
Date : 4th July 2020
Status : active
Reporter : Rajat.Shrestha
Project : Islington Site ID : 1
Creator : Dhurba Sen
Description : College website
Completion-date : 1st May 2020
parent issue <NULL>

---

COMMENTS :

Shrestha Rajat 21st January 2020
1# The landing page is still not working

Sanoti Neupane 2nd Febuvary
2# Fixed the landing page

Dhurba Sen
3# Somebody please Fix This!

---

Artefacts :
1 / Rajat Shrestha / Screenshot 1# / Screenshot of the bug / image / 2MB
2 / Dhurba Sen / spec.txt / Specification notes / note / 1KB
3 / Rajat Shrestha / Screenshot 2# / Another screenshot / image / 2MB

# 2. Normalization

It is the process of correcting table structure to reduce redundancy and data anomalies, which minimizes storage space. It applies a series of rules called normal forms (Coronel & Morris, 2018). The database in the coursework is required to be normalized till 3NF.

### 2.1.1. Un-normalized Form (UNF):

A database is said to be in UNF when it has not been normalized at all. The rules for creating a un-normalized form are:

- Entity and its attributes should be identified
- A Primary key need to be stated
- The repeating group should be acknowledged.

In relation, a distinctive describing a group of multiple entries for a single key attribute occurrence can be known as a repeating group. Example: Multiple items purchased by a customer in a bill (Coronel & Morris, 2018). From the above table we can observe that the destination column has repeating groups in it, representing the given data in UNF:

Listing all the attributes of Ticket entity:

Tickets (**Ticket_ID**, Type, Date, Status, Reporter, Parent_Issue, Project_ID, Title, Description, Creator, Completion_Date, {Username, Comment_ID, Date, Text}, {Username, Artefact_ID, Title, Description, Category, Size, Data}, {Username, Name, Department, Designation, Contact, Password, {Worklog_ID, Date, Milestone, Details}})

### 2.1.2. First Normal Form (1NF):

In First Normal Form, only atomic values are allowed at each cell and discourage repeating groups. For the database to be in 1NF it must be in UNF. The other rules for 1NF are:

- Primary Keys should be identified.
- Repeating groups from UNF must be separated.
- New table should have Composite Primary key including the Primary key of the original table.

Removing Repeating Groups from UNF:

Tickets = (**Ticket_ID**, Type, Date, Status, Reporter*, Parent_Issue, Project_ID, Title, Description, Creator*, Completion_Date)

Ticket_Comments = (**Ticket_ID***, **Comment_ID**, Username*(Commenter), Date, Text)

Ticket_Artefacts = (**Ticket_ID***, **Artefact_ID**, Username*(Uploader), Title, Description, Category, Size, Data)

Assigned_User = (**Ticket_ID***, **Username**, Name, Department, Designation, Contact, Password)

Assigned_User_Worklog = (**Ticket_ID***, **Username***, **Worklog_ID**, Date, Milestone, Details)

### 2.1.3. Second Normal Form (2NF):

For the relation to be in Second Normal Form it must already be in First Normal Form and should not include any partial dependencies. Partial dependency is the kind of functional dependencies in which a non-key is dependent on part of a composite key (Coronel & Morris, 2018). The rules for 2NF are:

- All the functional dependencies between no key and composite key and parts of the composite key should be shown for the entities with composite Primary Key.
- Non-keys which are dependent on a part of composite key should be moved to a new table and identify its keys.

Removing Partial Dependencies From 1NF By checking in each Entity:

**Ticket_ID** ⇒ Type, Date, Status, Reporter*, Parent_Issue, Project_ID, Title, Description, Creator*, Completion_Date

**Ticket_ID***, **Comment_ID** ⇒ Username*(Commenter), Date, Text
**Comment_ID** ⇒ Username*(Commenter), Date, Text

**Ticket_ID***, **Artefact_ID** ⇒ Username*(Uploader), Title, Description, Category, Size, Data
**Artefact_ID** ⇒ Username*(Uploader), Title, Description, Category, Size, Data

**Ticket_ID***, **Username** ⇒ Name, Department, Designation. Contact, Password
**Username** ⇒ Name, Department, Designation, Contact, Password

**Ticket_ID***, **Username***, **Worklog_ID** ⇒ Date, Milestone, Details
**Worklog_ID** ⇒ Date, Milestone, Details

Tickets = (**Ticket_ID**, Type, Date, Status, Reporter*, Parent_Issue, Project_ID, Title, Description, Creator*, Completion_Date)

Ticket_Comments = (**Ticket_ID***, **Comment_ID***)

Comments = (**Comment_ID**, Username*(Commenter), Date, Text)

Ticket_Artefacts = (**Ticket_ID***, **Artefact_ID***)

Artefacts = (**Artefact_ID**, Username*(Uploader), Title, Description, Category, Size, Data)

Assigned_User = (**Ticket_ID***, **Username***)

User = (**Username**, Name, Department, Designation, Contact, Password)

Assigned_User_Worklog = (**Ticket_ID***, **Username***, **Worklog_ID***)

User_Worklog = (**Worklog_ID**, Date, Milestone, Details)

### 2.1.4. Third Normal Form (3NF):

For the database to be in Third Normal Form it must already be in Second Normal Form and should not include any transitive dependencies. Transitive dependency is a type of functional dependency in which a non-key is dependent on another non-key element (Coronel & Morris, 2018). The rules for 3NF are:

- Functional dependencies between no key and non-key should be separated into a new table in case of an entity with multiple Non-key.
- Primary Keys of the new table should be identified.

Separating Transitive Dependencies:

In tickets Ticket_ID gives Project_ID but Project_Id can alone give Title, Description, Creator, and Completion_Date of the project.

Tickets = (**Ticket_ID**, Type, Date, Status, Reporter*, Parent_Issue*, Project_ID*)

Project = (**Project_ID**, Title, Description, Creator*, Completion_Date)

Ticket_Comments = (**Ticket_ID***, **Comment_ID***)

Comments = (**Comment_ID**, Username*(Commenter), Date, Text)

Ticket_Artefacts = (**Ticket_ID***, **Artefact_ID***)

Artefacts = (**Artefact_ID**, Username*(Uploader), Title, Description, Category, Size, Data)

Assigned_User = (**Ticket_ID***, **Username***)

User = (**Username**, Name, Department, Designation, Contact, Password)

Assigned_User_Worklog = (**Ticket_ID***, **Username***, **Worklog_ID***)

Worklog = (**Worklog_ID**, Date, Milestone, Details)

# 3. ER-Diagram

An Entity-relationship diagram represents the relationship between the entities in the database. ERD is one of the most common data but effective models where objects are divided into entities and their characteristics into attributes and entities are connected via elaborate relationships. (Nishadha, 2017). SQL Developer Data Modeler is a free graphical tool that improves productivity and simplifies data modelling tasks where users can create, browse and edit, logical, relational, physical, multi-dimensional, and data type models supporting collaborative development through integrated source code control (Oracle, 2020). SQL Developer Data Modeler was used to Create the final ER-Diagram.

## 3.1. Assumptions:

- Users are created by the Database Administrator or the manager
- Users can create Projects
- Users can create Tickets in Project (For Bug, Enhancement, or TODO according to Analyst)
- Users can add multiple Artefacts and Comments in a Ticket
- Multiple Users can be assigned to a ticket
- Multiple Users have separate Worklogs for the ticket
- All of the above Functions are traced
- Users will be identified by username
- Ticket, Comment, Artefact, Worklog and Projects will have Auto increasing Integer assigned as primary key for Ticket_ID, Comment_ID, Artefact_ID, Worklog_ID and Project_ID.

