**Bugg Report Forum**

Team Members:

1) Poojan Patel

2) Sanyasi Naidu Pyla

3) Sai Supreeth Segu

4) Vamshi Goud Valdas

5) Sai Krushi Annam

## ***PROJECT PROPOSAL***

**Context of the Project:**

We would like to create an efficient database management system for a platform where users can report various types of glitches(bugs) they have observed while playing a game. It’s nothing but a service request page in which we store information of developers and users of various games related to various platforms (pc, Xbox, ps4, mobile etc.,). Users will report bugs and developers will release the updates to fix bugs.

**SCOPE of the Project:**

We will be storing information about the below fields in the database which we will be creating:

* Different games played across the entire world.
* The gaming platforms like mobile, Xbox, desktop etc.
* The bugs information reported by the users for the specific games.
* The developers who will be fixing the bugs reported by the users.
* There will be an intermediate team between users and developers who filter the “‘good bugs” from the bugs which occurs due to incompatible system configurations(user end issues).
* This intermediate team even prioritize the bugs, depending upon the damage it is inflicting to the user experience.
* Intermediate team will escalate the “good bugs”(software bugs) to the Developer teams , where the resolution process begin.

This scope is not limited as we get along with the project we may add few required fields as on demand. This would be an high end scope for our project.

**OBJECTIVES:**

Main aim of our project is to create a Database platform that is convenient and efficient to use in storing and retrieving information about different bugs reported by various users, developers and the resolution status of those bugs. Not only those our project also facilitates the processes of maintaining, manipulating and the flexibility in updating and deletion of data when required.

## ***PROJECT ENVIRONMENT***

Database client : MySQL Workbench version 8.0.15

Database server: AWS RDS server

Web-application server: Nginx

Database connection: MySQL Connector/j8.0(JDBC Driver)

Programming Languages: Python, Java script, SQL, HTML, Css.

***HIGH LEVEL REQUIREMENTS***

### **Initial user roles**

|  |  |
| --- | --- |
| **User Role** | **Description** |
| customer | Raises request |
| Escalation engineer | Prioritize the bug and forward it to the concerned Developer |
| Developer | Fix the bug and releases the updates |

**Initial user story descriptions**

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| User1 | As a user, I want to raise a request for the bug observed in game |
| User2 | As a user, I want to Download an update |
| User3 | As a user, I want to check the status of created requests. |
| User4 | As a developer, I want to release an update |
| User5 | As an escalation engineer, I want to update the status of the request. |
| User6 | As a developer, I want to change the status of the request |
| User7 | As a Subscriber, I want to close the request. |

***HIGH LEVEL CONCEPTUAL DESIGN***

Entities:

User

Escalation engineer

Developer

Request

Game

Update

Relationships:

User<creates> Request

User<downloads> update

Escalation engineer <Updates> Request

Developer <releases> update

**SPRINT 1:**

Story ID Story description User1 As a Subscriber, I want to raise a request for the bug observed in game User2 As an escalation engineer, I want to update the status of requests. User3 As an escalation engineer, I want to prioritize the requests.

User4

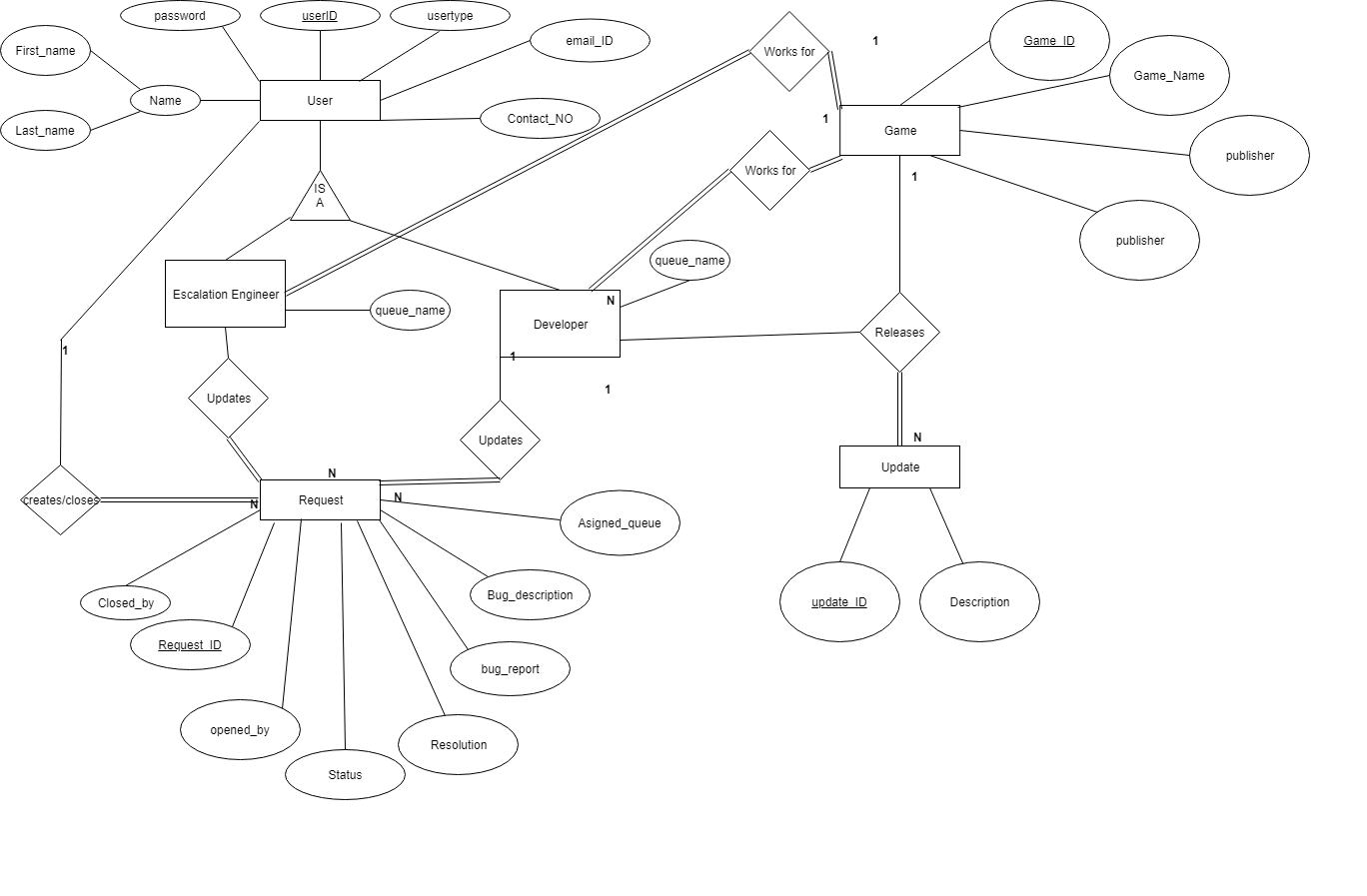
As an escalation engineer, I want to create master Request and escalate to developer team.

User5

As an escalation engineer, i want to take ownership of the request User6 As a developer, I want to release updates.

User7

As a developer engineer, i want to take ownership of the request User8 As a developer, I want to update the status of the request. User9 As a Subscriber, I want to Download an update. User10 As a Subscriber, I want to check the status of created requests. User11 As a Subscriber, I want to close the request.

****

Conceptual Design:

Entity: USER(**Parent**)

**Attributes:**

user\_ID[Simple/Atomic]

Password[Simple/Atomic]

user\_type[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Contact\_No[Simple/Atomic]

Primary key justification: Every user need to have unique **user\_ID**, so it is a good primary key.

Entity: **Escalation engineer/Developer -**inherits attributes from User

Game\_ID[Simple/Atomic]

Queue\_name[Simple/Atomic]

Employee\_ID

Primary key justification: Every Escalation engineer has unique **user\_ID**, so it is a good primary key.

Entity: **Game**

**Attributes**:

Game\_ID[Simple/Atomic]

Game\_Name[Simple/Atomic]

publisher

Platform[Simple/Atomic]

Entity: **Request**

**Attributes:**

Request\_ID[Simple/Atomic]

Opened\_by[single\_valued]

Assigned\_queue[Simple/Atomic]

Status[Simple/Atomic]

Bug\_type[Simple/Atomic]

Bug\_description[Simple/Atomic]

Resolution\_description[Simple/Atomic]

Closed\_by[Simple/Atomic]

Entity: **Update**

**Attributes:**

update\_ID[single valued]

released\_by[Simple/Atomic]

Game\_ID[Simple/Atomic]

Description[Simple/Atomic]

**Relationships:**

1. Escalation\_engineer/Developer <works for> Game.

Relationship: many - to - many

Participation: engineer/developer and the game has total participation

In first sprint, we have assumed that each each developer/escalation engineer can work for only one game. But in this sprint we have thought about a case where, a group of developers working for a publisher can work for few games which are released under the same publisher.

Escalation engineer teams are third party teams which supports few games.

2. User <raises> request

Relation : one to many

Participation: User has partial participation

Request has total participation in this relationship

Since,a request won’t exist if not raised.

3. Developer <releases> update

Relation: one to many

Participation: Developer has partial participation

Update has total participation in this relationship

Here, we have assumed that a group of developers can work on an update but the update can be released by any one of them.Hence, making it a one to many relationship.

4. Escalation\_engineer/developer <Manages> request

Relation : one to many.

Participation: Escalation engineer has partial participation

Request has partial participation in this relationship

A request can be closed only once and a user can close many requests.

**Logical Design:**

Table: **USER** primary key(user\_ID)

user\_ID

Password

user\_type

First\_name

Last\_name

Middle\_name

email\_ID

Contact\_No

Table: **Escalation engineer/Developer** primary key(Employee\_ID)

Employee\_ID[foreign key; references id of User table]

Game\_ID[foreign key; references id of Game table]

Queue

Table: **Game** primary key**(**Game\_ID**)**

Game\_ID

Game\_name

publisher

Platform

Table: **Request**

primary key(Request\_ID,Opened\_by,Owner\_Ship,Game\_ID,Closed\_by)

Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Game\_ID[foreign key; references id of Game table]

Assigned\_queue

Status

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

Table: **Update** primary key(update\_ID,released\_by,Game\_ID)

update\_ID[single valued]

released\_by[foreign key; references id of Escalation engineer table]

Game\_ID[foreign key; references id of Game table]

Description

**Sprint 2:**

We have found out that it’s not efficient to keep users, escalation engineers and developers under one table **USER**.So, we have created separate tables in this sprint.

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| User1 | As a user, I want to raise a request for the bug observed in game |
| User2 | As a user, I want to Download an update. |
| User3 | As a user, I want to check the status of created requests. |
| user4 | As an escalation engineer i want to take ownership of a request. |
| User5 | As an Escalation engineer, I want to prioritize the request |
| User6 | As a developer, I want to take ownership of the request. |
| User7 | As a developer, I want to change the status of the request |
| User8 | As a User/developer/escalation engineer, I want to close the request. |
| User9 | As an escalation engineer i want to assign request to Developer team. |
| User10 | As a Developer, i want to release an update |
| User11 | As a developer /escalation engineer I work for a developer team/escalation engineer team. |

Conceptual Design:

Entity: **USER**

**Attributes:**

user\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every user need to have unique **user\_ID**, so it is a good primary key.

Entity: **Escalation Engineer**

**Attributes:**

EscalationEngineer\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every Escalation Engineer need to have unique EscalationEngineer\_ID, so it is a good primary key.

Entity : **Developer**

**Attributes:**

Developer\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every Developer need to have unique Developer\_ID, so it is a good primary key.

Entity : **Escalation Team**

**Attributes:**

Escalation\_Team\_ID[Simple/Atomic]

Game\_ID[Simple/Atomic]

Total\_memebers[Simple/Atomic]

Primary key justification: Escalation\_Team\_ID uniquely define the Escalation Teams entity , so it is a good primary key.

Entity : **Developers Team**

**Attributes:**

Developer\_Team\_ID[Simple/Atomic]

Game\_ID[Simple/Atomic]

Total\_memebers[Simple/Atomic]

Primary key justification: Developer\_Team\_ID uniquely define the Developer Teams entity , so it is a good primary key.

Entity : **EscalationEngineer\_Works\_for**

**Attributes:**

EscalationEngineer\_ID

Team\_ID

Working\_since

Primary key justification: EscalationEngineer\_ID and Team\_ID together uniquely define the EscalationEngineer\_Works\_for entity, so both together is a good primary key.

Entity : **Developer\_Works\_for**

**Attributes:**

Developer\_ID

Team\_ID

Working\_since

Primary key justification: Developer\_ID and Team\_ID together uniquely define the Developer\_Works\_for entity , so both together is a good primary key.

Entity : **GAME**

**Attributes:**

Game\_ID

Name

Platform

Publisher

Primary key justification: Every GAME need to have unique Game\_ID, so it is a good primary key.

Entity : **Games\_Played\_by\_User**

**Attributes:**

User\_ID

Game\_ID

Playing\_since

Primary key justification: User\_ID and Game\_ID together uniquely define the Games\_Played\_by\_User entity , so both together is a good primary key.

Entity : **Request**

**Attributes: Primary Key(**Request\_ID)

Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Assigned\_Team\_ID[foreign key; references id of Escalations team table]

Status

priority

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

Primary key justification: Every REQUEST need to have unique Request\_ID, so it is a good primary key.

Entity: **Bug\_Update**

**Attributes: Primary key(**update\_ID**)**

update\_ID[single valued]

released\_by[foreign key; references id of Escalation engineer table]

Game\_ID[foreign key; references id of Game table]

Update\_version

Description

Primary key justification: Every UPDATE need to have unique update\_ID, so it is a good primary key.

