Name: Talha Matal

**Table of Contents**

[Project Direction Overview 2](#_Toc66715848)

[Use Cases and Fields 2](#_Toc66715849)

[Structural Database Rules 2](#_Toc66715850)

[Conceptual Entity-Relationship Diagram 2](#_Toc66715851)

[Full DBMS Physical ERD 2](#_Toc66715852)

[History Table and Trigger 2](#_Toc66715853)

[Index Identification and Creations 2](#_Toc66715854)

[Stored Procedure Execution and Explanations 2](#_Toc66715855)

[Question Identification and Explanations 2](#_Toc66715856)

[Query Executions and Explanations 2](#_Toc66715857)

[Data Visualizations 3](#_Toc66715858)

[Summary and Reflection 3](#_Toc66715859)

# Project Direction Overview

Update, if necessary, the overview that describes who the database will be for, what kind of data it will contain, how you envision it will be used, and most importantly, why you are interested in it.

For my term project, I would like to develop a web-based application which can be used in an enterprise environment to track / document / request/ decommission Online Storage (SAN or NAS) for a particular application within the organization. In an enterprise environment / organization the application can be used to request new storage , track the storage that is used / allocated for a particular application/server and it will also help in determining the TCO for that application from the storage perspective.

The application development overall is a complex task and involved much greater steps for the purpose of this class I will only be focusing on the database component of the application which will store all the data which includes application name, IT owner info, storage details (SAN or NAS), server/hostname where the storage will be required, request that was raised for that and the date when it was raised. For the decommissioning part of the application the database will record the request number, application name, IT owner info, storage details (SAN or NAS), server/hostname from which the storage will be removed and the date it was raised. For the TCO I will be adding the cost($) per GB for the storage so the value of the cost can be added to the database itself.

I envision that the database will be used when a new application is onboarded and requires online storage they will be onboarded to this database by creating a new app ID and add the application name. When a new storage request is created the request number / details(amount (GB) and the type of SAN or NAS, server /hostname) along with the date it is created on will be added to the database. For the decommission /removal request similar information including the request number /details (amount (GB) and the type of SAN or NAS, server/hostname) along with the date it was created will be added to the database. Lastly there will be field for the cost associated with that amount of storage which can be used to calculate the total cost of ownership for the application consuming the storage.

I am interested in this project as I have a background in enterprise data storage and being a storage admin myself I can understand the challenges of not having such a central repository to track the usage for the storage consumed for each application etc.

# Use Cases and Fields

Update, if necessary, the five or more use cases that enumerate steps of how the database will be typically used, and also identify significant database fields needed to support each use case.

For my project one of the important use cases for the database is when an application owner wants to onboard the application so it can start requesting online storage.

Application Onboarding

1. The application owner connects to the web-based application to onboards a new application.
2. The web application asks the owner to provide details like application name, IT owner name, contact info (email etc.) for the IT owner and lastly any servers /hostnames for the application that will require storage.

From the database perspective it involves in creating the application entries within the database where the application name , IT owner name , contact details and server details for the application (hostnames) are added.

|  |  |  |
| --- | --- | --- |
| Field | Data it stores | Why needed |
| AppID | Unique Numeric application number ID | It is needed to uniquely identify each application from the other |
| Application Name | Name of the application that is requesting the storage | It is needed to identify the application when trying to list the amount of storage associated with each |
| IT Owner Name | It stores the full name of the application owner the person responsible for requesting the storage | This field is needed to identify who is the owner for the application requesting storage, so a contact reference is present in case they need to be contacted |
| IT Owner Contact | This stores the email address for the IT Owner | This is required so the owner can be contacted in case they need to be reached out for any questions or queries regarding there storage |
| Server Name | Stores the name of server which the application will be requesting storage | This is required so if the application has multiple components requiring storage so the correct server can be identified where the storage is needed |

Another important use case for my project is to put in the details for request for the storage within the database.

Request for storage

1. The application (IT) owner opens a new storage request.
2. The requestor provides the information within the request which includes the application name, type of storage required (SAN or NAS), the amount of storage required, the server the storage is required for, the location where the storage is required, what type of environment this is for PROD, UAT or DEV and the replication type (synchronous or asynchronous).
3. The database will calculate and add a column /field for the total cost for the requested amount of storage.