Therefore, the final ER-Diagram will look like the following:

## 3.2. Final ER-diagram:

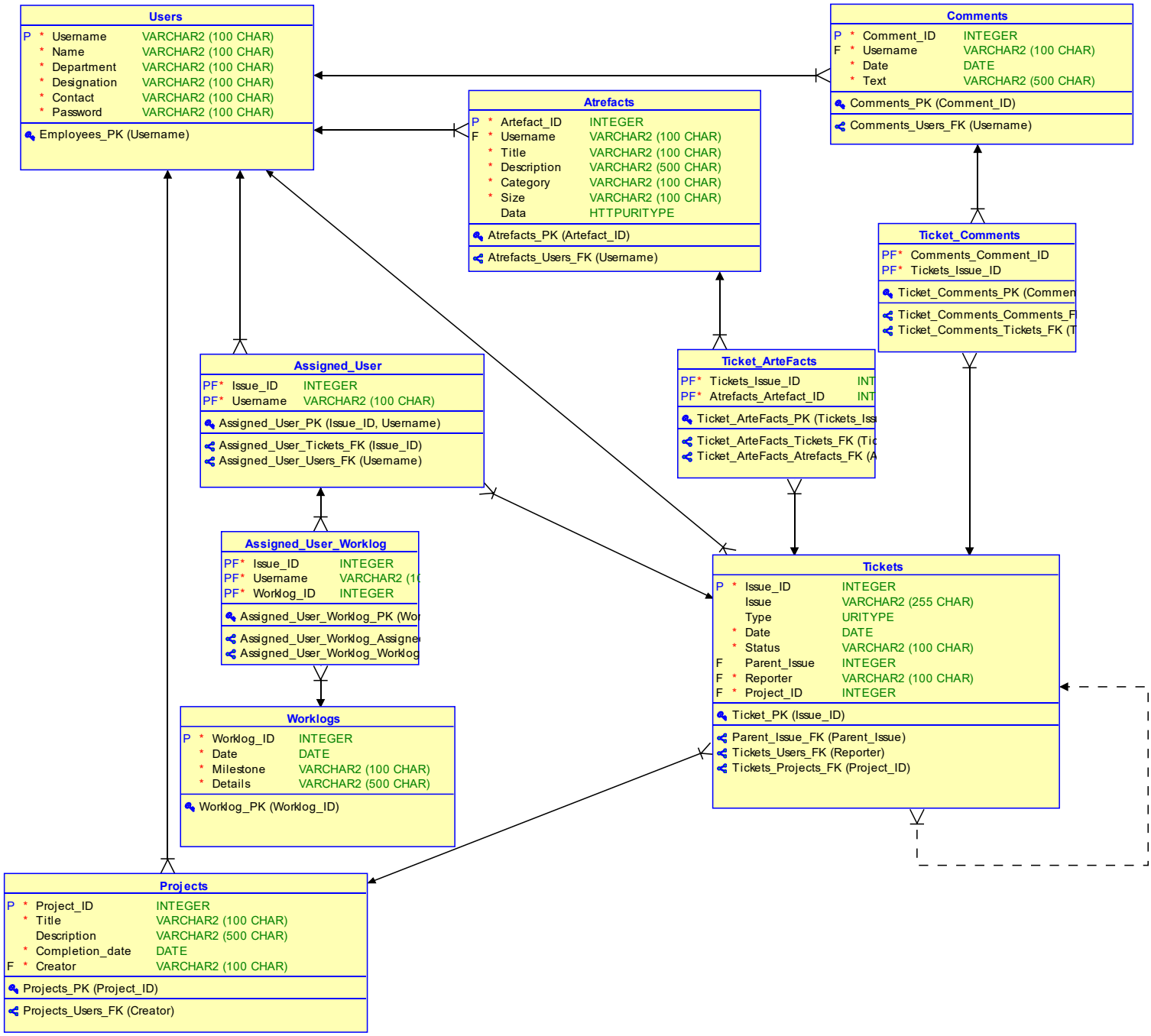



*Figure 1: ER-Diagram created using the given entities*

# 4. Data Dictionary

*Table 1: Data Dictionary for the Users table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Username | Y | Logical Type | VARCHAR | P | | VARCHAR2 | 100 | |
| Name | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Department | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Designation | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Contact | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Password | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |

*Table 2: Data Dictionary for Projects table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Project_ID | Y | Logical Type | Integer | P | | INTEGER | | |
| Title | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Description | N | Logical Type | VARCHAR | | | VARCHAR2 | 500 | |
| Completion_date | Y | Logical Type | Datetime | | | DATE | | |
| Creator | Y | Logical Type | VARCHAR | | F | VARCHAR2 | 100 | |

*Table 3: Data Dictionary for the Tickets table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Issue_ID | Y | Logical Type | Integer | P | | INTEGER | | |
| Issue | N | Logical Type | VARCHAR | | | VARCHAR2 | 255 | |
| Type | N | Logical Type | URIType | | | VARCHAR2 | 100 | |
| Date | Y | Logical Type | Date | | | DATE | | |
| Status | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Parent_Issue | N | Logical Type | Integer | | F | INTEGER | | |
| Project_ID | Y | Logical Type | Integer | | F | INTEGER | | |
| Reporter | Y | Logical Type | VARCHAR | | F | VARCHAR2 | 100 | |

*Table 4: Data Dictionary for the Artefacts table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Artefact_ID | Y | Logical Type | Integer | P | | INTEGER | | |
| Username | Y | Logical Type | VARCHAR | | F | VARCHAR2 | 100 | |
| Title | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Description | Y | Logical Type | VARCHAR | | | VARCHAR2 | 500 | |
| Category | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Size | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Data | N | Logical Type | HTTPURIType | | | HTTPURITYPE | | |

*Table 5: Data Dictionary for the Comments table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Comment_ID | Y | Logical Type | Integer | P | | INTEGER | | |
| Username | Y | Logical Type | VARCHAR | | F | VARCHAR2 | 100 | |
| Date | Y | Logical Type | Date | | | DATE | | |
| Text | Y | Logical Type | VARCHAR | | | VARCHAR2 | 500 | |

*Table 6: Data Dictionary for the Worklog table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Worklog_ID | Y | Logical Type | Integer | P | | INTEGER | | |
| Date | Y | Logical Type | Date | | | DATE | | |
| Milestone | Y | Logical Type | VARCHAR | | | VARCHAR2 | 100 | |
| Details | Y | Logical Type | VARCHAR | | | VARCHAR2 | 500 | |

*Table 7: Data Dictionary for Ticket_Comments Bridge table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Tickets_Issue_ID | Y | Logical Type | Integer | P | F | INTEGER | | 1 |
| Comments_Comment_ID | Y | Logical Type | Integer | P | F | INTEGER | | 1 |

*Table 8: Data Dictionary for Ticket_Artefacts Bridge table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Tickets_Issue_ID | Y | Logical Type | Integer | P | F | INTEGER | | 1 |
| Atrefacts_Artefact_ID | Y | Logical Type | Integer | P | F | INTEGER | | 1 |

*Table 9: Data Dictionary for Assigned_user Bridge table*

| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Issue_ID | Y | Logical Type | Integer | P | F | INTEGER | | |
| Username | Y | Logical Type | VARCHAR | P | F | VARCHAR2 | 100 | |