**RelationShips:**

1. Escalation\_engineer/Developer <works for> Escalation team/developer team.

Relationship: many - to - many

Participation: engineer/developer and the respective teams has total participation

In first sprint, we have assumed that each each developer/escalation engineer can work for only one game. But in this sprint we have thought about a case where, a group of developers working for a publisher can work for few games which are released under the same publisher.

So we have added new tables Developer team and Escalation team which supports only one game. We have mapped escalation engineer/developer to many teams.

2. User <raises> request

Relation : one to many

Participation: User has partial participation

Request has total participation in this relationship

Since,a request won’t exist if not raised.

3. Developer <releases>bug\_ update

Relation: one to many

Participation: Developer has partial participation

Bug\_update has total participation in this relationship

Here, we have assumed that a group of developers can work on an update but the update can be released by any one of them.Hence, making it a one to many relationship.

4. Escalation\_engineer/developer <Updates> request

Relation : one to many.

Participation: Escalation engineer has partial participation

Request has total participation in this relationship

A request can be closed only once and a user can close many requests.

**LOGICAL DESIGN:**

Table: **USER primary key(**user\_ID)

user\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Highest\_normal form- 4NF

Table:**Escalation Engineer Primary Key(**EscalationEngineer\_ID**)**

EscalationEngineer\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Table:**Developer Primary Key(**Developer\_ID)

Developer\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Table: **Escalation Teams Primary Key(**Escalation\_Team\_ID)

Escalation\_Team\_ID

Game\_ID[foreign key, references to Game table]

Total\_members

Highest Normal Form : 4NF

Table: **Developers Teams Primary Key(**Developer\_Team\_ID)

Developer\_Team\_ID

Game\_ID[foreign key, references to Game table]

Total\_members

Highest Normal Form : 4NF

Table:**EscalationEngineer\_Works\_for**

**Primary Key(EscalationEngineer\_ID,Team\_ID)**

EscalationEngineer\_ID[foreign key, references to **Escalation Engineer** table]

Team\_ID[foreign key, references to **Escalation Teams** table]

Working\_since

Highest Normal Form : 4NF

Table:**Developer\_Works\_for** Primary Key(Developer\_ID,Team\_ID)

Developer\_ID[foreign key, references to **Developer** table]

Team\_ID[foreign key, references to **Developer Teams** table]

Working\_since

Highest Normal Form : 4NF

Table:**GAME Primary Key(**Game\_ID**)**

Game\_ID

Name

Platform

Publisher

Highest Normal Form : 2NF

Name → {publisher}, due to presence of transitive dependency, the highest normal form achieved is 2NF. We can normalize it more but it will result in more unnecessary tables and so will be joints.

Table:**Games\_Played\_by\_User Primary Key(**User\_ID, Game\_ID**)**

User\_ID[foreign key; references id of user table]

Game\_ID[foreign key; references id of Game table]

Playing\_since

Highest Normal Form - 4NF

Table:  **Request Primary Key(**Request\_ID)

Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Assigned\_Team\_ID[foreign key; references id of Escalation engineer teams table]

Status

priority

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

* Here anyone of the 3(Escalation Engineer, Developer, User) can close the Request, so that’s the reason the “Closed\_by” attribute has foreign key references to 3 tables.

Highest Normal Form - 4NF

Table: **Bug\_Update Primary key(**update\_ID**)**

update\_ID[single valued]

released\_by[foreign key; references id of Developer table][Developer ID]

Game\_ID\_for\_Update[foreign key; references id of Game table]

Update\_version

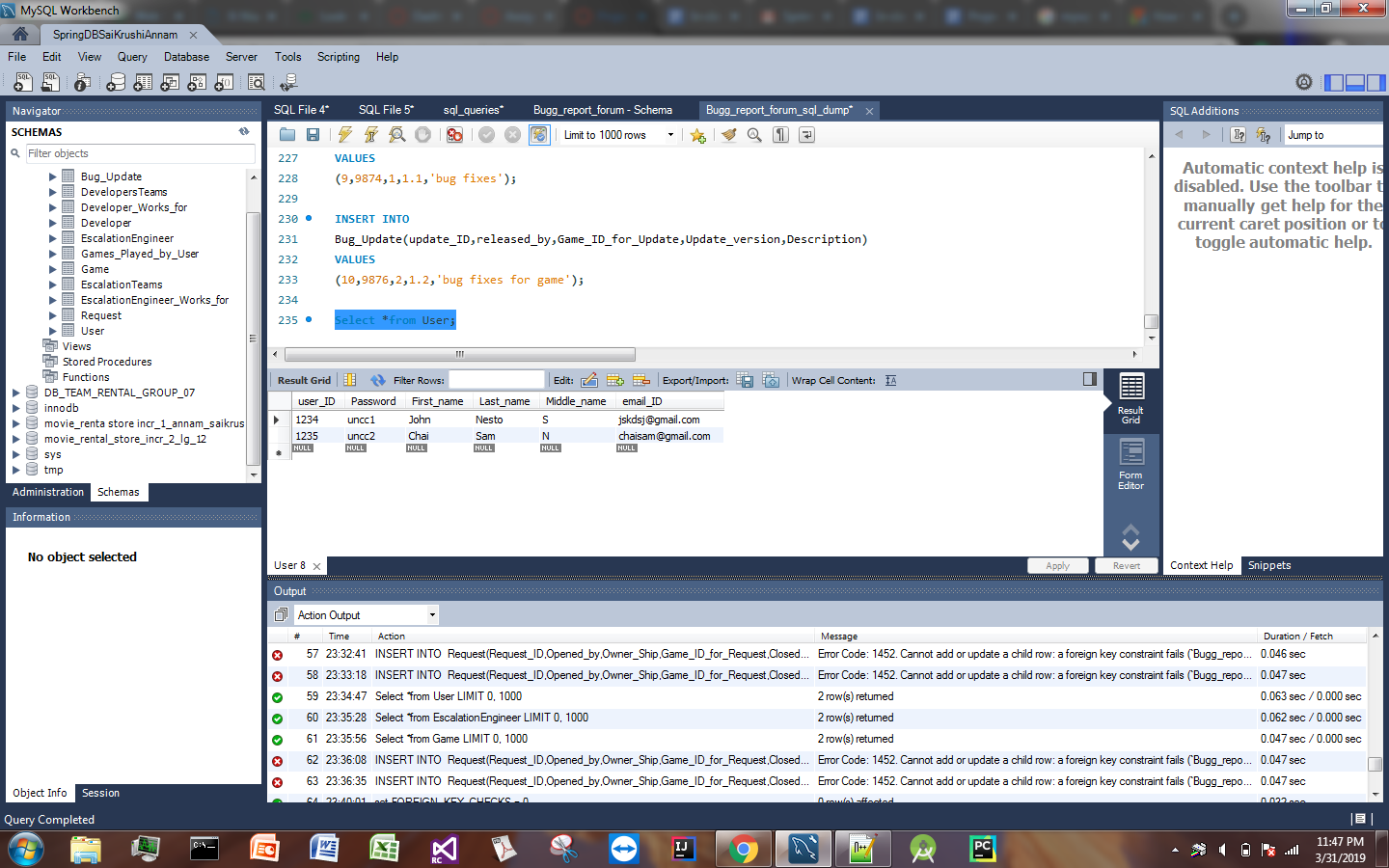
Description

Bug\_Request[foreign key; references id of Request table][Request\_ID]

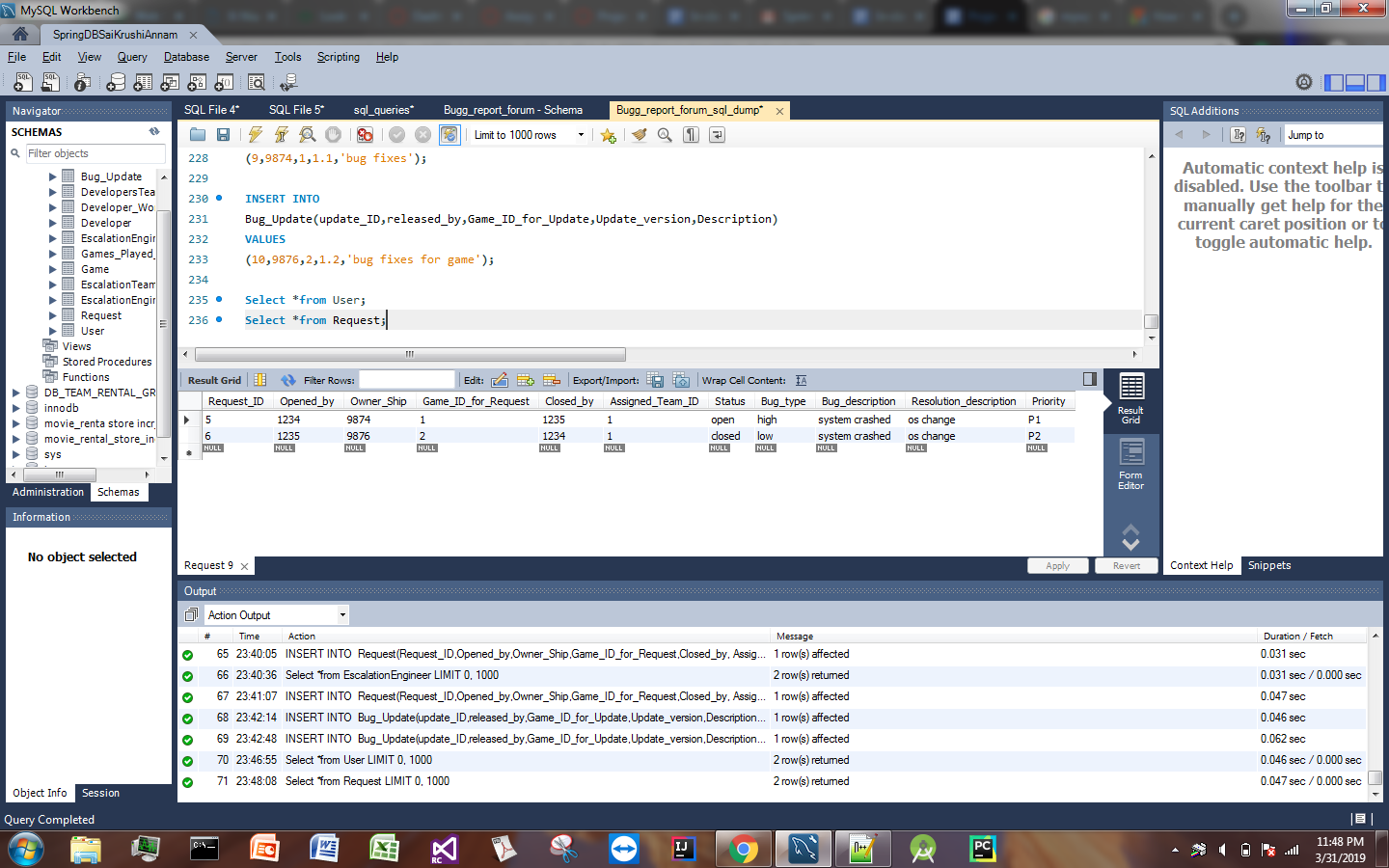
Highest Normal Form - 4NF

**SQL SCREENSHOTS**:

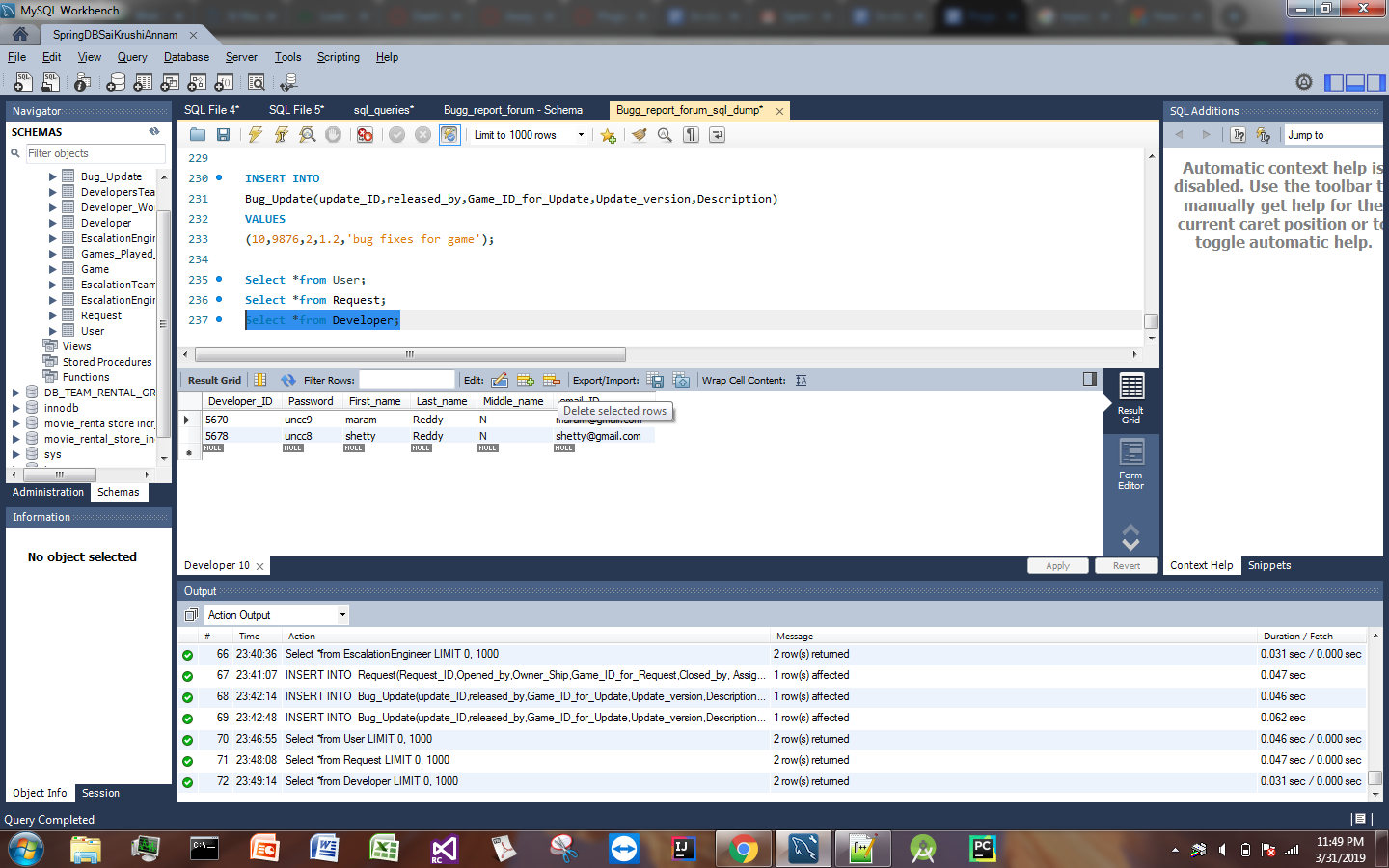
Select \*from User;