The below fields will be required for this use case from a database perspective.

|  |  |  |
| --- | --- | --- |
| Field | Data it stores | Why needed |
| Request No | Unique Alphanumeric request number | It is needed to identify the request number that was raised for that storage and we have a record of when and by whom the request was raised |
| Request Type | It stores the type of request like add request or removal request | This is needed so it can be distinguished if the request was for a new provision or for removal |
| Application Name | Name of the application that is requesting the storage | It is needed to identify the application when trying to list the amount of storage associated with each |
| Requestor Name | It stores the full name for the requestor that is requesting the storge | This field is needed to identify who is the requestor for that storage request is, so a contact reference is present in case they need to be contacted |
| Type of storage | This field contains what type of storage is required SAN or NAS | This field is required so it can be identified easily what type of storage is requested so the costing for that appropriate type of storage can be done later |
| Amount of storage | This field contains the amount of storage that is request in Gigabytes | This field is required so that record of the amount of storage request with each request is kept and the same value is used in calculating the cost associated with each request |
| Server Name | Stores the name of server which the application will be requesting storage | This is required so there is a track of where the storage was allocated to and can be tracked easily from the usage perspective |
| Location | Stores the location / site where the storage is needed | This field is required so that it can be identified which location to provision the storage from |
| Total Cost | It stores the total cost of the storage that is being removed | This field is required to identify the cost for the storage associated with that request |
| Environment | This stores the type of environment the request is for PROD, UAT or DEV | It is required to identify the environment type, so the storage is provisioned accordingly |
| Replication | This stores the type of replication required for the storage that is being requested | This is required to identify the replication if it is required for the storage request so the request can be provisioned accordingly |

Decommission /Removal for storage

1. The application (IT) owner opens a storage removal / decommission request.
2. The requestor provides the information within the request which includes the application name, type of storage to be removed (SAN or NAS), the amount of storage removed, the server the storage is to be removed for, the location and the environment it is for.
3. The database will calculate and add a column /field for the total cost for the requested amount of storage.

The below fields will be required for this use case from a database perspective.

|  |  |  |
| --- | --- | --- |
| Field | Data it stores | Why needed |
| Request No | Unique Alphanumeric request number | It is needed to identify the request number that was raised for that removal and we have a record of when and by whom the request was raised |
| Request Type | It stores the type of request like add request or removal request | This is needed so it can be distinguished if the request was for a new provision or for removal |
| Application Name | Name of the application that is requesting the removal of storage | It is needed to identify the application when trying to list the storage that was removed for a particular application |
| Requestor Name | It stores the full name for the requestor that is requesting the removal | This field is needed to identify who is the requestor for that storage removal request is, so a contact reference is present in case they need to be contacted |
| Type of storage | This field contains what type of storage needs to be removed SAN or NAS | This field is required so it can be identified easily what type of storage is removed so the costing for that appropriate type of storage can be done later |
| Amount of storage | This field contains the amount of storage that is removed in Gigabytes | This field is required so that record of the amount of storage removed with each request is kept for cost calculation and removal |
| Server Name | Stores the name of server which the application will be requesting storage | This is required so there is a track of where the storage was removed from so it can be tracked easily from the usage perspective |
| Location | Stores the location / site where the storage is removed from | This field is required so that it can be identified which location to remove the storage from |
| Total Cost | It stores the total cost of the storage that is being removed | This field is required to identify the cost for the storage associated with that request |
| Environment | This stores the type of environment the request is for PROD, UAT or DEV | It is required to identify the environment type, so the storage is provisioned accordingly |

The next use case for the database is for the application owner to lookup past records for the request that were raised for that application to request storage.

Storage requests lookup

1. The application owner connects to the web application.
2. The application owner provides the application name and can look up all the requests and the dates associated with those.
3. The web application will query the underlying database to get all the required information for that application like requests for provisioning, request for removal, amount of storage associated with each, type of storage SAN or NAS, the server name the storage is associated with, the location and the environment that the request was for.