*Table 10: Data Dictionary for Assigned_User_Worklog Bridge table*

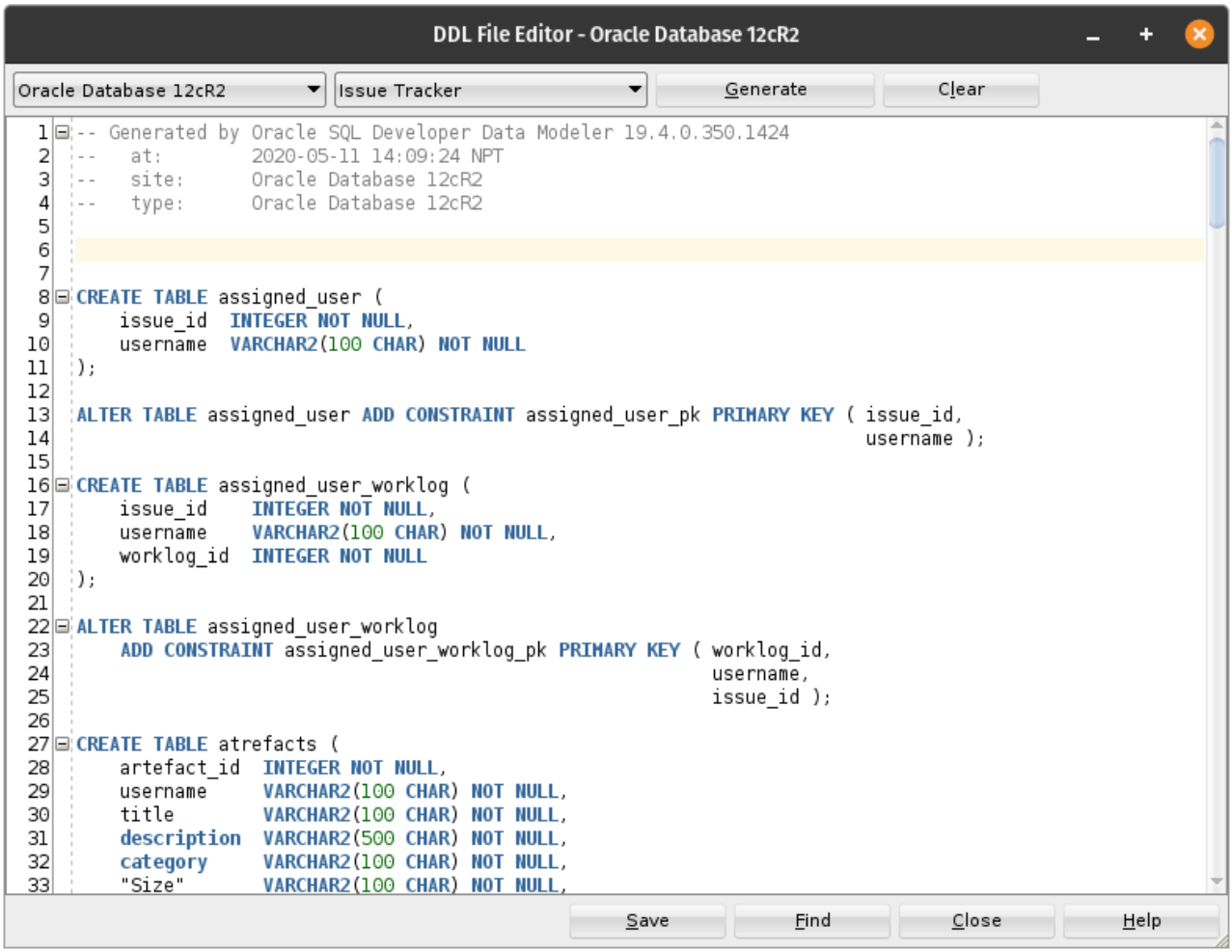
| Column_Name | Mandatory | DataType Kind | Logical Type Name | PK | FK | Native Type | T Size | Example |
|---|---|---|---|---|---|---|---|---|
| Issue_ID | Y | Logical Type | Integer | P | F | INTEGER | | |
| Username | Y | Logical Type | VARCHAR | P | F | VARCHAR2 | 100 | |
| Worklog_ID | Y | Logical Type | Integer | P | F | INTEGER | | |

# 5. Generation of Database

## 5.1. Create Statements

### 5.1.1. Generating DDL Script and Creating Tables:

The Datamodeler allows the generation of DDL Script according to the designed ER-Model. The following figure shows how the script was created and the script is also included. Then the script is pasted into the SQL Developer and ran to generate the tables.




```
1  -- Generated by Oracle SQL Developer Data Modeler 19.4.0.350.1424
2  --   at:        2020-05-11 14:09:24 NPT
3  --   site:      Oracle Database 12cR2
4  --   type:      Oracle Database 12cR2
5
6
7
8  CREATE TABLE assigned_user (
9      issue_id  INTEGER NOT NULL,
10     username  VARCHAR2(100 CHAR) NOT NULL
11 );
12
13 ALTER TABLE assigned_user ADD CONSTRAINT assigned_user_pk PRIMARY KEY ( issue_id,
14                                                                        username );
15
16 CREATE TABLE assigned_user_worklog (
17     issue_id    INTEGER NOT NULL,
18     username    VARCHAR2(100 CHAR) NOT NULL,
19     worklog_id  INTEGER NOT NULL
20 );
21
22 ALTER TABLE assigned_user_worklog
23     ADD CONSTRAINT assigned_user_worklog_pk PRIMARY KEY ( worklog_id,
24                                                           username,
25                                                           issue_id );
26
27 CREATE TABLE atrefacts (
28     artefact_id  INTEGER NOT NULL,
29     username     VARCHAR2(100 CHAR) NOT NULL,
30     title        VARCHAR2(100 CHAR) NOT NULL,
31     description  VARCHAR2(500 CHAR) NOT NULL,
32     category     VARCHAR2(100 CHAR) NOT NULL,
33     "Size"       VARCHAR2(100 CHAR) NOT NULL,
```