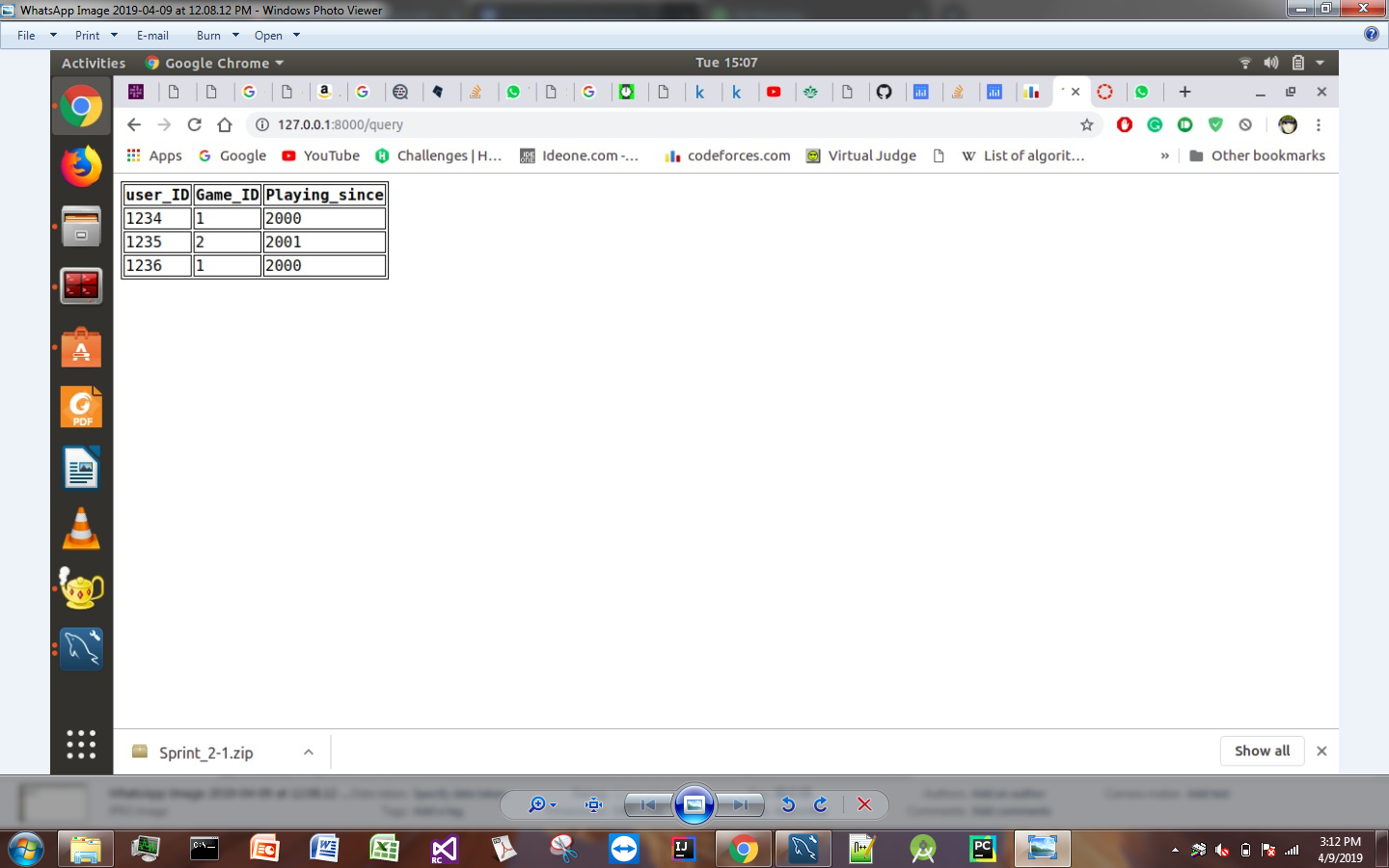
Select \*from Request;

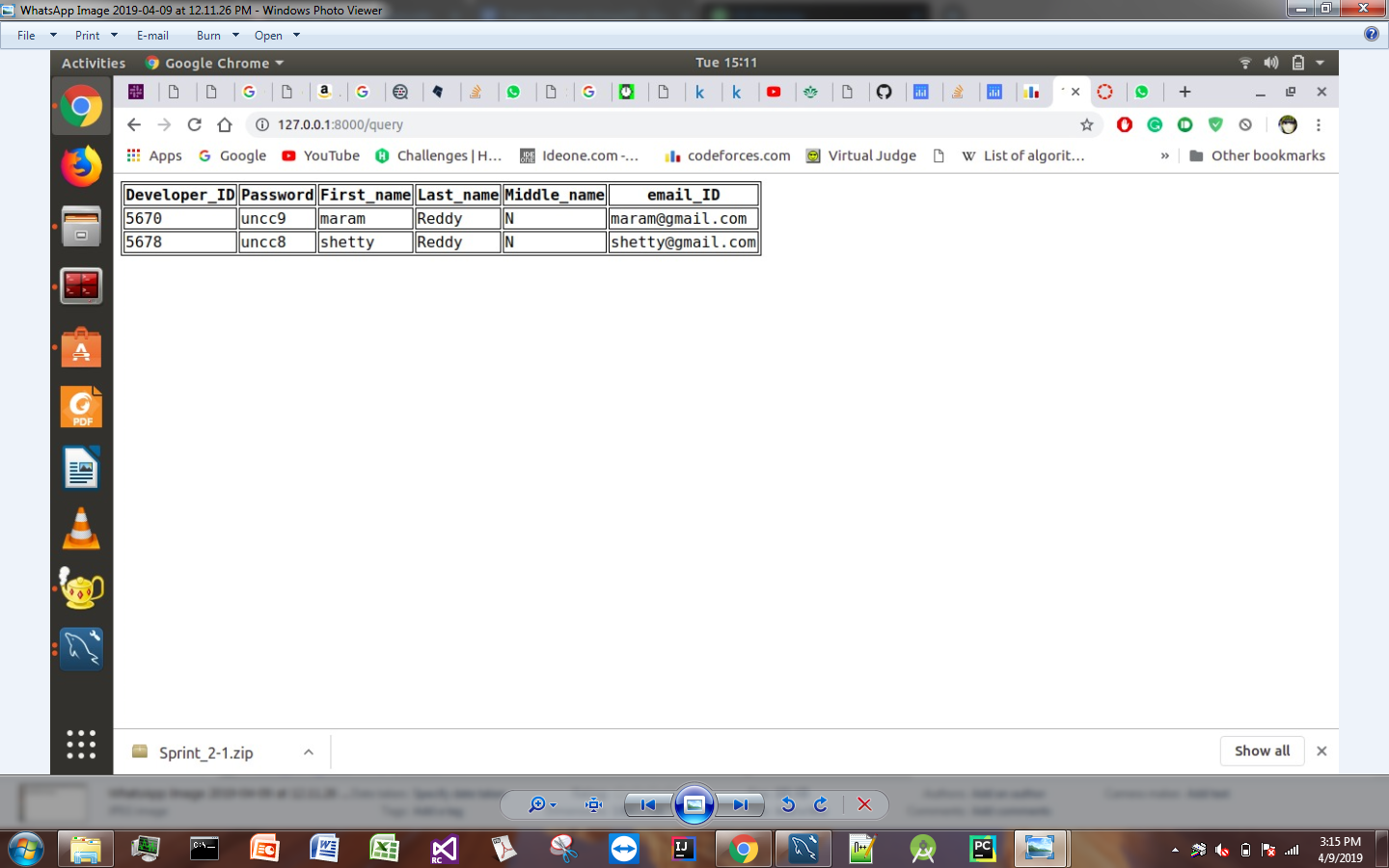


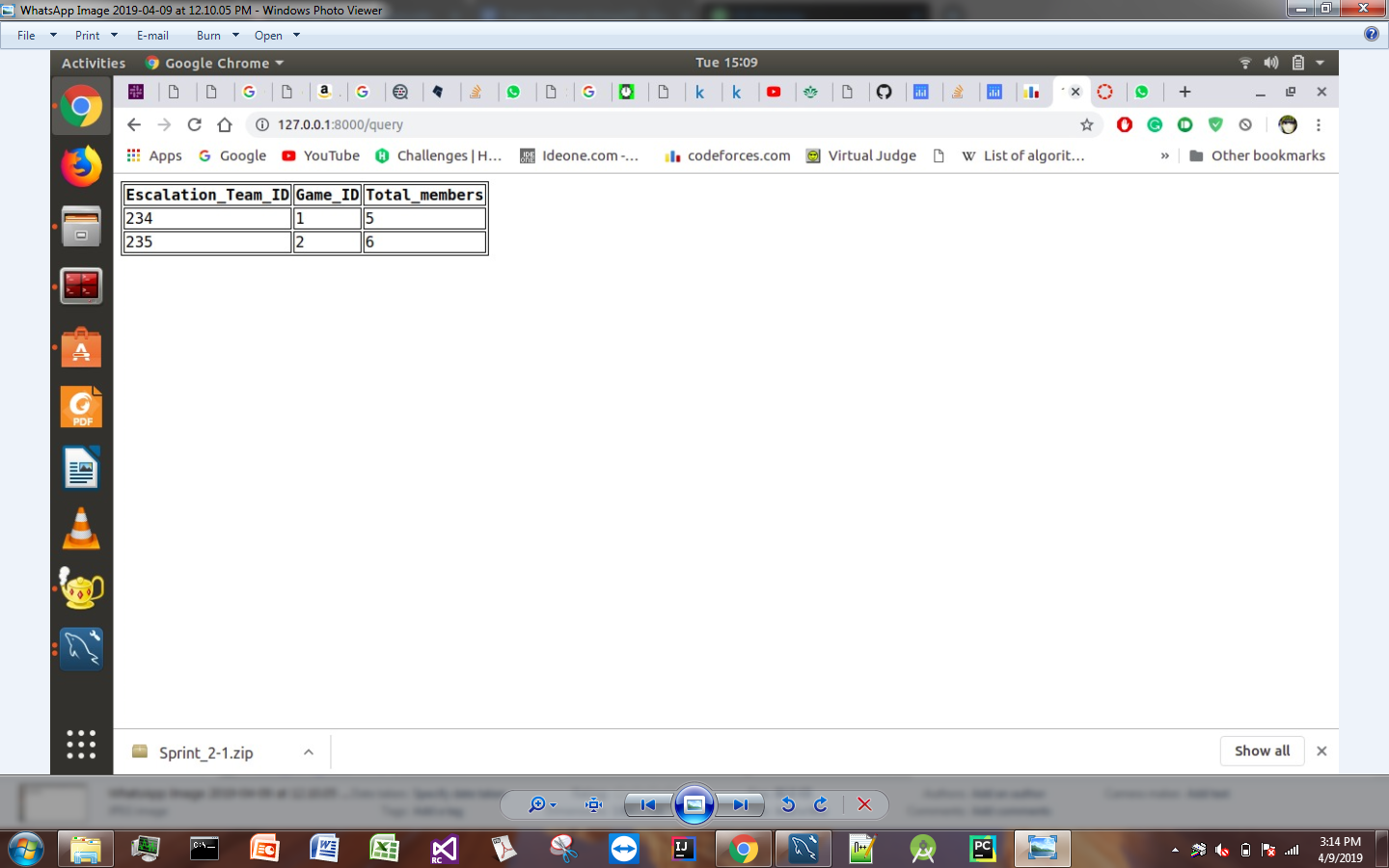
Select \*from Developer;



UI SCREENSHOTS:







**SPRINT 3**

**User stories:**

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| User12 | As an escalation engineer i want to create a master request on all the same type of requests. |
| User13 | As a developer i will manage the master request. |

Conceptual Design:

Entity: **USER**

**Attributes:**

user\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every user need to have unique **user\_ID**, so it is a good primary key.

Entity: **Escalation Engineer**

**Attributes:**

EscalationEngineer\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every Escalation Engineer need to have unique EscalationEngineer\_ID, so it is a good primary key.

Entity : **Developer**

**Attributes:**

Developer\_ID[Simple/Atomic]

Password[Simple/Atomic]

Name[composite]

First\_name

Last\_name

Middle\_name

email\_ID[Simple/Atomic]

Primary key justification: Every Developer need to have unique Developer\_ID, so it is a good primary key.