This use case with the storage request lookup will use the below fields as this is only going to look up the data from the database and present it for the application owner to review.

|  |  |  |
| --- | --- | --- |
| Field | Data it stores | Why needed |
| Request No | Unique Alphanumeric request number | It is needed to identify the request number that was raised for that removal and we have a record of when and by whom the request was raised |
| Request Type | It stores the type of request like add request or removal request | This is needed so it can be distinguished if the request was for a new provision or for removal |
| Application Name | Name of the application that is requesting the removal of storage | It is needed to identify the application when trying to list the storage that was removed for a particular application |
| Requestor Name | It stores the full name for the requestor that is requesting the removal | This field is needed to identify who is the requestor for that storage removal request is, so a contact reference is present in case they need to be contacted |
| Type of storage | This field contains what type of storage needs to be removed SAN or NAS | This field is required so it can be identified easily what type of storage is removed so the costing for that appropriate type of storage can be done later |
| Amount of storage | This field contains the amount of storage that is removed in Gigabytes | This field is required so that record of the amount of storage removed with each request is kept for cost calculation and removal |
| Server Name | Stores the name of server which the application will be requesting storage | This is required so there is a track of where the storage was removed from so it can be tracked easily from the usage perspective |
| Location | Stores the location / site where the storage is removed from | This field is required so that it can be identified which location to remove the storage from |
| Environment | This stores the type of environment the request is for PROD, UAT or DEV | It is required to identify the environment type, so the storage is provisioned accordingly |

Calculate the total cost of ownership for storage for an application

1. The application owner enters the application name within the web application.
2. And requests to provide the total storage allocated regardless of type and the total cost of ownership for that storage.
3. The web application to query the database to lookup the request associated with that application and calculate the total cost of ownership for that application based on the storage that is requested and the storage that is removed for that application.

This use case will utilize existing fields as described below by looking up the application name and the storage requests associated with each app and finding the total cost field for each request and providing the result based on new storage request and request for removal.

|  |  |  |
| --- | --- | --- |
| Field | Data it stores | Why needed |
| Request No | Unique Alphanumeric request number | It is needed to identify the request number that was raised for that removal and we have a record of when and by whom the request was raised |
| Request Type | It stores the type of request like new provisioning, expansion request or removal | This is needed so it can be distinguished if the request was for a new provision or for removal |
| Application Name | Name of the application that is requesting the removal of storage | It is needed to identify the application when trying to list the storage that was removed for a particular application |
| Amount of storage | This field contains the amount of storage that is removed in Gigabytes | This field is required so that record of the amount of storage removed with each request is kept for cost calculation and removal |
| Total Cost | It stores the total cost of the storage that is being removed | This field is required to identify the cost for the storage associated with that request |

# Structural Database Rules

Update, if necessary, the list of structural database rules for all significant entities and relationships, with the constraints defined, based upon the use cases you defined, along with supporting explanations.

I will start with the use cases that I have defined for my databases above.

Starting with the first use case.

Application Onboarding

1. The application owner connects to the web-based application to onboards a new application.
2. The web application asks the owner to provide details like application name, IT owner name, contact info (email etc.) for the IT owner and lastly any servers /hostnames for the application that will require storage.

From the database perspective it involves in creating the application entries within the database where the application name , IT owner name , contact details and server details for the application (hostnames) are added. From the perspective of the database as we need to store the application info so the entity that I can identify is the Application. I can also see that the server is another entity that can be used for the database. Another entity that I can identify is the IT owner. So, from the first use case I identified three entities Application, Server, and IT owner.

Moving to the second use case that I have defined.

Request for storage

1. The application (IT) owner opens a new storage request.
2. The requestor provides the information within the request which includes the application name, type of storage required (SAN or NAS), the amount of storage required, the server the storage is required for, the location where the storage is required, what type of environment this is for PROD, UAT or DEV and the replication type (synchronous or asynchronous).
3. The database will calculate and add a column /field for the total cost for the requested amount of storage.

From this I can see that there are a few significant data points Request, Application, Type of storage, Server, Location, Environment and Replication.

Moving to the third use case that I have defined.

Decommission /Removal for storage

1. The application (IT) owner opens a storage removal / decommission request.
2. The requestor provides the information within the request which includes the application name, type of storage to be removed (SAN or NAS), the amount of storage removed, the server the storage is to be removed for, the location and the environment it is for.
3. The database will calculate and add a column /field for the total cost for the requested amount of storage.

From this I can see that there are a few significant data points Request, Application, Type of storage, Server, Location and Environment.

Moving to the next use case.

Storage requests lookup

1. The application owner connects to the web application.
2. The application owner provides the application name and can look up all the requests and the dates associated with those.
3. The web application will query the underlying database to get all the required information for that application like requests for provisioning, request for removal, amount of storage associated with each, type of storage SAN or NAS, the server name the storage is associated with, the location and the environment that the request was for.