*Figure 2: Process of generating the DDL Script via SQL developer Datamodeler*

### 5.1.2. Running DDL Scripts:

```
CREATE TABLE assigned_user (
    issue_id  INTEGER NOT NULL,
    username  VARCHAR2(100 CHAR) NOT NULL
);

ALTER TABLE assigned_user ADD CONSTRAINT assigned_user_pk PRIMARY KEY ( issue_id,
                                                                       username );

CREATE TABLE assigned_user_worklog (
    issue_id    INTEGER NOT NULL,
    username    VARCHAR2(100 CHAR) NOT NULL,
    worklog_id  INTEGER NOT NULL
);

ALTER TABLE assigned_user_worklog
    ADD CONSTRAINT assigned_user_worklog_pk PRIMARY KEY ( worklog_id,
                                                          username,
                                                          issue_id );

CREATE TABLE atrefacts (
    artefact_id  INTEGER NOT NULL,
    username     VARCHAR2(100 CHAR) NOT NULL,
    title        VARCHAR2(100 CHAR) NOT NULL,
    description  VARCHAR2(500 CHAR) NOT NULL,
    category     VARCHAR2(100 CHAR) NOT NULL,
    "Size"       VARCHAR2(100 CHAR) NOT NULL,
    data         httpuritype
);

ALTER TABLE atrefacts ADD CONSTRAINT atrefacts_pk PRIMARY KEY ( artefact_id );

CREATE TABLE comments (
    comment_id  INTEGER NOT NULL,
    username    VARCHAR2(100 CHAR) NOT NULL,
    "Date"      DATE NOT NULL,
    text        VARCHAR2(500 CHAR) NOT NULL
);

ALTER TABLE comments ADD CONSTRAINT comments_pk PRIMARY KEY ( comment_id );

CREATE TABLE projects (
    project_id       INTEGER NOT NULL,
    title            VARCHAR2(100 CHAR) NOT NULL,
    description      VARCHAR2(500 CHAR),
    completion_date  DATE NOT NULL,
    creator          VARCHAR2(100 CHAR) NOT NULL
```

```
);

ALTER TABLE projects ADD CONSTRAINT projects_pk PRIMARY KEY ( project_id );

CREATE TABLE ticket_artefacts (
    tickets_issue_id       INTEGER NOT NULL,
    atrefacts_artefact_id  INTEGER NOT NULL
);

ALTER TABLE ticket_artefacts ADD CONSTRAINT ticket_artefacts_pk PRIMARY KEY ( ticket
s_issue_id,
                                                                              atrefa
cts_artefact_id );

CREATE TABLE ticket_comments (
    comments_comment_id  INTEGER NOT NULL,
    tickets_issue_id     INTEGER NOT NULL
);

ALTER TABLE ticket_comments ADD CONSTRAINT ticket_comments_pk PRIMARY KEY ( comments
_comment_id,
                                                                            tickets_
issue_id );

CREATE TABLE tickets (
    issue_id      INTEGER NOT NULL,
    issue         VARCHAR2(255 CHAR),
    type          URITYPE,
    "Date"        DATE NOT NULL,
    status        VARCHAR2(100 CHAR) NOT NULL,
    parent_issue  INTEGER,
    reporter      VARCHAR2(100 CHAR) NOT NULL,
    project_id    INTEGER NOT NULL
);

ALTER TABLE tickets ADD CONSTRAINT ticket_pk PRIMARY KEY ( issue_id );

CREATE TABLE users (
    username     VARCHAR2(100 CHAR) NOT NULL,
    name         VARCHAR2(100 CHAR) NOT NULL,
    department   VARCHAR2(100 CHAR) NOT NULL,
    designation  VARCHAR2(100 CHAR) NOT NULL,
    contact      VARCHAR2(100 CHAR) NOT NULL,
    password     VARCHAR2(100 CHAR) NOT NULL
);

ALTER TABLE users ADD CONSTRAINT employees_pk PRIMARY KEY ( username );
```

```
CREATE TABLE worklogs (
    worklog_id  INTEGER NOT NULL,
    "Date"      DATE NOT NULL,
    milestone   VARCHAR2(100 CHAR) NOT NULL,
    details     VARCHAR2(500 CHAR) NOT NULL
);

ALTER TABLE worklogs ADD CONSTRAINT worklog_pk PRIMARY KEY ( worklog_id );

ALTER TABLE assigned_user
    ADD CONSTRAINT assigned_user_tickets_fk FOREIGN KEY ( issue_id )
        REFERENCES tickets ( issue_id );

ALTER TABLE assigned_user
    ADD CONSTRAINT assigned_user_users_fk FOREIGN KEY ( username )
        REFERENCES users ( username );

ALTER TABLE assigned_user_worklog
    ADD CONSTRAINT assigned_user_worklog_assigned_user_fk FOREIGN KEY ( issue_id,
                                                                        username )
        REFERENCES assigned_user ( issue_id,
                                   username );

ALTER TABLE assigned_user_worklog
    ADD CONSTRAINT assigned_user_worklog_worklogs_fk FOREIGN KEY ( worklog_id )
        REFERENCES worklogs ( worklog_id );

ALTER TABLE atrefacts
    ADD CONSTRAINT atrefacts_users_fk FOREIGN KEY ( username )
        REFERENCES users ( username );

ALTER TABLE comments
    ADD CONSTRAINT comments_users_fk FOREIGN KEY ( username )
        REFERENCES users ( username );

ALTER TABLE tickets
    ADD CONSTRAINT parent_issue_fk FOREIGN KEY ( parent_issue )
        REFERENCES tickets ( issue_id );

ALTER TABLE projects
    ADD CONSTRAINT projects_users_fk FOREIGN KEY ( creator )
        REFERENCES users ( username );

ALTER TABLE ticket_artefacts
    ADD CONSTRAINT ticket_artefacts_atrefacts_fk FOREIGN KEY ( atrefacts_artefact_id
 )
        REFERENCES atrefacts ( artefact_id );
```

```
ALTER TABLE ticket_artefacts
    ADD CONSTRAINT ticket_artefacts_tickets_fk FOREIGN KEY ( tickets_issue_id )
        REFERENCES tickets ( issue_id );

ALTER TABLE ticket_comments
    ADD CONSTRAINT ticket_comments_comments_fk FOREIGN KEY ( comments_comment_id )
        REFERENCES comments ( comment_id );

ALTER TABLE ticket_comments
    ADD CONSTRAINT ticket_comments_tickets_fk FOREIGN KEY ( tickets_issue_id )
        REFERENCES tickets ( issue_id );

ALTER TABLE tickets
    ADD CONSTRAINT tickets_projects_fk FOREIGN KEY ( project_id )
        REFERENCES projects ( project_id );

ALTER TABLE tickets
    ADD CONSTRAINT tickets_users_fk FOREIGN KEY ( reporter )
        REFERENCES users ( username );

CREATE SEQUENCE a_artefact_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER a_artefact_id_trg BEFORE
    INSERT ON atrefacts
    FOR EACH ROW
    WHEN ( new.artefact_id IS NULL )
BEGIN
    :new.artefact_id := a_artefact_id_seq.nextval;
END;
/

CREATE SEQUENCE c_comment_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER c_comment_id_trg BEFORE
    INSERT ON comments
    FOR EACH ROW
    WHEN ( new.comment_id IS NULL )
BEGIN
    :new.comment_id := c_comment_id_seq.nextval;
END;
/

CREATE SEQUENCE p_project_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER p_project_id_trg BEFORE
    INSERT ON projects
    FOR EACH ROW
    WHEN ( new.project_id IS NULL )
```

```
BEGIN
    :new.project_id := p_project_id_seq.nextval;
END;
/

CREATE SEQUENCE t_issue_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER t_issue_id_trg BEFORE
    INSERT ON tickets
    FOR EACH ROW
    WHEN ( new.issue_id IS NULL )
BEGIN
    :new.issue_id := t_issue_id_seq.nextval;
END;
/

CREATE SEQUENCE w_worklog_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER w_worklog_id_trg BEFORE
    INSERT ON worklogs
    FOR EACH ROW
    WHEN ( new.worklog_id IS NULL )
BEGIN
    :new.worklog_id := w_worklog_id_seq.nextval;
END;
/
```