Entity : **Escalation Team**

**Attributes:**

Escalation\_Team\_ID[Simple/Atomic]

Game\_ID[Simple/Atomic]

Total\_memebers[Simple/Atomic]

Primary key justification: Escalation\_Team\_ID uniquely define the Escalation Teams entity , so it is a good primary key.

Entity : **Developers Team**

**Attributes:**

Developer\_Team\_ID[Simple/Atomic]

Game\_ID[Simple/Atomic]

Total\_memebers[Simple/Atomic]

Primary key justification: Developer\_Team\_ID uniquely define the Developer Teams entity , so it is a good primary key.

Entity : **EscalationEngineer\_Works\_for**

**Attributes:**

EscalationEngineer\_ID

Team\_ID

Working\_since

Primary key justification: EscalationEngineer\_ID and Team\_ID together uniquely define the EscalationEngineer\_Works\_for entity, so both together is a good primary key.

Entity : **Developer\_Works\_for**

**Attributes:**

Developer\_ID

Team\_ID

Working\_since

Primary key justification: Developer\_ID and Team\_ID together uniquely define the Developer\_Works\_for entity , so both together is a good primary key.

Entity : **GAME**

**Attributes:**

Game\_ID

Name

Platform

Publisher

Primary key justification: Every GAME need to have unique Game\_ID, so it is a good primary key.

Entity : **Games\_Played\_by\_User**

**Attributes:**

User\_ID

Game\_ID

Playing\_since

Primary key justification: User\_ID and Game\_ID together uniquely define the Games\_Played\_by\_User entity , so both together is a good primary key.

Entity : **Request**

**Attributes: Primary Key(**Request\_ID)

Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Assigned\_Team\_ID[foreign key; references id of Escalations team table]

Status

priority

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

Primary key justification: Every REQUEST need to have unique Request\_ID, so it is a good primary key.

Entity: **Bug\_Update**

**Attributes: Primary key(**update\_ID**)**

update\_ID[single valued]

released\_by[foreign key; references id of Escalation engineer table]

Game\_ID[foreign key; references id of Game table]

Update\_version

Description

Primary key justification: Every UPDATE need to have unique update\_ID, so it is a good primary key.

**RelationShips:**

1. Escalation\_engineer/Developer <works for> Escalation team/developer team.

Relationship: many - to - many

Participation: engineer/developer and the respective teams has total participation

In first sprint, we have assumed that each each developer/escalation engineer can work for only one game. But in this sprint we have thought about a case where, a group of developers working for a publisher can work for few games which are released under the same publisher.

So we have added new tables Developer team and Escalation team which supports only one game. We have mapped escalation engineer/developer to many teams.

2. User <raises> request

Relation : one to many

Participation: User has partial participation

Request has total participation in this relationship

Since,a request won’t exist if not raised.

3. Developer <releases>bug\_ update

Relation: one to many

Participation: Developer has partial participation

Bug\_update has total participation in this relationship

Here, we have assumed that a group of developers can work on an update but the update can be released by any one of them.Hence, making it a one to many relationship.

4. Escalation\_engineer/developer <Updates> request

Relation : one to many.

Participation: Escalation engineer has partial participation

Request has total participation in this relationship

A request can be closed only once and a user can close many requests.

**Modifications in conceptual design:**

* We have added a new table ‘Master\_Request’. This Master\_Request can be created by an escalation engineer on multiple Requests of same type
* We have linked this master request table with the update table

Entity: **Master\_Request**

**Attributes:**

Master\_Request\_ID[Simple/Atomic]

Opened\_by[single\_valued]

Assigned\_queue[Simple/Atomic]

Update\_ID

Closed\_by[Simple/Atomic]

**Relationships:**

Master\_Request < created on> Request

Cardinality: one to many

Participation: request has partial participation. [few requests doesn’t require Master\_Request’s ]

Master\_Request has total participation.

Update <included in> Master\_request

Cardinality: one to one

Participation: Update has partial participation

Master\_request has partial participation

**LOGICAL DESIGN:**

Table: **USER primary key(**user\_ID)

user\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Highest\_normal form- 4NF

Table:**Escalation Engineer Primary Key(**EscalationEngineer\_ID**)**

EscalationEngineer\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Table:**Developer Primary Key(**Developer\_ID)

Developer\_ID

Password

First\_name

Last\_name

Middle\_name

email\_ID

Table: **Escalation Teams Primary Key(**Escalation\_Team\_ID)

Escalation\_Team\_ID

Game\_ID[foreign key, references to Game table]

Total\_members

Highest Normal Form : 4NF

Table: **Developers Teams Primary Key(**Developer\_Team\_ID)

Developer\_Team\_ID

Game\_ID[foreign key, references to Game table]

Total\_members

Highest Normal Form : 4NF

Table:**EscalationEngineer\_Works\_for**

**Primary Key(EscalationEngineer\_ID,Team\_ID)**

EscalationEngineer\_ID[foreign key, references to **Escalation Engineer** table]

Team\_ID[foreign key, references to **Escalation Teams** table]

Working\_since

Highest Normal Form : 4NF

Table:**Developer\_Works\_for** Primary Key(Developer\_ID,Team\_ID)

Developer\_ID[foreign key, references to **Developer** table]

Team\_ID[foreign key, references to **Developer Teams** table]

Working\_since

Highest Normal Form : 4NF

Table:**GAME Primary Key(**Game\_ID**)**

Game\_ID

Name

Platform

Publisher

Highest Normal Form : 2NF

Name → {publisher}, due to presence of transitive dependency, the highest normal form achieved is 2NF. We can normalize it more but it will result in more unnecessary tables and so will be joints.

Table:**Games\_Played\_by\_User Primary Key(**User\_ID, Game\_ID**)**

User\_ID[foreign key; references id of user table]

Game\_ID[foreign key; references id of Game table]

Playing\_since

Highest Normal Form - 4NF

Table:  **Request Primary Key(**Request\_ID)

Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Assigned\_Team\_ID[foreign key; references id of Escalation engineer teams table]

Status

priority

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

* Here anyone of the 3(Escalation Engineer, Developer, User) can close the Request, so that’s the reason the “Closed\_by” attribute has foreign key references to 3 tables.

Highest Normal Form - 4NF

Table: **Bug\_Update Primary key(**update\_ID**)**

update\_ID[single valued]

released\_by[foreign key; references id of Developer table][Developer ID]

Game\_ID\_for\_Update[foreign key; references id of Game table]

Update\_version

Description

Bug\_Request[foreign key; references id of Request table][Request\_ID]

Highest Normal Form - 4NF

**Updated Logical Design:**

Table:  **Request Primary Key(**Request\_ID)

Request\_ID

Master\_Request\_ID[Foreign key; references to Master\_Request\_ID of Master\_Request table]

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Escalation engineer table]

Assigned\_Team\_ID[foreign key; references id of Escalation engineer teams table]

Status

priority

Bug\_type

Bug\_description

Resolution\_description

Closed\_by[foreign key; references id of Escalation engineer table]

Table:  **Master\_Request Primary Key(**Master\_Request\_ID)

Master\_Request\_ID

Opened\_by[single\_valued][foreign key; references id of User table]

Owner\_Ship[foreign key; references id of Developer table]

Assigned\_Team\_ID[foreign key; references id of Developer Teams table]

Resolution\_description

Closed\_by[foreign key; references id of Developertable]

**Indexes:**

1. **User Table:**

**Index 1**: Primary key is user\_ID - Default INDEX

INDEX\_NAME - user\_ID (Primary Key)

<type - clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on FirstName and LastName of the User

INDEX\_NAME - User\_full\_name

<type - Non-Clustered>

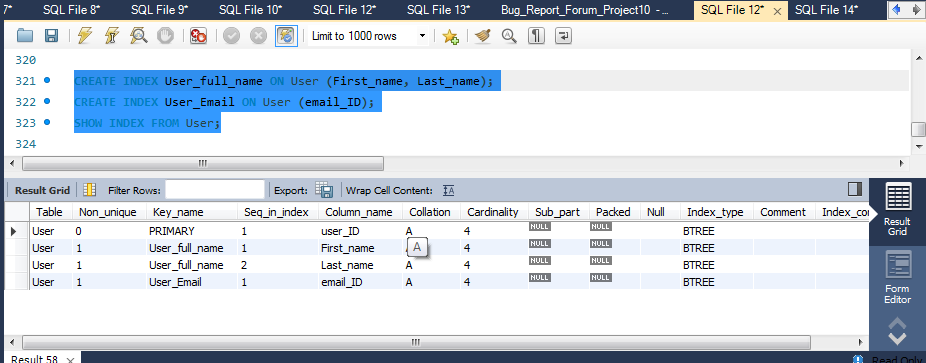
**Justification**: This gives easy search on the UserName as this avoids Table Scan when we want to search based on the name.

**Index 3**: Index on Email-ID of the User

INDEX\_NAME - User\_Email

<type - Non-Clustered>

**Justification**: The email\_id of user also may be unique and if we create INDEX on email this gives us an easy search to identify the user.



1. **Developer Table:**

**Index 1**: Primary key is Developer\_ID - Default INDEX

INDEX\_NAME - Developer\_ID

<type - clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on FirstName and LastName of the Developer

INDEX\_NAME - Developer\_full\_name

<type - Non-Clustered>

**Justification**: This gives easy search based on the Developer Name as this avoids Table Scan when we want to search based on the name.

**Index 3**: Index on Email-ID of the Developer

INDEX\_NAME - Developer\_Email\_ID

<type - Non-Clustered>

**Justification**: The email\_id of developer also may be unique and if we create INDEX on email this gives us an easy search to identify the Developer.

**Index 3**: Index on Salary\_per\_annum of the Developer

INDEX\_NAME - Developer\_Salary

<type - Non-Clustered>

**Justification**: When we want to search for Developers on some particular salary range then this indexing will be helpful and gets the faster results. This Index arranges the Salary in Ascending Order.



3. **EscalationEngineer Table:**

**Index 1**: Primary key is EscalationEngineer\_ID - Default INDEX

INDEX\_NAME - EscalationEngineer\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on FirstName and LastName of the EscalationEngineer

INDEX\_NAME - EscalationEngineer\_full\_name

<type - Non-Clustered>

**Justification**: This gives easy search based on the EscalationEngineer Name as this avoids Table Scan when we want to search based on the name.

**Index 3**: Index on Email-ID of the EscalationEngineer

INDEX\_NAME - EscalationEngineer\_Email\_ID

<type - Non-Clustered>

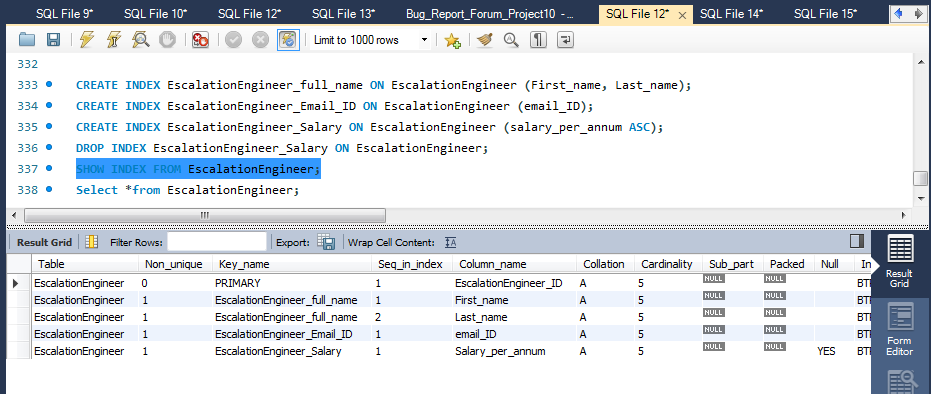
**Justification**: The email\_id of Escalation engineer also may be unique and if we create INDEX on email this gives us an easy search to identify the EscalationEngineer.

**Index 3**: Index on Salary\_per\_annum of the EscalationEngineer

INDEX\_NAME - Developer\_Salary

<type - Non-Clustered>

**Justification**: When we want to search for EscalationEngineeron some particular salary range then this indexing will be helpful and gets the faster results. This Index arranges the Salary in Ascending Order.



4. **Game Table:**

**Index 1**: Primary key is Game\_ID - Default INDEX

INDEX\_NAME - Game\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Game Name

INDEX\_NAME - Game\_Name

<type - Non-Clustered>

**Justification**: This gives easy search based on the Game

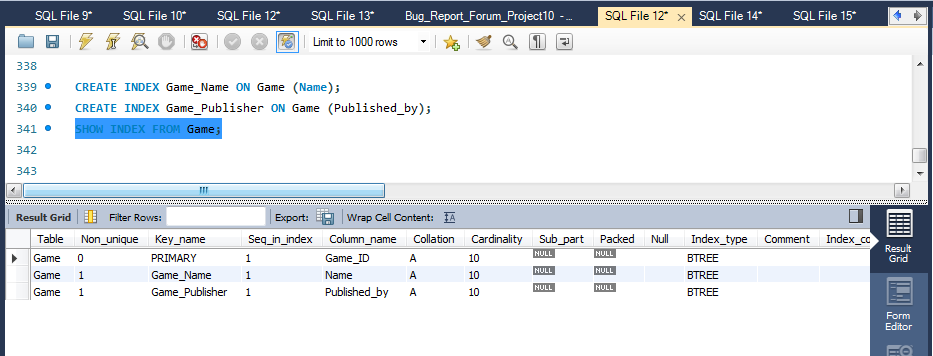
Name as this avoids Table Scan when we want to search based on the name.

**Index 3**: Index on Publishers of the Game

INDEX\_NAME - Game\_Publisher

<type - Non-Clustered>

**Justification**: When we want to search what games are released by some particular publisher then this Index is helpful.