As no new fields are identified by this use case so no new entities can be identified by this use case.

The last use case for my database involves in the lookup of all the requests to identify the cost of storage associated with each application.

Calculate the total cost of ownership for storage for an application

1. The application owner enters the application name within the web application.
2. And requests to provide the total storage allocated regardless of type and the total cost of ownership for that storage.
3. The web application to query the database to lookup the request associated with that application and calculate the total cost of ownership for that application based on the storage that is requested and the storage that is removed for that application.

Based on the use case I can identify Cost as another entity that is associated with the data within the database. We will name it as Invoice for the entity name.

So far I have identified the below Entities

1. Application
2. Server
3. IT owner
4. Request
5. Type of storage
6. Location
7. Environment
8. Invoice
9. Replication type

Based on the entities that we have identified we can start writing down the Associative Structural database rules.

As each application that is being onboarded can have multiple servers associated with it whereas each server can be owned by many applications so we can write the below rule

1. Each application is associated with one or many servers; each server is associated with one or many applications.

As each application that is being onboarded can have an IT owner associated with it whereas each IT owner may own one or many applications so we can write the below rule

1. Each application is associated with one IT owner; each IT owner can be associated with one or many applications.

Each request that is being created by on the web application is for a specific application and there can be multiple requests for each application. We can write the below.

1. Each request is opened for a specific application; each application can have multiple requests opened.

For the request that is opened it can be for either a SAN or a NAS. So, we can conclude that the request is of one type only and each storage type can be associated with many requests.

1. Each request must contain one type of storage; each type of storage maybe contained in many requests.

As in an enterprise environment the servers they can be spread across multiple sites (datacenters) so the server can be in one location and each location can have multiple servers.

1. Each server has one location; each location has multiple servers.

For every request that gets created there can be one server defined in the request

1. Each request is associated with one server; each server can be associated with multiple requests.

For every server belongs to a specific environment like PROD, UAT or DEV so the provisioning can be done accordingly. So, we can say

1. Each server has an environment; an environment can have multiple servers.

The Invoice is associated with each of the requests only so we can say

1. Each request has an Invoice associated with it; the Invoice can be associated with one request.

For the request that gets opened there can only one type of replication that be selected so another rule is as below

1. Each request may have a replication type, each replication type can have many requests.

The above we defined some of the associative structural database rules, but we can also define some of the specialization- generalization structural database rules also

Moving to the second use case we can identify that there is a type of storage which can be either SAN Storage or NAS Storage but as there is another storage type object based that is we are not catering for in our design so we can say that this relationship is partially complete but is disjoint as the storage type can be either SAN Storage or NAS Storage but not both.

1. The type of storage is SAN Storage, NAS Storage, or none of these.

For the location of the server there will be an address field for each of the location which is normalized further to add two more entities for City and State. The Request type field and the requestor name field are also identified as entities so the a few more structural database rules need to be defined as below.

1. Each Location has a City; Each City can have many locations.
2. Each Location has a State; Each state has many locations.
3. Each request has a type; each request\_type can be with many requests
4. Each request is opened by a requestor(Person) ; Each requestor(Person) can open many requests

The new structural database rule that needs to be defined for this history table is as below.

1. Each Invoice can have many cost changes; Each cost change is for one Invoice.

Listing all the Structural Database Rule

1. Each application is associated with one or many servers; each server is associated with one or many applications.
2. Each application is associated with one IT owner; each IT owner can be associated with one or many applications.
3. Each request is opened for a specific application; each application can have multiple requests opened.
4. Each request must contain one type of storage; each type of storage maybe contained in many requests.
5. Each server has one location; each location has multiple servers.
6. Each request is associated with one server; each server can be associated with multiple requests.
7. Each server has an environment; an environment can have multiple servers.
8. Each request has an Invoice associated with it; the Invoice can be associated with one request.
9. Each request may have a replication type, each replication type can have many requests.
10. The type of storage is SAN Storage, NAS Storage, or none of these.
11. Each Location has a City; Each city has many locations.
12. Each Location has a State; Each state has many locations.
13. Each request has a type; each request\_type can be with many requests
14. Each request is opened by a requestor(Person) ; Each requestor(Person) can open many requests
15. Each Invoice can have many cost changes; Each cost change is for one Invoice.