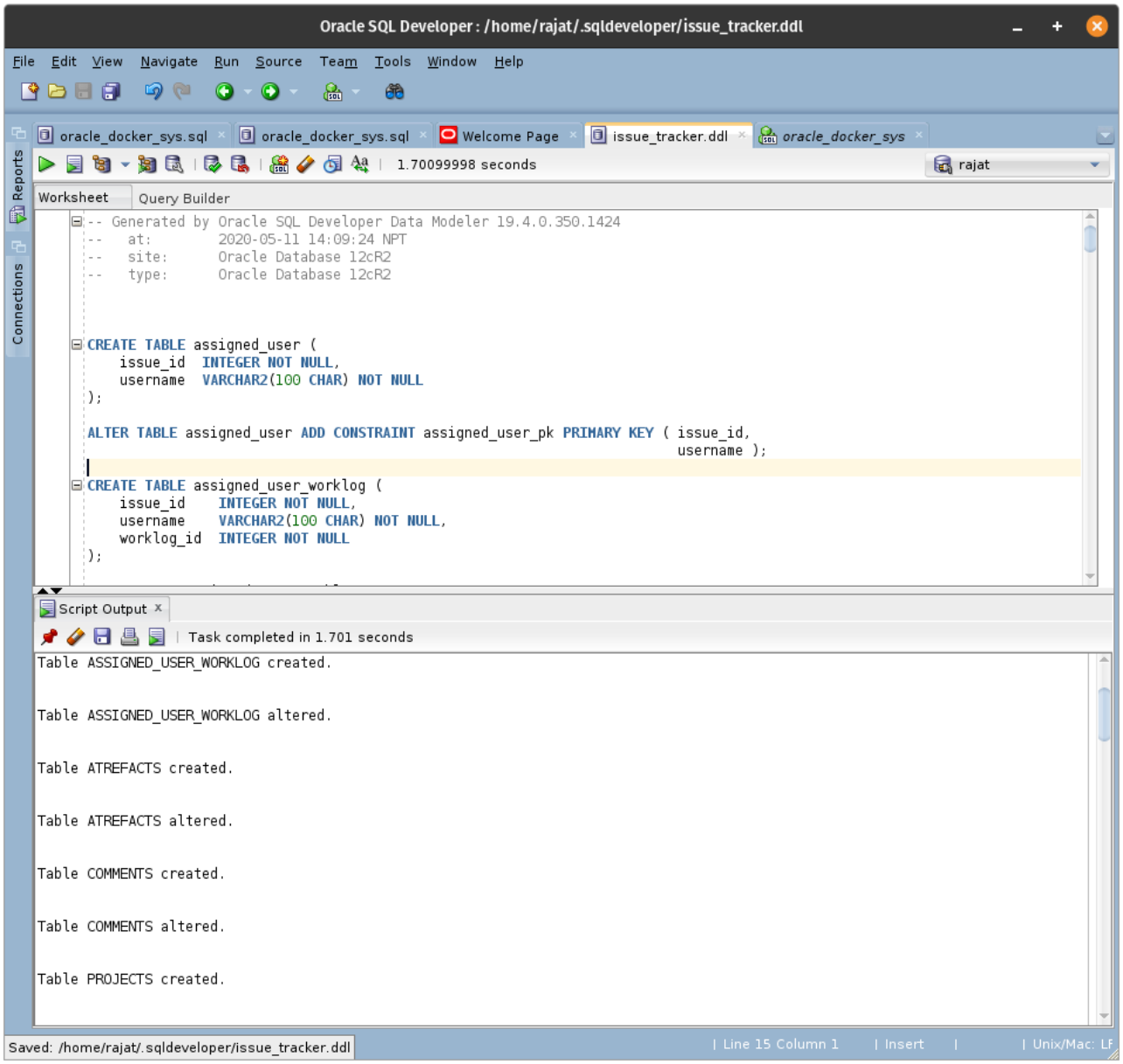



*Figure 3: Running the DDL Script in SQL Developer*

# 6. User Roles:

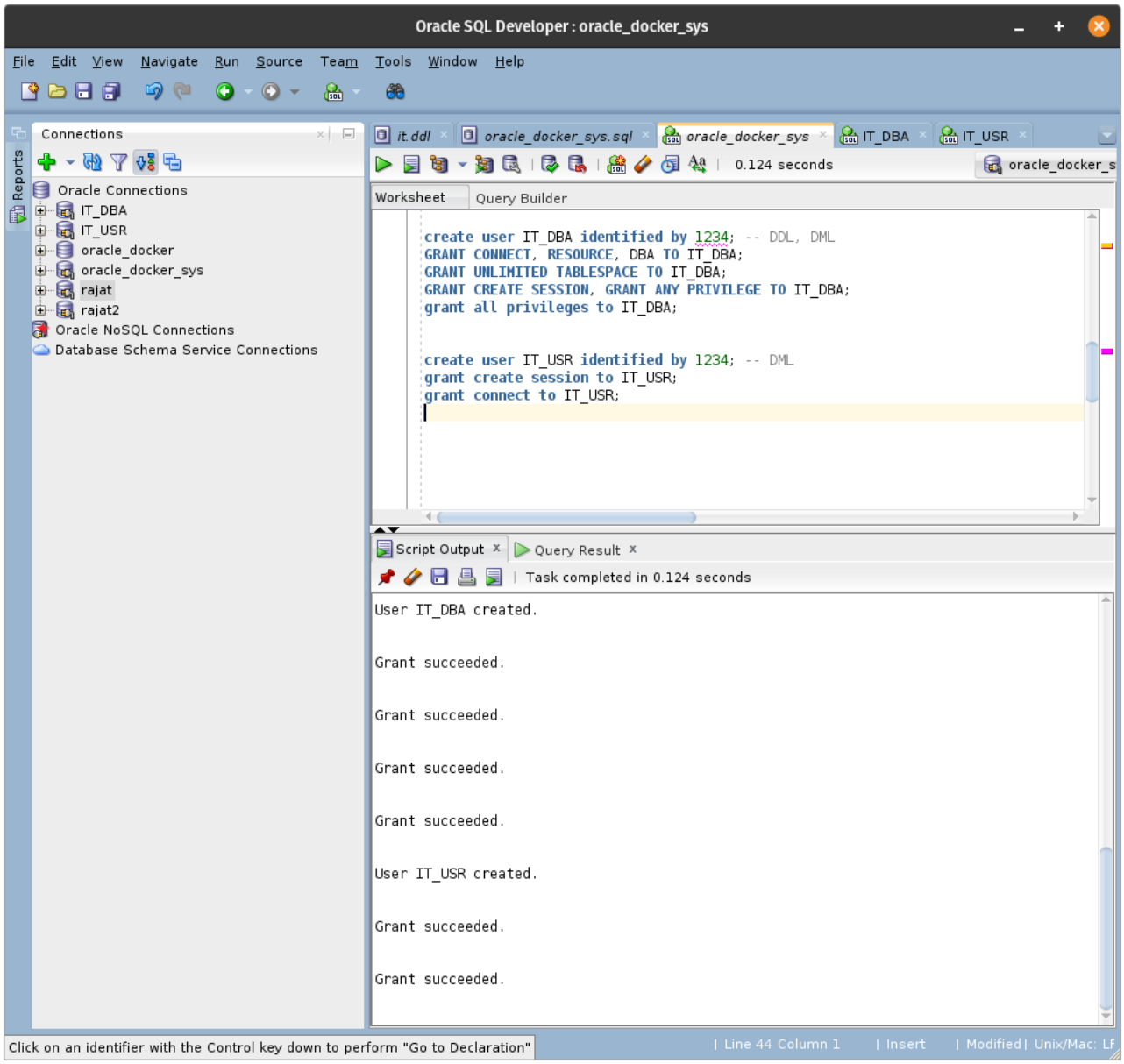



*Figure 4: Creating Two user with different Privileges*

# 7. Deadlock in Ticket:

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks.