4. **DeveloperTeams Table:**

**Index 1**: Primary key is Developer\_Team\_ID - Default INDEX

INDEX\_NAME - Developer\_Team\_ID

<type - Clustered>

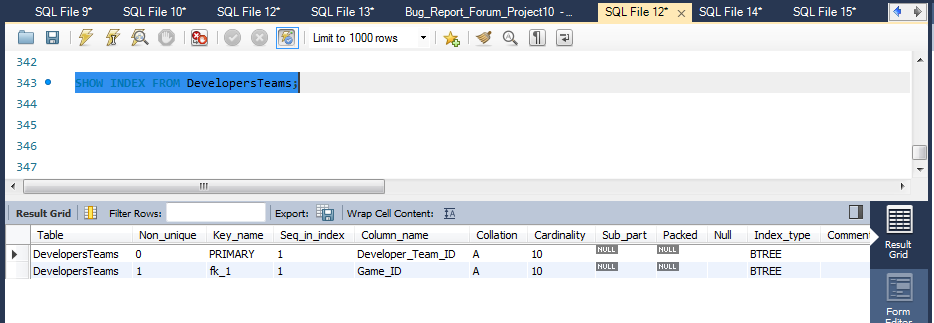
**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Game\_ID

INDEX\_NAME - Game\_ID

<type - Non-Clustered>

**Justification**: Game\_ID is a Foreign Key which is Not Unique. This Index is used for faster searchers based on game ID like when we want to search which teams are working for particular game.



5. **EscalationTeams Table:**

**Index 1**: Primary key is Escalation\_Team\_ID - Default INDEX

INDEX\_NAME - Game\_ID

<type - Clustered>

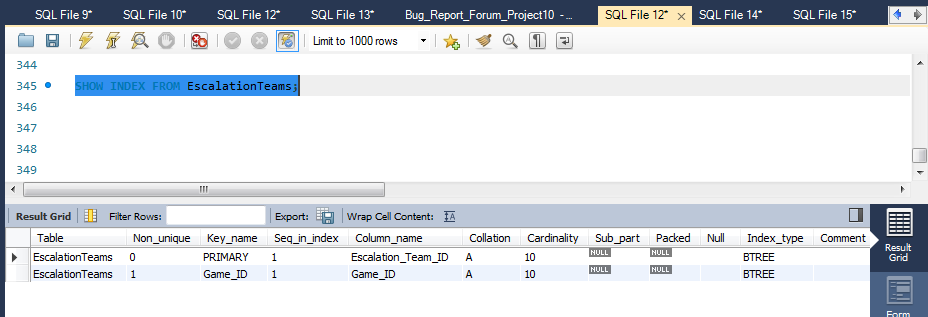
**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Game\_ID

INDEX\_NAME - Game\_ID

<type - Non-Clustered>

**Justification**: Game\_ID is a Foreign Key which is Not Unique. This Index is used for faster searchers based on game ID like when we want to search which Escalation Engineer teams are monitoring the request for a particular game.



6. **Developer\_Works\_for Table:**

**Index 1**: Primary key is Developer\_ID - Default INDEX

INDEX\_NAME - Developer\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Team\_ID

INDEX\_NAME - Team\_ID

<type - Clustered>

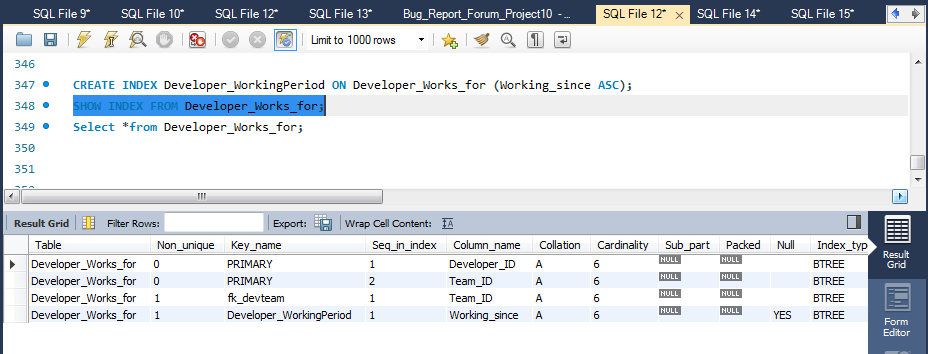
**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 3**: Index on Working\_since

INDEX\_NAME - Developer\_WorkingPeriod

<type - Non-Clustered>

**Justification**: When we want to make search regarding the developers working for some particular period then this INDEX is helpful. This Index arranges the Salary in Ascending Order.



7. **EscalationEngineer\_Works\_for Table:**

**Index 1**: Primary key is EscalationEngineer\_ID - Default INDEX

INDEX\_NAME - EscalationEngineer\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Team\_ID

INDEX\_NAME - Team\_ID

<type - Clustered>

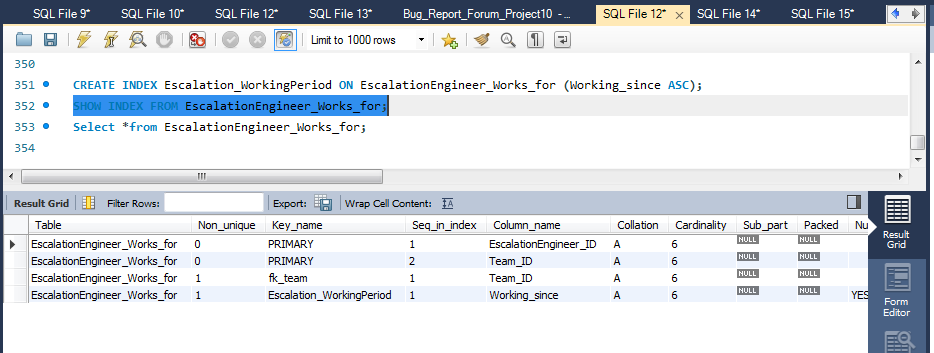
**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 3**: Index on Working\_since

INDEX\_NAME - Escalation\_WorkingPeriod

<type - Non-Clustered>

**Justification**: When we want to make search regarding the Escalation Engineers working for some particular period then this INDEX is helpful. This Index arranges the Salary in Ascending Order.



8. **Games\_Played\_by\_User Table:**

**Index 1**: Primary key is user\_ID - Default INDEX

INDEX\_NAME - user\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Game\_ID

INDEX\_NAME - Game\_ID

<type - Clustered>

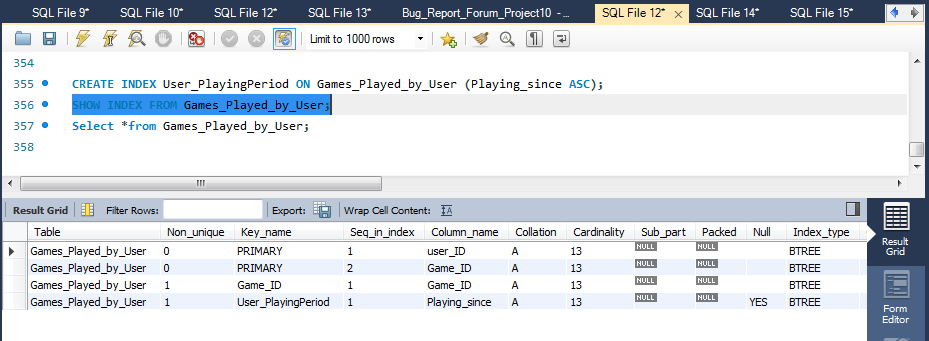
**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Playing\_since

INDEX\_NAME - User\_PlayingPeriod

<type - Non-Clustered>

**Justification**: To know from which year or time the user is playing a particular game, basically this index is helpful to get information on games which are played for some particular period. So to such information this index is created. This index arranges the time in ascending order.



9. **Bug\_Update Table:**

**Index 1**: Primary key is update\_ID - Default INDEX

INDEX\_NAME - update\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on released\_by

INDEX\_NAME - released\_by

<type - Non-Clustered>

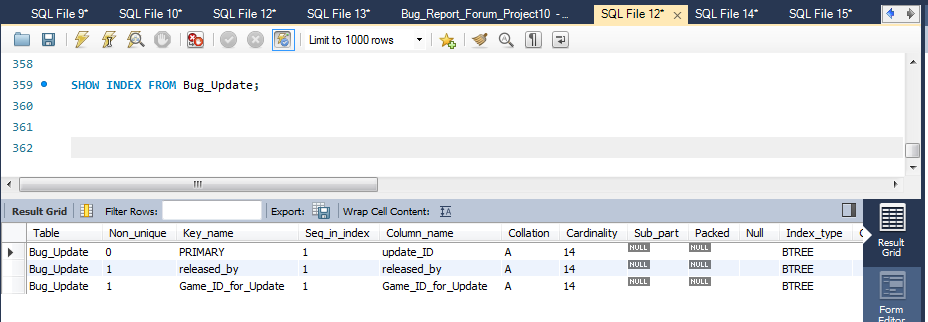
**Justification**: released\_by is foreign key for this table which refers to Developer\_ID of the Developer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 3**: Index on Game\_ID\_for\_Update

INDEX\_NAME - Game\_ID\_for\_Update

<type - Non-Clustered>

**Justification**: Game\_ID\_for\_Update is foreign key for this table which refers to Game\_ID of the Game Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.



10. **Master\_Request Table:**

**Index 1**: Primary key is Master\_Request\_ID - Default INDEX

INDEX\_NAME - Master\_Request\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Owner\_Ship

INDEX\_NAME - Owner\_Ship

<type - Non-Clustered>

**Justification**: Owner\_Ship is foreign key for this table which refers to Developer\_ID of the Developer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 3**: Index on Closed\_by

INDEX\_NAME - Closed\_by

<type - Non-Clustered>

**Justification**: Closed\_by is foreign key for this table which refers to Developer\_ID of the Developer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 4**: Index on update\_ID

INDEX\_NAME - update\_ID

<type - Non-Clustered>

**Justification**: update\_ID is foreign key for this table which refers to update\_ID of the Bug\_Update Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 5**: Index on Opened\_by

INDEX\_NAME - Opened\_by

<type - Non-Clustered>

**Justification**: Opened\_by is foreign key for this table which refers to EscalationEngineer\_ID of the EscalationEngineer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 6**: Index on Assigned\_Team\_ID

INDEX\_NAME - Assigned\_Team\_ID

<type - Non-Clustered>

**Justification**: Assigned\_Team\_IDis foreign key for this table which refers to Developer\_Team\_ID of the DeveloperTeams Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 7**: Index on Time\_created

INDEX\_NAME - Master\_Request\_CreatedTime

<type - Non-Clustered>

**Justification**: To know from which year(Date) or time this particular Master\_Request is created, basically this index is helpful to get information on Master Requests which are created by Escalation Engineers. So to such information this index is created. This index arranges the time in ascending order.

**Index 8**: Index on Status

INDEX\_NAME - Master\_Request\_Status

<type - Non-Clustered>

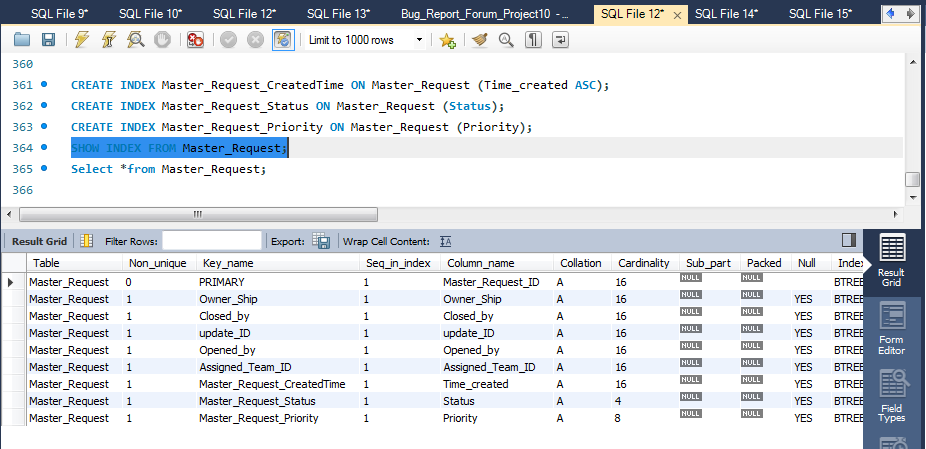
**Justification**: When we make searches based on the status that means to fetch the requests which are in open or closed or work-in-progress condition then this index is useful.

**Index 9**: Index on Priority

INDEX\_NAME - Master\_Request\_Priority

<type - Non-Clustered>

**Justification**: For searches which involves Priority of the Bug then this index is useful.



11. **Request Table:**

**Index 1**: Primary key is Request\_ID - Default INDEX

INDEX\_NAME - Request\_ID

<type - Clustered>

**Justification**: For Primary key a default Index is created, so we need not create new INDEX on Primary Key.

**Index 2**: Index on Owner\_Ship

INDEX\_NAME - Owner\_Ship

<type - Non-Clustered>

**Justification**: Owner\_Ship is foreign key for this table which refers to EscalationEngineer\_ID of the EscalationEngineer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 3**: Index on Closed\_by

INDEX\_NAME - Closed\_by

<type - Non-Clustered>

**Justification**: Closed\_by is foreign key for this table which refers to EscalationEngineer\_ID of the EscalationEngineer Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 4**: Index on Master\_Request\_ID

INDEX\_NAME - Master\_Request\_ID

<type -Non-Clustered>

**Justification**: Master\_Request\_ID is foreign key for this table which refers to Master\_Request\_ID of the Master\_Request Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 5**: Index on Opened\_by

INDEX\_NAME - Opened\_by

<type - Non-Clustered>

**Justification**: Opened\_by is foreign key for this table which refers to user\_ID of the User Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 6**: Index on Assigned\_Team\_ID

INDEX\_NAME - Assigned\_Team\_ID

<type - Non-Clustered>

**Justification**: Assigned\_Team\_ID is foreign key for this table which refers to EscalationEngineer\_Team\_ID of the EscalationTeams Table. When there are joins then this index helps in faster scanning of tables to fetch data and impact performance when the primary key value is updated, or if the row is deleted.