# Conceptual Entity-Relationship Diagram

Update, if necessary, the conceptual ERD that visualizes the structural database rules, along with supporting explanations.

For my Entity-Relationship Diagram I am using crow’s foot representation.

1. Each application is associated with one or many servers; each server is associated with one or many applications.

As each application is associated with one or many servers so the relationship from the perspective of the application is mandatory so we use a bar to represent that, and it is plural (many) so we use a crow’s foot to represent that. From the perspective of the server the relationship is mandatory and plural (many) so we use one bar and a crow’s foot to represent that.

1. Each application is associated with one IT owner; each IT owner can be associated with one or many applications.

As each application is associated with only one IT owner so the relationship from the perspective of the application is mandatory and singular(one) so we two bars to represent that. From the perspective of the IT owner the relationship is optional (can) and plural (many) so we use a circle and a crow’s foot to represent that.

1. Each request is opened for a specific application; each application can have multiple requests opened.

As each request is opened for a single application so the relationship from the perspective of the request is mandatory (is) so we use a bar to represent that, and it is singular (specific), so we use a crow’s foot to represent that. From the perspective of the Application the relationship is optional (can have) and plural (multiple), so we use a circle with a crow’s foot to represent it.

1. Each request must contain one type of storage; each type of storage maybe contained in many requests.

The relationship is mandatory (must contain) and singular (one) from the perspective of the request, so we use two bars to represent that and from the perspective of the type of storage the relationship is optional (maybe) and plural (many) so we use a circle and a crow’s foot for the representation.

1. Each server has one location; each location has multiple servers.

The relationship is mandatory (has) and singular (one) from the perspective of the server, so we use two bars to represent that and from the perspective of the location the relationship is mandatory (has) and plural (multiple) so we use a bar and a crow’s foot for the representation.

1. Each request is associated with one server; each server can be associated with multiple requests.

The relationship is mandatory (is) and singular (one) from the perspective of the request, so we two bars to represent that and from the perspective of server the relationship is optional (can be) and plural (multiple) so we use a circle and a crow’s foot for the representation.

1. Each server has an environment; an environment can have multiple servers.

The relationship is mandatory (has) and singular (an) from the perspective of the server, so we use two bars to represent that and from the perspective of environment the relationship is optional (can have) and plural (multiple) so we use a circle and a crow’s foot for the representation.

1. Each request has an Invoice associated with it; an Invoice is associated with one request.

The relationship is mandatory (has) and singular (an) from the perspective of the request, so we use two bars to represent that and from the perspective of Invoice the relationship is mandatory (is) and singular (one) so we use two bars for the representation.

1. Each request may have a replication type, each replication type can have many requests.

The relationship is optional (may) and singular (a) from the perspective of the request, so we use a circle and a bar to represent that and from the perspective of replication type the relationship is optional (can) and plural (many) so we use a circle and a crow’s foot for the representation.

1. The type of storage is SAN Storage, NAS Storage, or none of these.

As the type of storage type be SAN Storage or NAS Storage but not both so the relationship is disjoint so we use a “d” to represent that, and it is partially complete as there are other storage types (like object storage) that can be requested so we use a single bar to represent that.

1. Each Location has a City; Each City can have many Locations.

The relationship is mandatory (has) and singular (a) from the perspective of the location, so we use two bars to represent that and from the perspective of the City the relationship is optional (can have) and plural (many) so we use a circle and a crow’s foot for the representation.

1. Each Location has a State; Each State can have many Locations.

The relationship is mandatory (has) and singular (a) from the perspective of the location, so we use two bars to represent that and from the perspective of the State the relationship is optional (can have) and plural (many) so we use a circle and a crow’s foot for the representation.

1. Each request has a type; each request\_type can be with many requests.

The relationship is mandatory (has) and singular (a) from the perspective of the request, so we use two bars to represent that and from the perspective of the Request\_type the relationship is optional (can be) and plural (many) so we use a circle and a crow’s foot for the representation.