For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish. Transaction B holds locks on those very rows in the Orders table but needs to update the rows in the Accounts table held by Transaction A. Transaction A cannot complete its transaction because of the lock on Orders. Transaction B cannot complete its transaction because of the lock on Accounts. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions. The following figure shows this situation.

https://docs.oracle.com/javadb/10.8.3.0/devguide/cdevconcepts28436.html

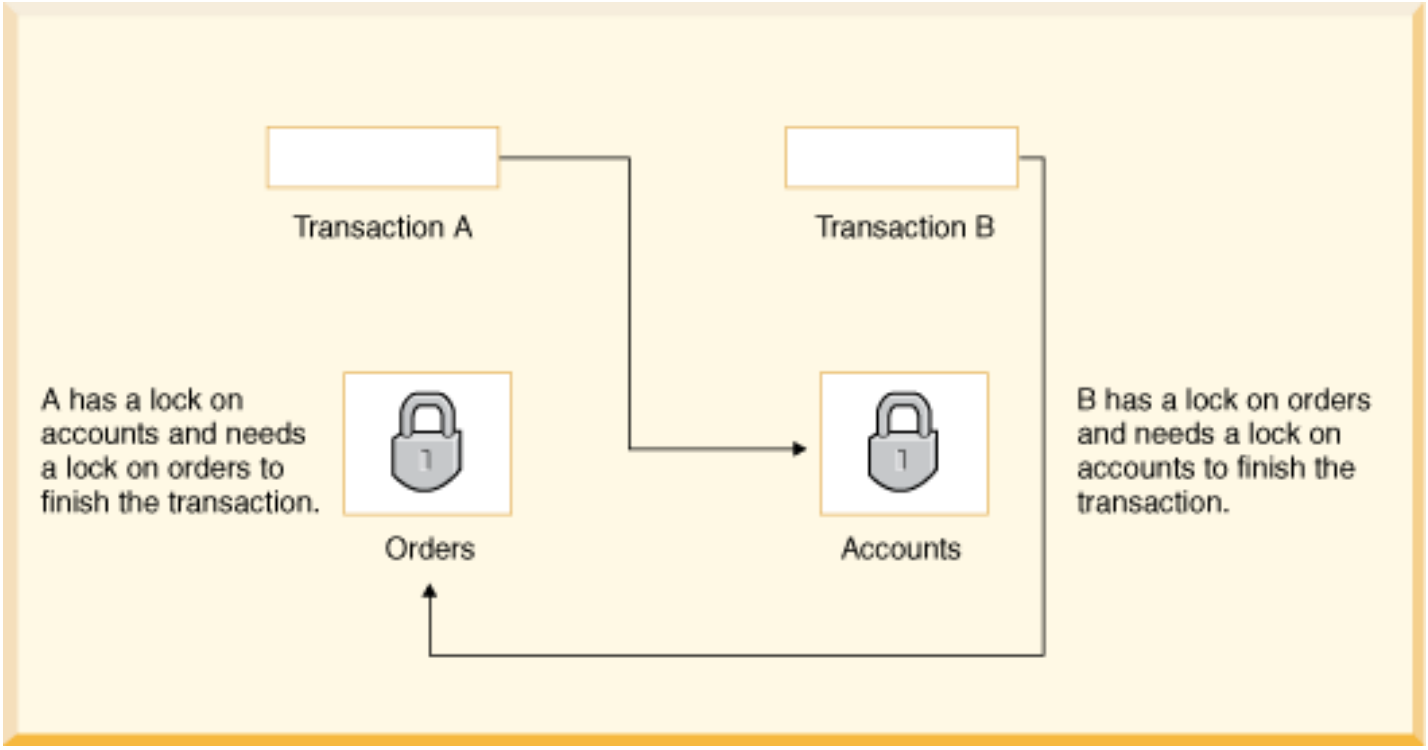



*Figure 5: Deadlock situation example (oracle).*

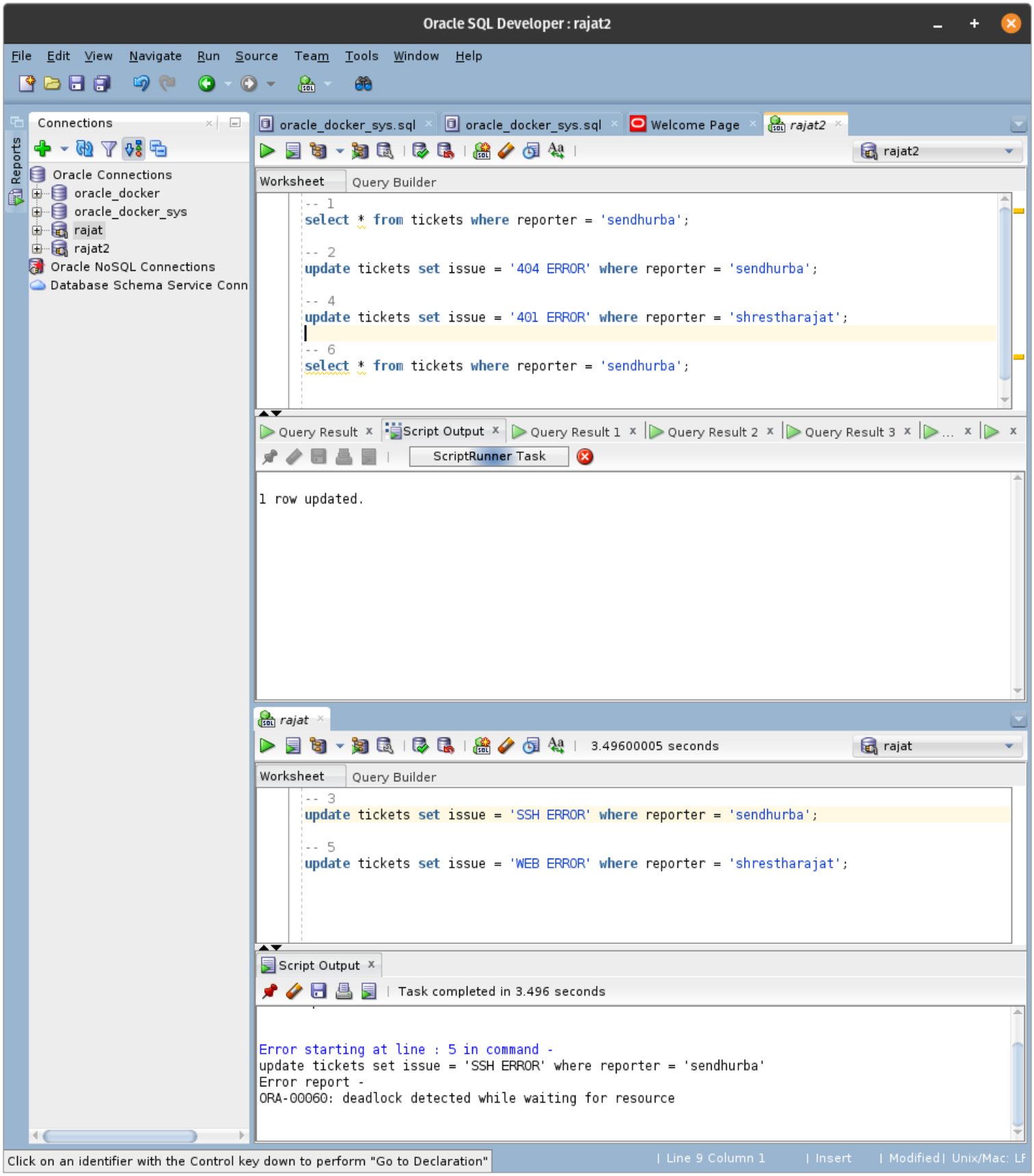



*Figure 6: Deadlock situation in Tickets Table*

# 8. Shared Lock & Exclusive lock:

## 8.2. Shared Lock (S):

A Shared Lock is basically a read-only lock for a row-level. Any number of resources can fetch the data to read when the shared lock is present on the resource. That means that many process IDs can have a shared lock on the same resource to read the respective data.