**Index 7**: Index on Time\_created

INDEX\_NAME - Request\_CreatedTime

<type - Non-Clustered>

**Justification**: To know from which year(Date) or time this particular Master\_Request is created, basically this index is helpful to get information on Master Requests which are created by Escalation Engineers. So to such information this index is created. This index arranges the time in ascending order.

**Index 8**: Index on Status

INDEX\_NAME - Request\_Status

<type - Non-Clustered>

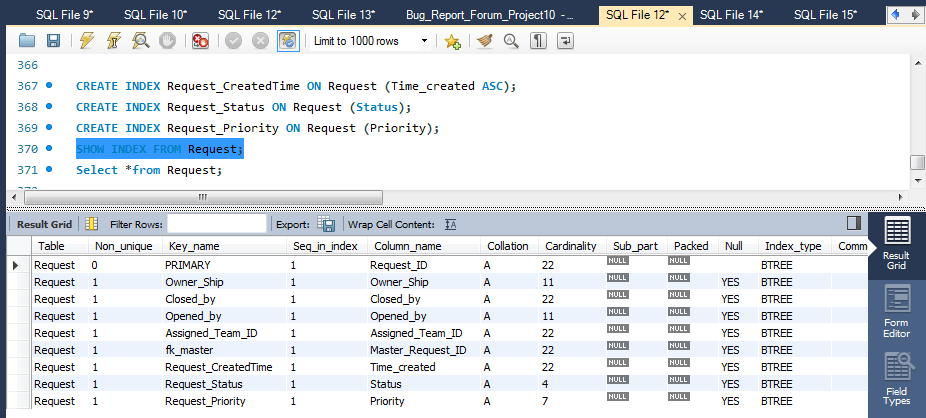
**Justification**: When we make searches based on the status that means to fetch the requests which are in open or closed or work-in-progress condition then this index is useful.

**Index 9**: Index on Priority

INDEX\_NAME - Request\_Priority

<type - Non-Clustered>

**Justification**: For searches which involves Priority of the Bug then this index is useful.



***VIEWS AND STORED PROGRAMS***

1. **Stored procedure**: <Request\_Details>

DELIMITER $$

USE `Bug\_Report\_Forum\_Project10`$$

CREATE PROCEDURE `Request\_Details` (IN Request\_ID varchar(20))

BEGIN

Select t1.Request\_ID,t2.EscalationEngineer\_ID,t4.Developer\_ID,t6.Game\_ID,t6.Name Game\_Name,t1.Priority,

t1.Status,t5.update\_ID,t5.Description Bug\_description

from Request t1

INNER JOIN EscalationEngineer t2 ON t2.EscalationEngineer\_ID = t1.Owner\_Ship

INNER JOIN Master\_Request t3 ON t3.Master\_Request\_ID = t1.Master\_Request\_ID

INNER JOIN Developer t4 ON t4.Developer\_ID = t3.Owner\_Ship

INNER JOIN Bug\_Update t5 ON t5.update\_ID = t3.update\_ID

INNER JOIN Game t6 ON t6.Game\_ID = t1.Game\_ID\_for\_Request

WHERE t1.Request\_ID = Request\_ID;

END$$

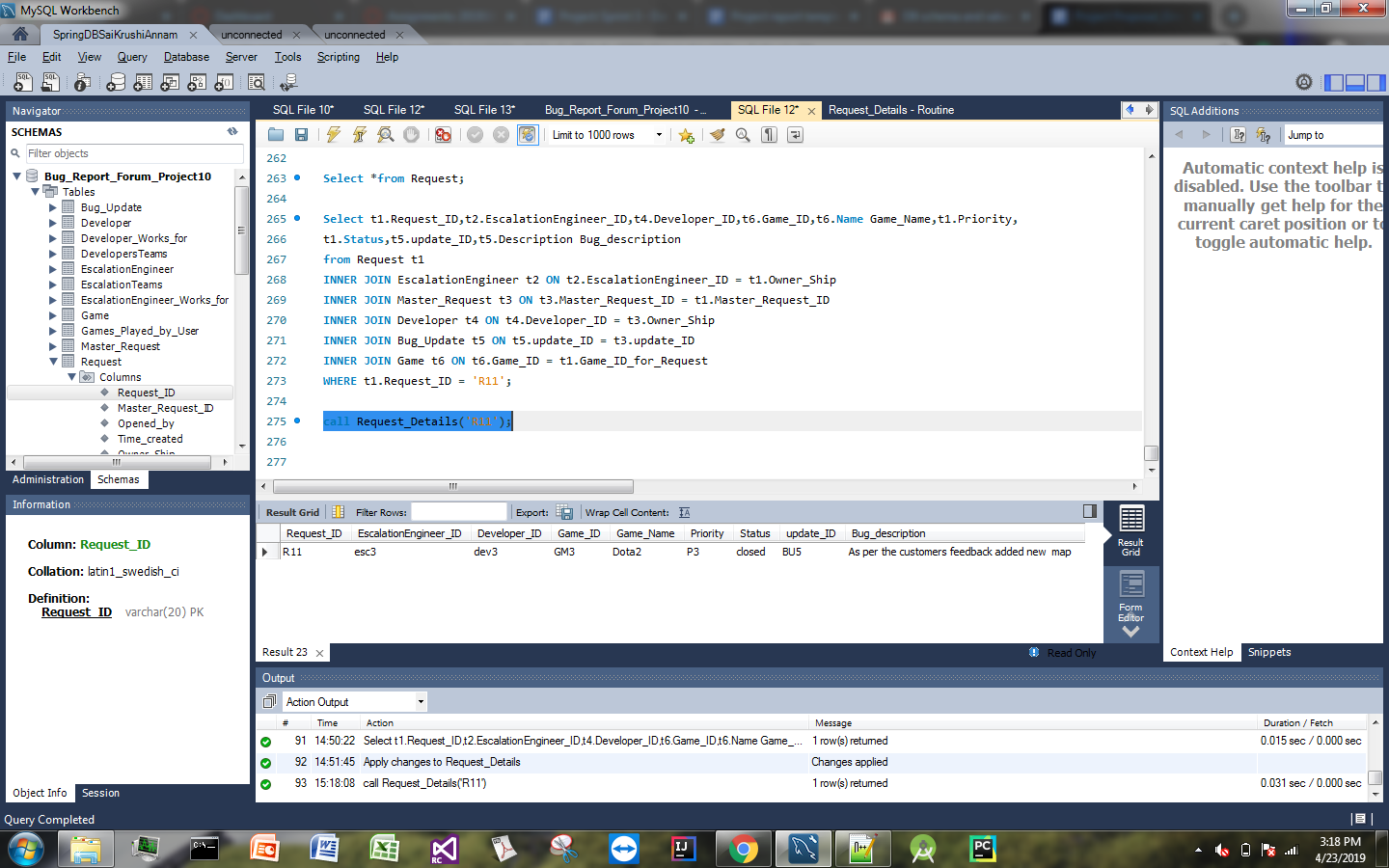
DELIMITER ;

**Parameters -** Input parameter is Request\_ID

Output we are displaying the relevant details like developer,escalation engineer working on the request, status, game etc.

**Goal :** The main idea is to fetch the details of any request from the user.

EXECUTION OF STORED\_PROCEDURE:



2. **Stored procedure**:<Developer\_Details>

DELIMITER $$

USE `Bug\_Report\_Forum\_Project10`$$

CREATE PROCEDURE `Developer\_Details` (IN Developer\_ID varchar(15))

BEGIN

Select t1.Developer\_ID,concat(t1.First\_name,' ',t1.Middle\_name,' ',t1.Last\_name) Developer\_name,t2.Team\_ID Team,

t4.Game\_ID,t4.Name Game\_Name,t6.Request\_ID,t2.Working\_since

from Developer t1

INNER JOIN Developer\_Works\_for t2 On t2.Developer\_ID = t1. Developer\_ID

INNER JOIN DevelopersTeams t3 ON t3.Developer\_Team\_ID = t2.Team\_ID

INNER JOIN Game t4 ON t4.Game\_ID = t3.Game\_ID

INNER JOIN Games\_Played\_by\_User t5 ON t5.Game\_ID = t4.Game\_ID

INNER JOIN Request t6 ON t6.Opened\_by = t5.user\_ID

WHERE t1.Developer\_ID = Developer\_ID

ORDER BY t6.Request\_ID;

END$$

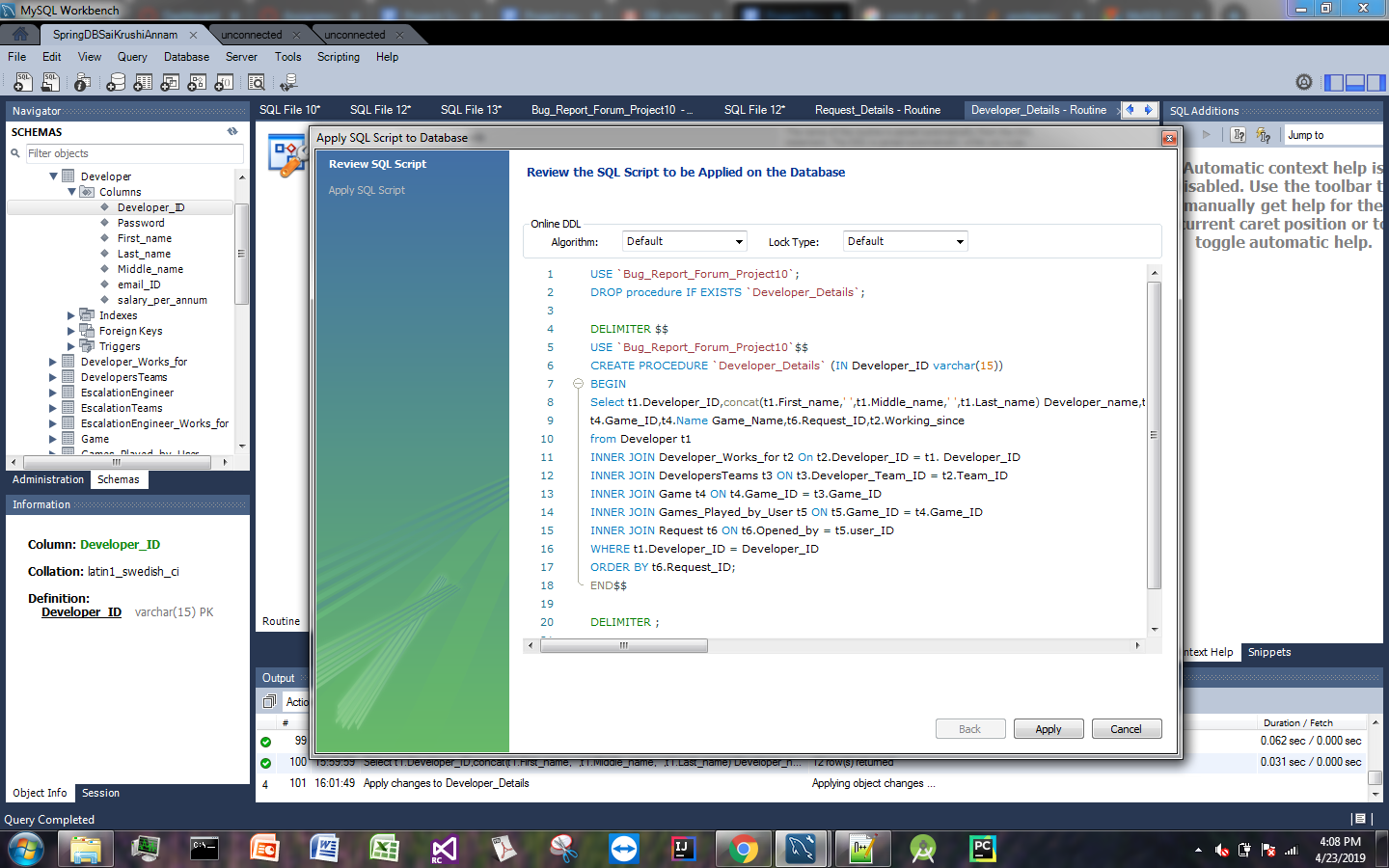
DELIMITER ;