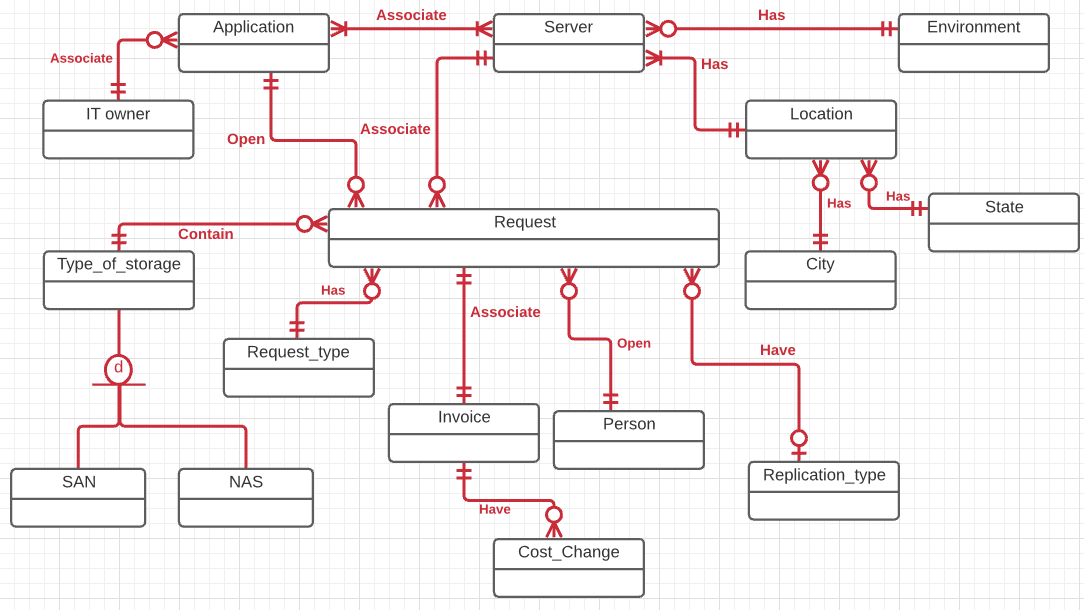
1. Each request is opened by a requestor(Person) ; Each requestor(Person) can open many requests.

The relationship is mandatory (is) and singular (a) from the perspective of the request, so we use two bars to represent that and from the perspective of the requestor(Person) the relationship is optional (can open) and plural (many) so we use a circle and a crow’s foot for the representation.

1. Each Invoice can have many cost changes; Each cost change is for one Invoice.

The relationship is optional (can) and plural (many) from the perspective of the invoice but it mandatory (is) and singular(one) from the perspective of the cost change.

The combined final Conceptual Entity relationship diagram is shown below for all the database rules.



# Full DBMS Physical ERD

Update, if necessary, your normalized DBMS physical ERD, along with supporting explanations.

For the associative relationship mapping I have identified my relationship classification from the conceptual ERD above. The relationships in my ERD are Request/ Application, Request/Replication\_type, Request/Type\_of\_storage, Request/Invoice, Request/Server, Application/IT owner, Application/Server, Server/Environment and Server/Location.

Request/ Application is a 1:M relationship. Each request is opened for a specific application; each application can have multiple requests opened.

Request/Replication\_type is 1:M relationship. Each request may have a replication type, each replication type can have many requests.

Request/type\_of\_storage is a 1:M relationship. Each request must contain one type of storage; each type of storage maybe contained in many requests.

Request/Invoice is a 1:1 relationship. Each request has an Invoice associated with it; an Invoice is associated with one request.

Request/Server is a 1:M relationship. Each request is associated with one server; each server can be associated with multiple requests.

Request/Request\_type is a 1:M relationship. Each request has a type; each request\_type can be with many requests.

Request/Person is a 1:M relationship. Each request is opened by a requestor(Person) ; Each requestor(Person) can open many requests.

Application/ IT owner is a 1:M relationship. Each application is associated with one IT owner; each IT owner can be associated with one or many applications.

Application /Server is a M:N relationship. Each application is associated with one or many servers; each server is associated with one or many applications.

Server/Environment is a 1:M relationship. Each server has an environment; an environment can have multiple servers.

Server/Location is a 1:M relationship. Each server has one location; each location has multiple servers.

Location/State is a 1:M relationship. Each Location has a State; Each State can have many Locations.

Location/City is a 1:M relationship. Each Location has a City; Each City can have many Locations.

Invoice / Cost Change is 1:M relationship. Each Invoice can have many cost changes; Each cost change is for one Invoice.