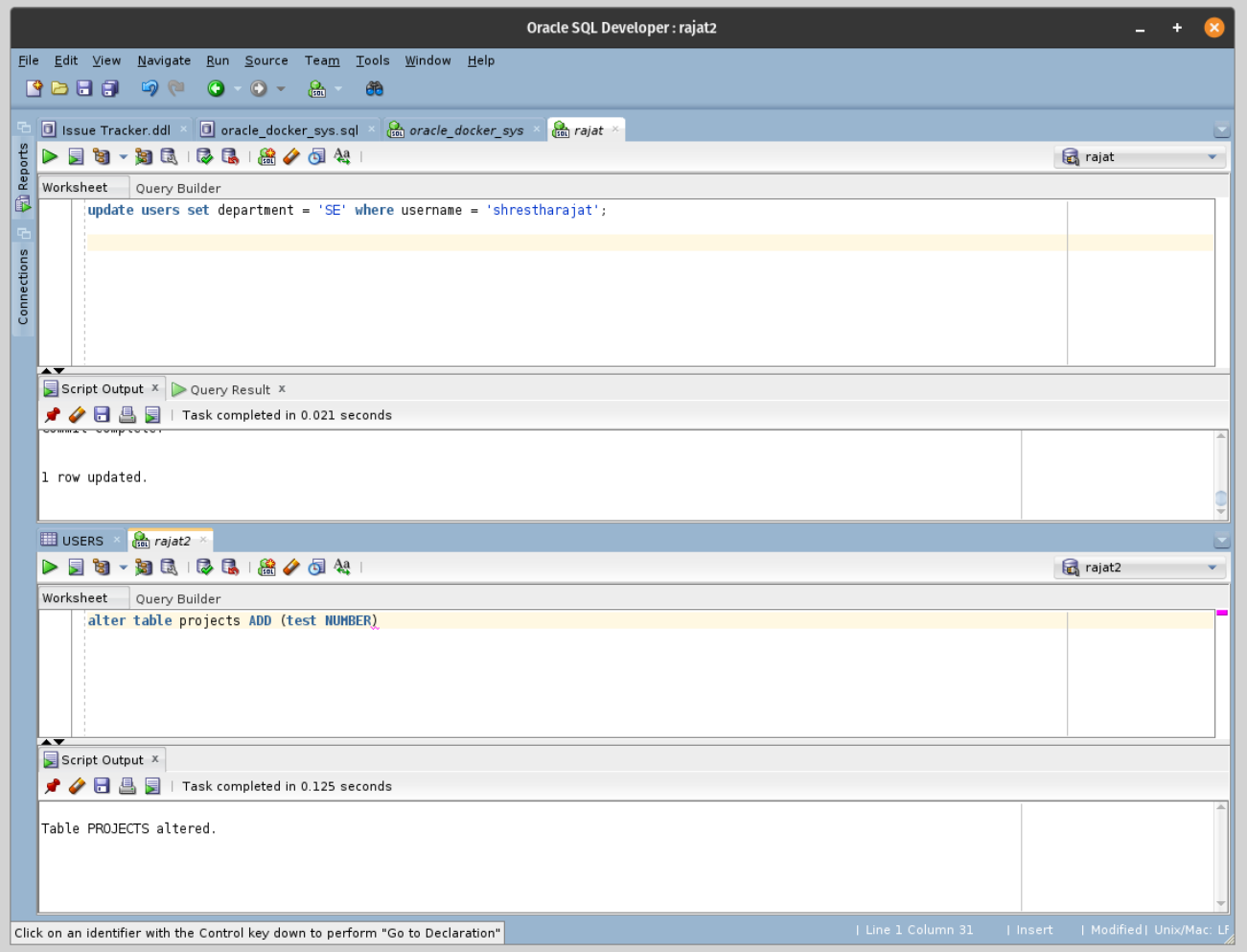



*Figure 7: Shared lock example*

## 8.3. Exclusive Lock (X):

The Exclusive Lock is used and valid on a single transaction, that locks either row or a page depending on the data. The mechanism for understanding is simple, where an exclusive lock can be applied only on a single resource. There cannot be more than one exclusive lock on the same resource. Either Insert, Update or Delete commands happen over with the Exclusive lock and these commands will not be in effect until the exclusive lock is released from the resource.

The exclusive lock occurs while updating the table but not committing as shown in the tickets table so the other session is halted until the first lock has been settled.