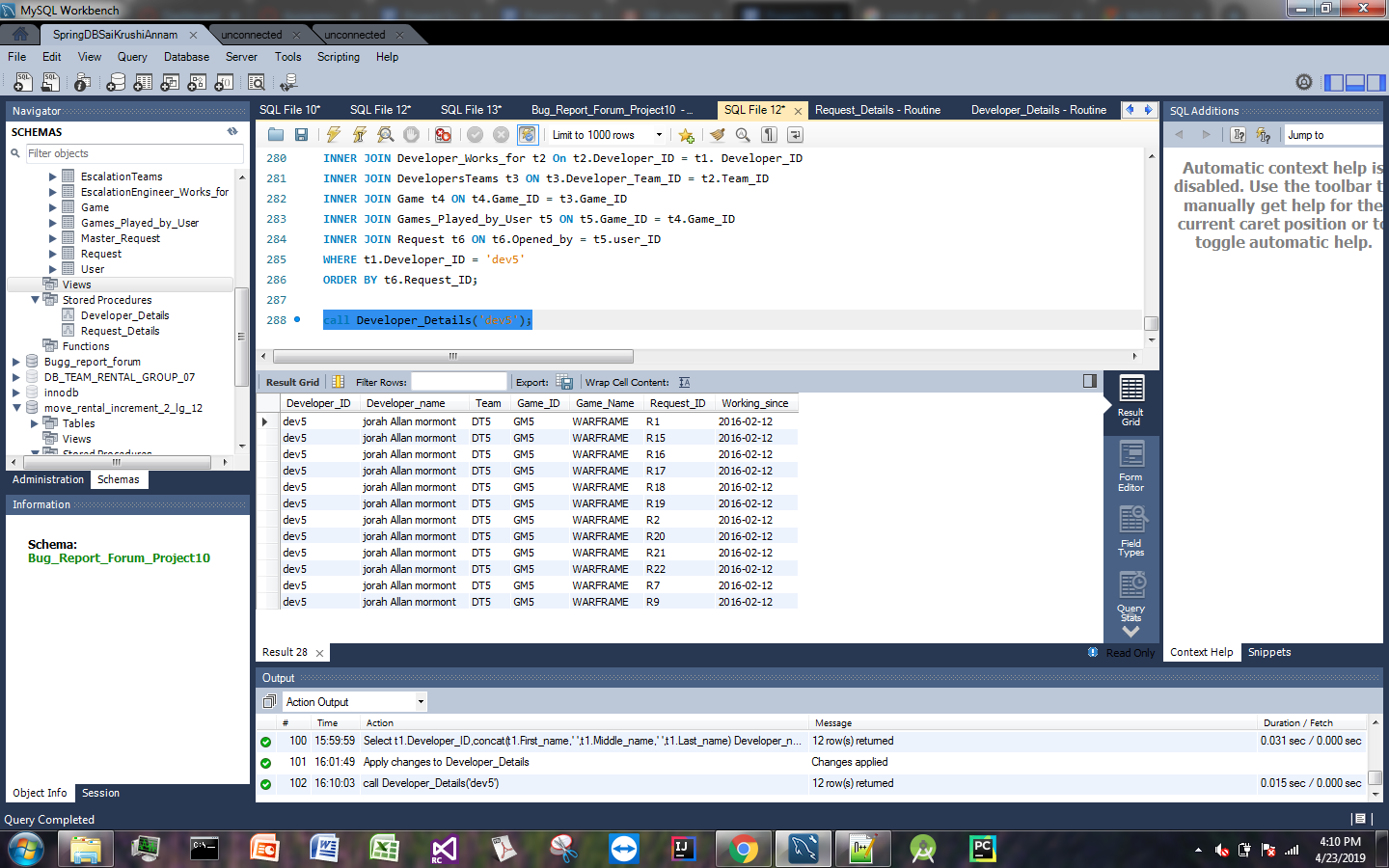
**Parameters -** Input parameter is Developer\_ID

Output we are displaying the relevant details like developer Name, Team, Game\_ID, Game Name, Working Since.

**Goal :** The main idea is to fetch the details of any developer and to know what requests he/she is working upon and from which date they are working on the Request.

EXECUTION OF STORED\_PROCEDURE:





**VIEWS:**

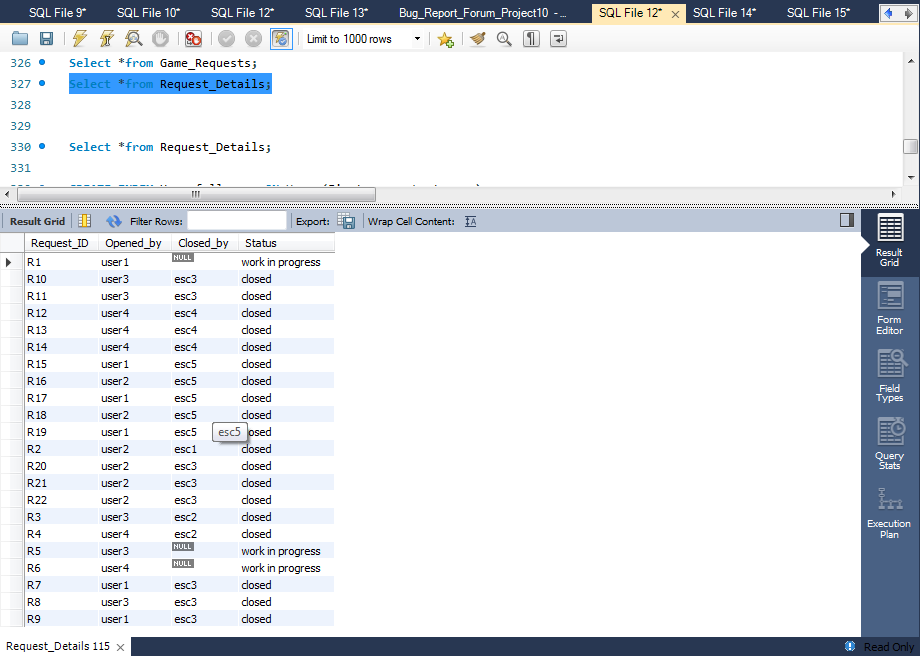
**View1**: <Request\_Details>

Goal: <This view is used to fetch details of any request in the request table, with this we can get details of which request are opened by which user and closed by which escalation Engineer and its status. This can be used by Escalation engineer for request management.

CREATE VIEW Request\_Details AS

Select Request\_ID,Opened\_by,Closed\_by,Status from Request;

Select \*from Request\_Details;



**View2**: <Game\_Requests>

Goal: <This view is used to fetch details regarding the number of requests raised for same game\_ID, this is used by escalation-engineers to know the count of requests raised on one specific game.>

CREATE VIEW Game\_Requests AS

Select t1.Game\_ID,t1.Name Game\_Name,count(t2.Request\_ID) Number\_of\_Requests

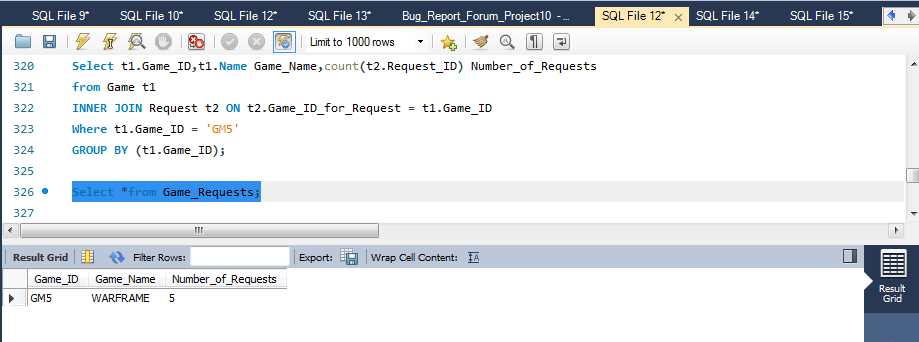
from Game t1

INNER JOIN Request t2 ON t2.Game\_ID\_for\_Request = t1.Game\_ID

Where t1.Game\_ID = 'GM5'

GROUP BY (t1.Game\_ID);

Select \*from Game\_Requests;



**View3**: <User\_EscalationEngineer\_View>

Goal: <This view is used to fetch details regarding the requests raised by user for which games and their updates, this is used by escalation-engineers to know about the different requests raised by different user.>

CREATE VIEW User\_EscalationEngineer\_View AS

Select t1.user\_ID, concat(t1.First\_name,' ',t1.Middle\_name,' ',t1.Last\_name) User\_name,t2.Request\_ID, t2.Status,

t2.Priority, t2.Time\_created, t3.Game\_ID, t3.Name Game\_Name, t4.update\_ID, t4.released\_by

from User t1

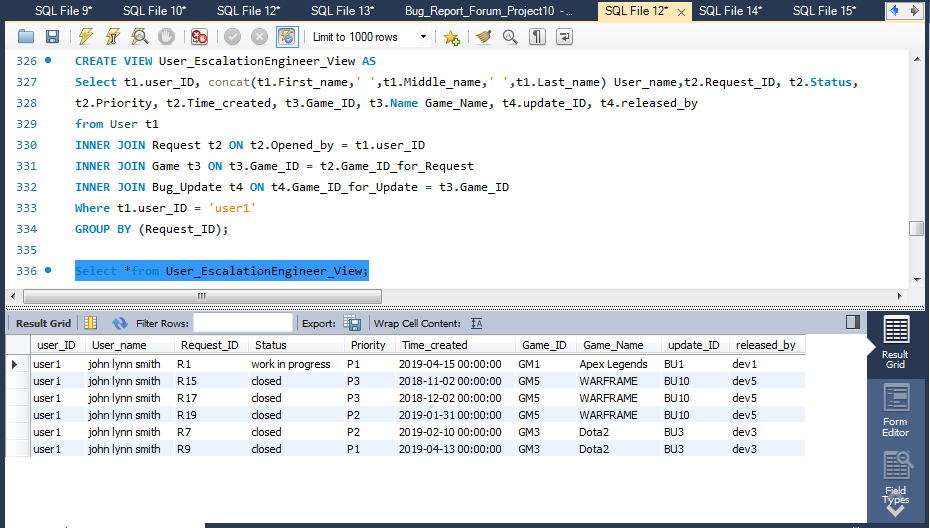
INNER JOIN Request t2 ON t2.Opened\_by = t1.user\_ID

INNER JOIN Game t3 ON t3.Game\_ID = t2.Game\_ID\_for\_Request

INNER JOIN Bug\_Update t4 ON t4.Game\_ID\_for\_Update = t3.Game\_ID

Where t1.user\_ID = 'user1'

GROUP BY (Request\_ID);

****

**Trigger**:

**Trigger 1:**

<After Insertion Trigger - DML Trigger> on <Request Table>

**Goal**: This Trigger automatically set the Last\_modified field to the current timestamp before inserting the data into Request Table.

DELIMITER $$

CREATE TRIGGER after\_request\_insert

before insert ON request

FOR EACH ROW

BEGIN

SET

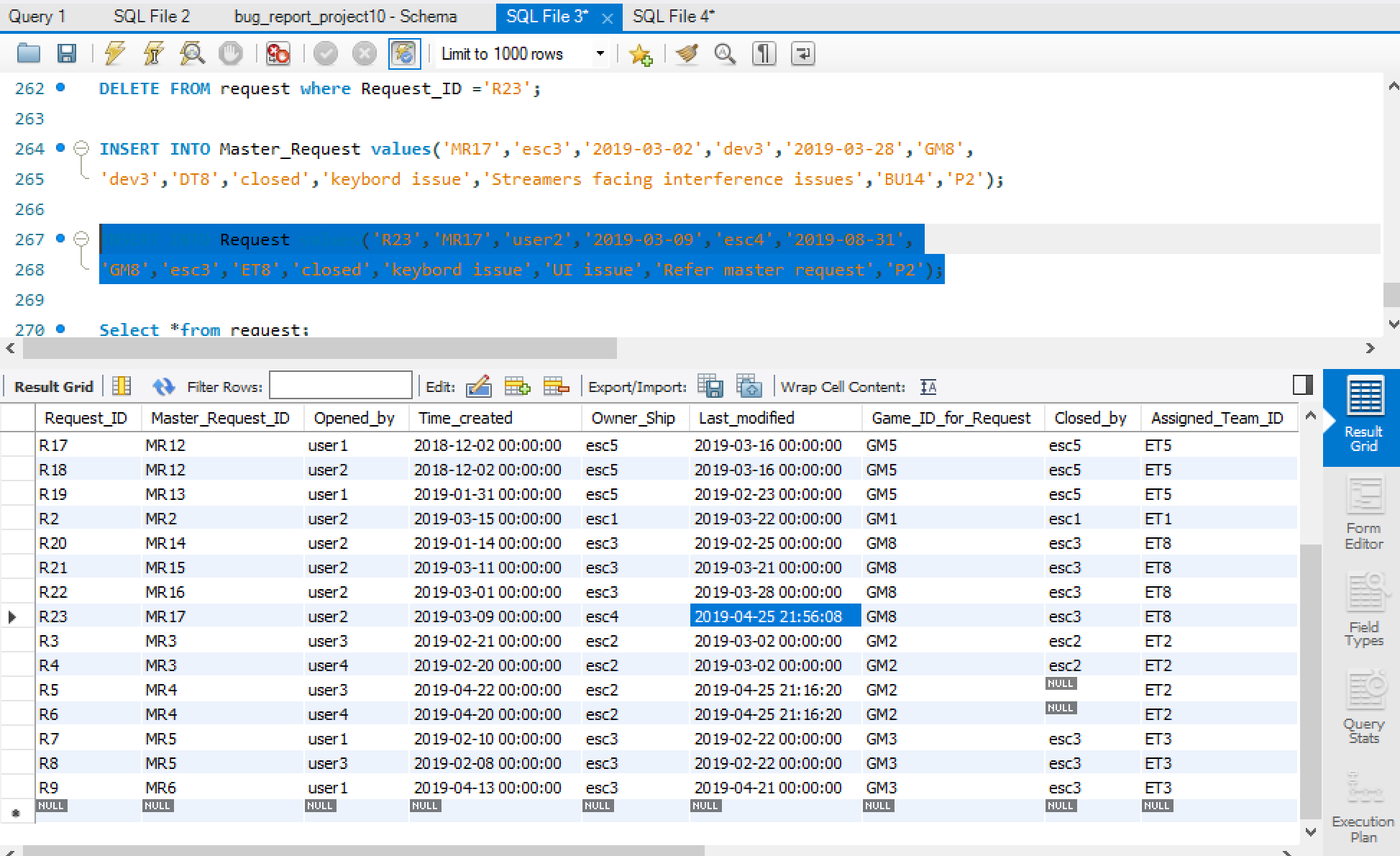
new.Last\_modified = current\_timestamp();

END $$

DELIMITER ;

INSERT INTO Request values('R23','MR17','user2','2019-03-09','esc4','2019-08-31',

'GM8','esc3','ET8','closed','keybord issue','UI issue','Refer master request','P2');



**Trigger 2:**

<Update a row - DML Trigger> on <Master\_Request Table>

**Goal**: This Trigger automatically sets the value of status field as ‘closed’ in the Request table when the Master\_Request is closed.