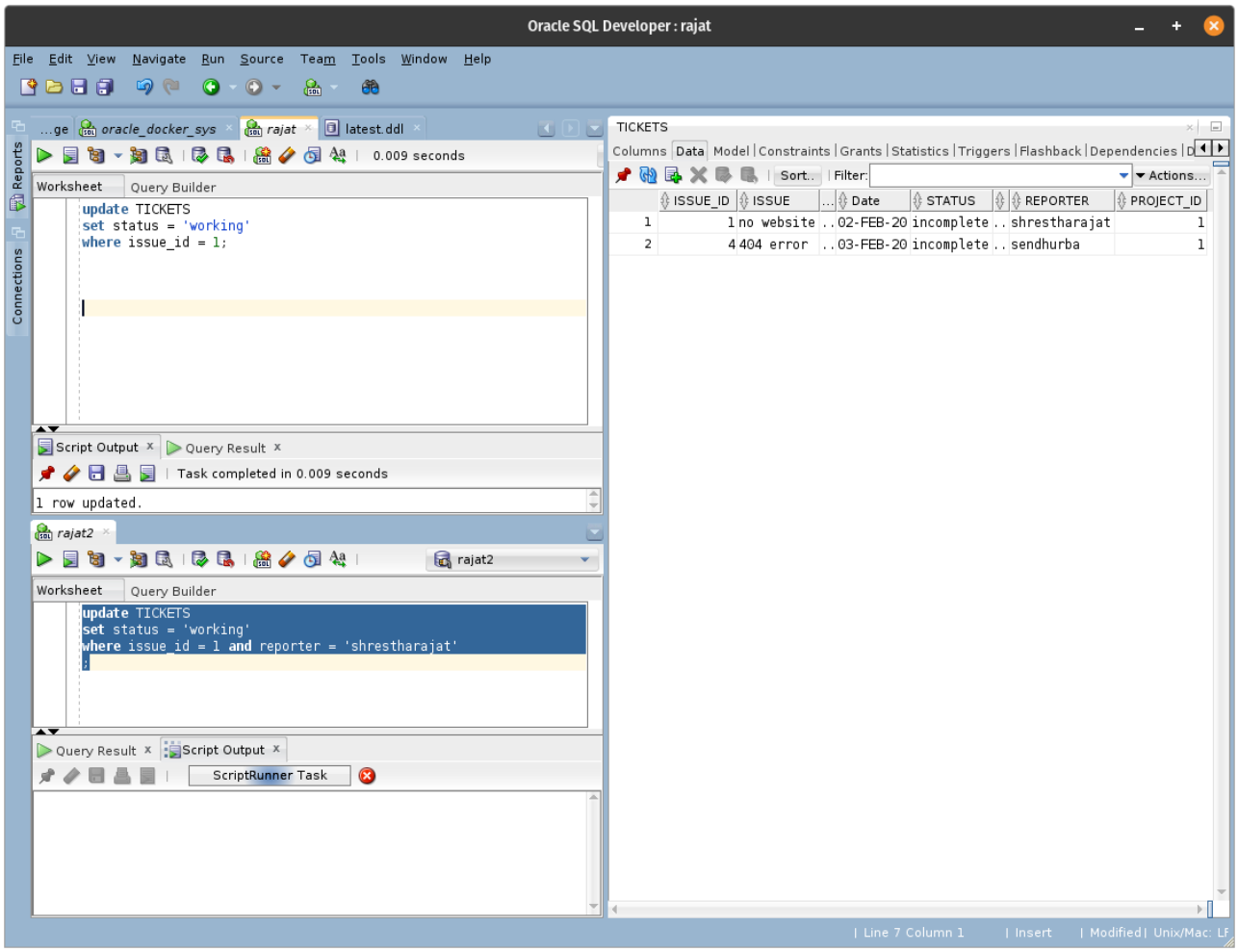



*Figure 8: Trying to Lock via Exclusive lock in a pre-existing exclusive lock.*

While an admin is writing something (exclusive lock) on the table:

- Nobody can read it, because it's still being written, and she's blocking your view => If an object is exclusively locked, shared locks cannot be obtained.
- Other teachers won't come up and start writing either, or the board becomes unreadable, and confuses students => If an object is exclusively locked, other exclusive locks cannot be obtained.

When the users are reading (shared locks) what is on the tables:

- They all can read what is on it, together => Multiple shared locks can co-exist.
- The teacher waits for them to finish reading before she clears the board to write more => If one or more shared locks already exist, exclusive locks cannot be obtained.

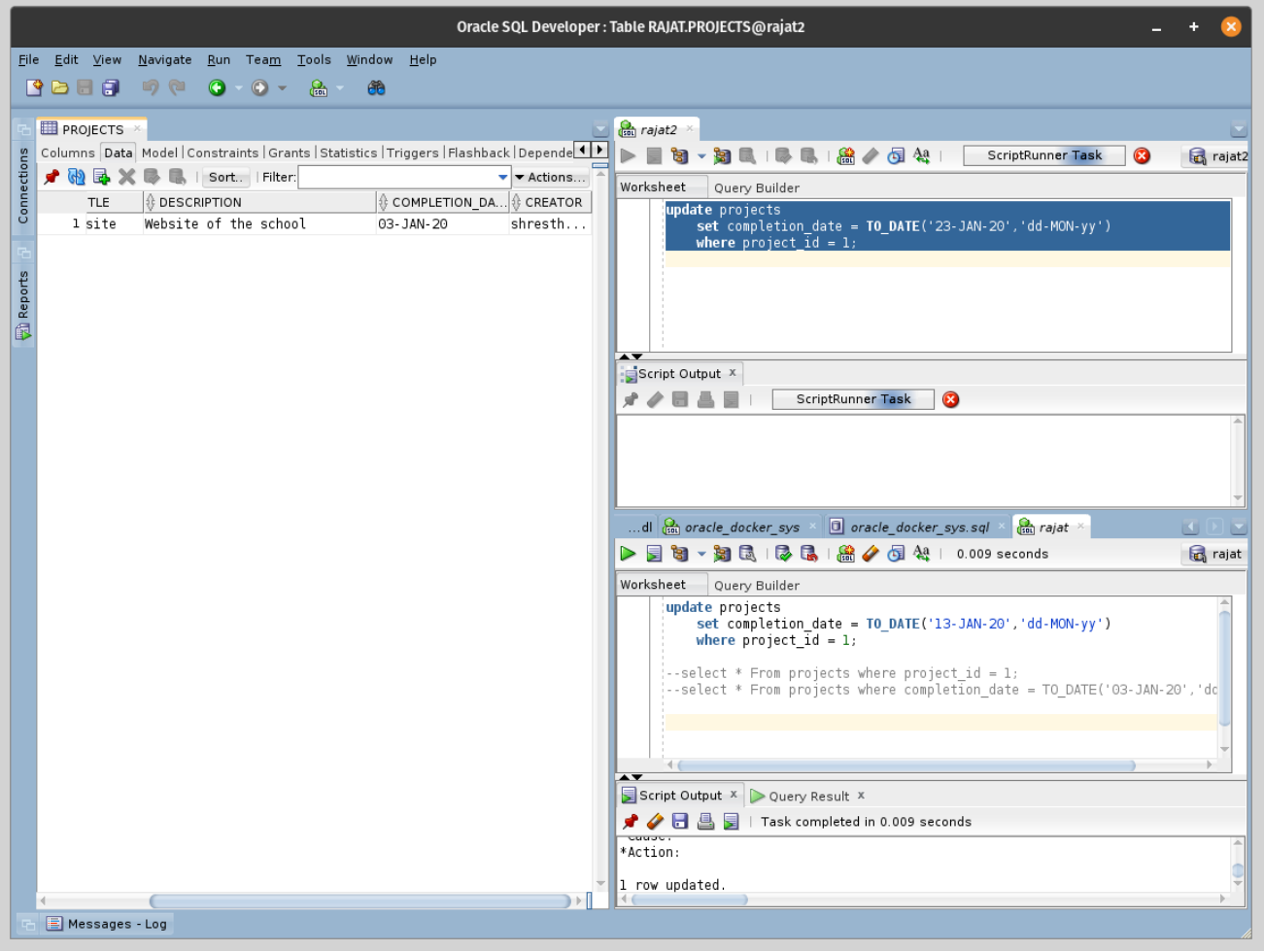



*Figure 9: Different locks in a same table*

# 9. References

Coronel, C. & Morris, S., 2018. *Database Systems: Design, Implementation, & Management.* s.l.:Cengage Learning.

Coronel, C. & Morris, S., 2018. *Database Systems: Design, Implementation, & Management.* s.l.:Cengage Learning.

Coronel, C. & Morris, S., 2018. *Database Systems: Design, Implementation, & Management.* s.l.:Cengage Learning.

Kreines, D., 2009. *Oracle Data Dictionary Pocket Reference.* 1st ed. Chicago: O'Reilly Media.

Nishadha, 2017. *Ultimate Guide to ER Diagrams ( Entity Relationship Diagrams ).* [Online]
Available at: https://creately.com/blog/diagrams/er-diagrams-tutorial/
[Accessed 28 January 2020].

Oracle, 2020. *Data Modeling with Oracle SQL Developer.* [Online]
Available at: https://www.oracle.com/in/database/technologies/appdev/datamodeler.html
[Accessed 28 January 2020].

Scottish Qualifications Authority, 2010. *Outcome 1:Fundamentals of Database Design.* [Online]
Available at: https://www.sqa.org.uk/e-learning/MDBS01CD/page_06.htm
[Accessed 12 1 2019].

Watt, A., 2012. *Database Design.* 2nd ed. Vancouver: BCcampus Open Textbook.