DELIMITER $$

CREATE TRIGGER after\_update\_master\_request

after update ON master\_request

FOR EACH ROW

BEGIN

if new.status="closed" then

update request set status="closed" where Master\_Request\_ID=new.Master\_Request\_ID;

end if;

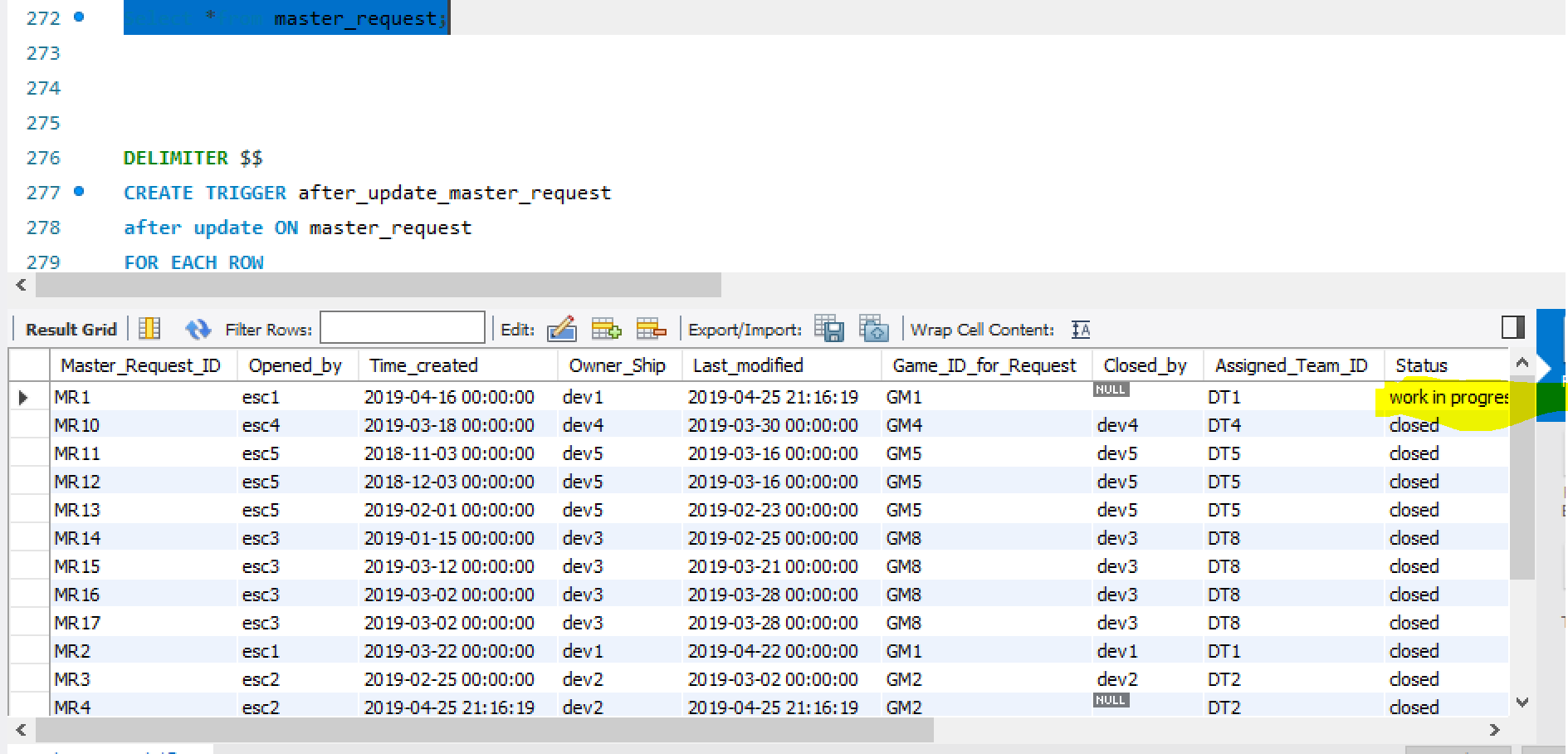
END $$

DELIMITER ;

TRIGGER EXPLANATION:

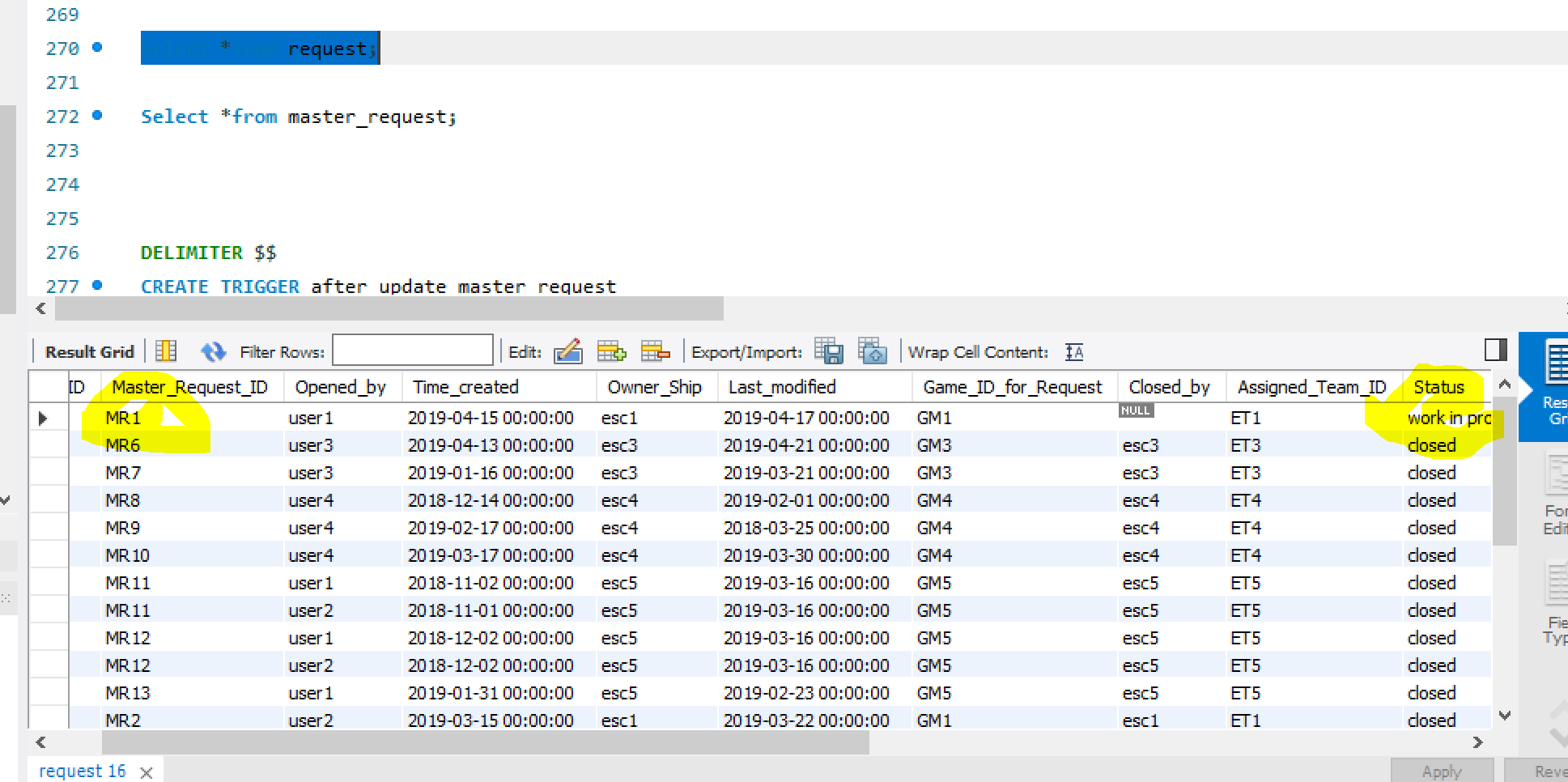
Before Updating: Master\_Request Table

Request Status is in Work-in-Progress



Before Updating: Request Table

Request Status is in Work-in-Progress

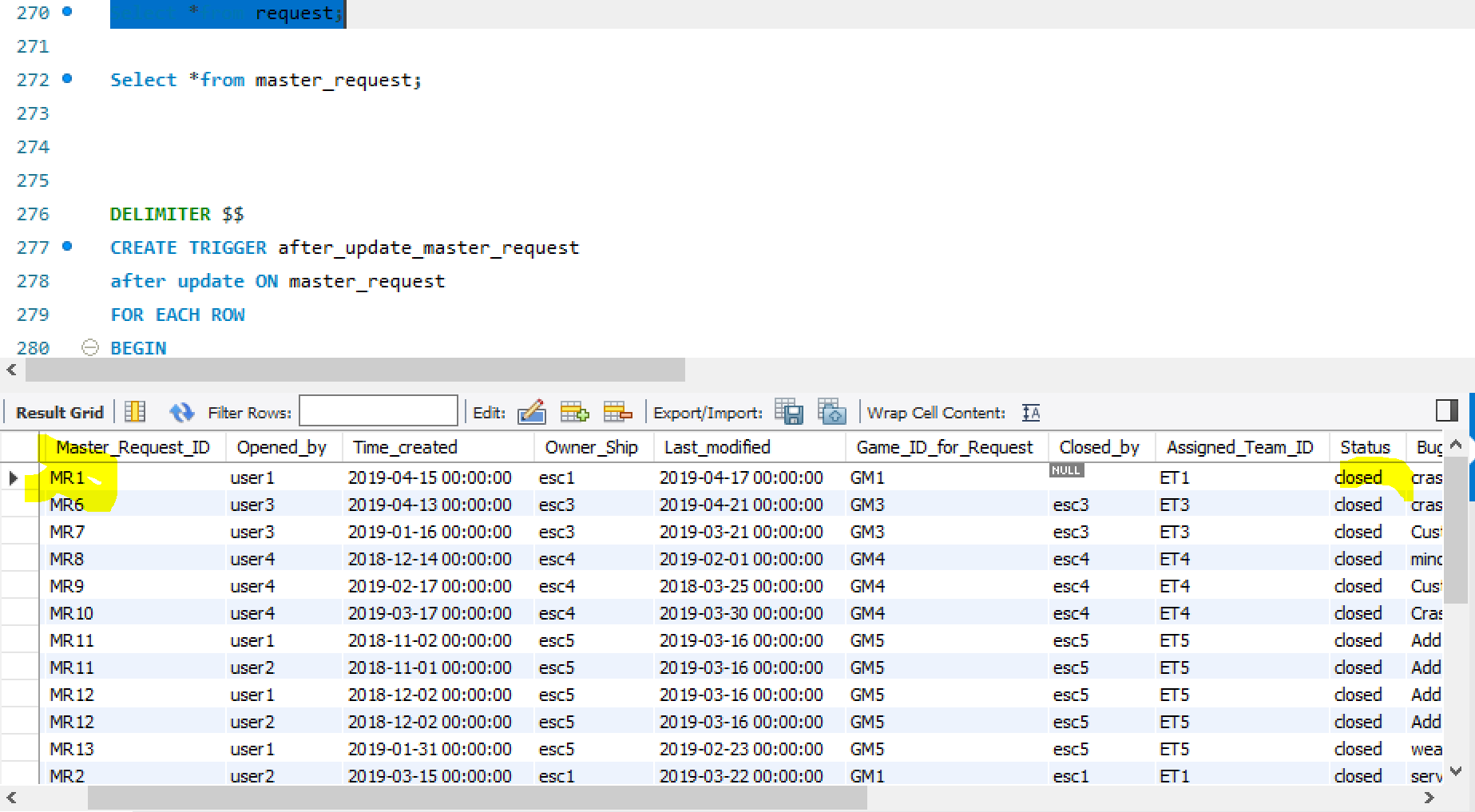


After Updating the Master\_Request Table:

UPDATE master\_request set status="closed" where Master\_Request\_ID="MR1";

After Updating: Request Table

Request Status is in ‘closed’



**Events**:

Event 1: Last\_modified\_Deletion

DELIMITER $$

CREATE EVENT Last\_modified\_Deletion

ON SCHEDULE EVERY 1 day

STARTS CURRENT\_TIMESTAMP

DO

BEGIN

DELETE FROM Master\_Request

WHERE Last\_modified < DATE\_SUB(curdate(), INTERVAL 365 DAY);

END$$

DELIMITER ;

EVENT\_TYPE:

<DELETE ROW Event - DDL EVENT>

**Goal**: This event deletes the rows in the Master\_Request Table when Last\_modified date is more than a year.

Event 2: Last\_modified\_Deletion\_Request

DELIMITER $$

CREATE EVENT Last\_modified\_Deletion\_Request

ON SCHEDULE EVERY 1 day

STARTS CURRENT\_TIMESTAMP

DO

BEGIN

DELETE FROM Request

WHERE Last\_modified < DATE\_SUB(curdate(), INTERVAL 365 DAY);

END$$

DELIMITER ;

EVENT\_TYPE:

<DELETE ROW Event - DDL EVENT>

**Goal**: This event deletes the rows in the Request Table when Last\_modified date is more than a year.