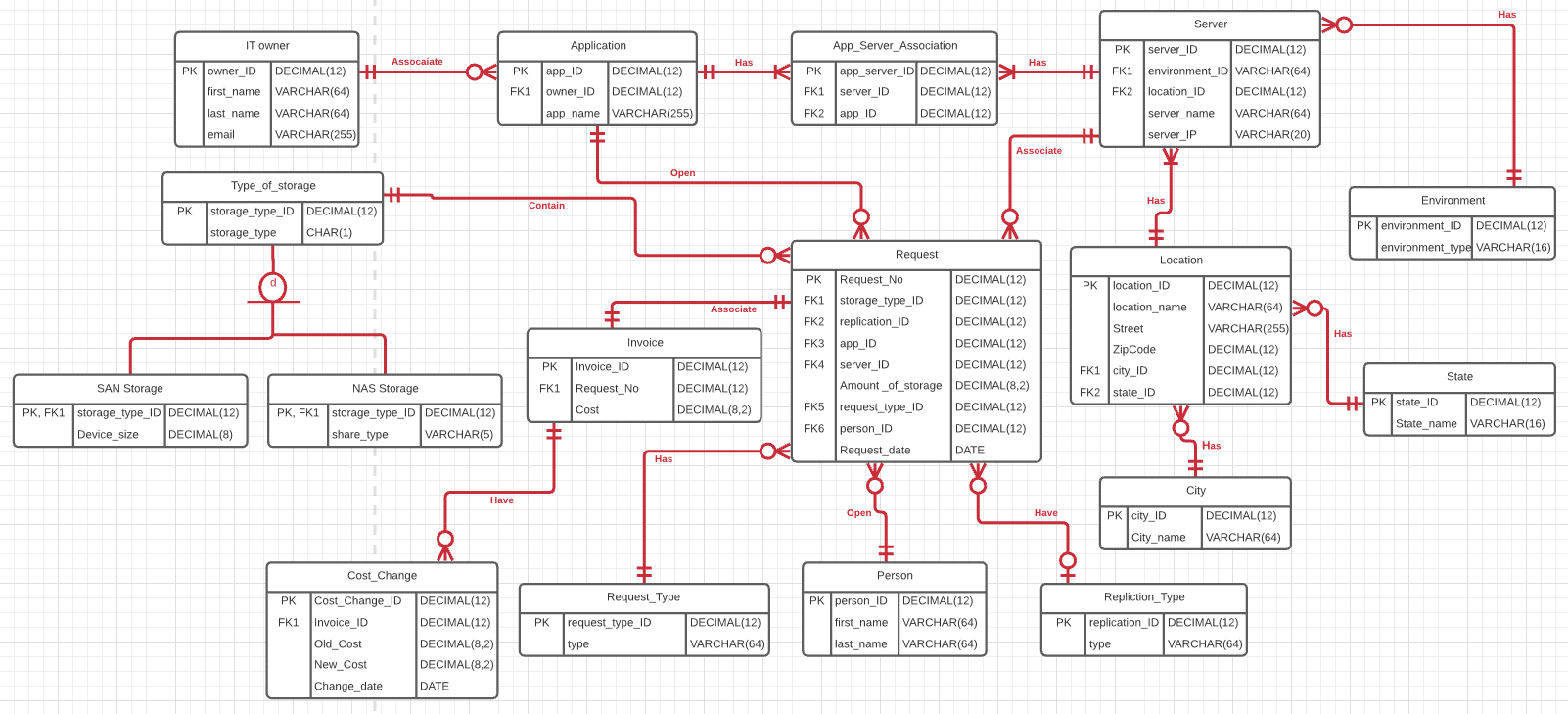
As a best practice I placed a synthetic primary key for each table (DECIMAL(12)) I created and based on the relationship classification I placed the foreign key as per the classification. Like for 1:1 relationship I placed the foreign key constraint in one of entity in the relationship that is for the Request/Invoice relationship I placed the foreign key (request\_no ) in the Invoice table. For the 1:M relationship I placed the foreign key constraint in the entity table with at most one (singular) relationship. Like for the Request/Type\_of\_storage, Request/Replication\_type, Request/Application, Request/Server, Request/Request\_type, Request/Person the foreign key constraint was placed in the Request entity table. Similarly, for Application/IT owner the foreign key was placed in the Application entity table. For the Server/Environment and Server/Location the foreign key constraint was placed in the Server entity table. For the Location/State and Location/City the foreign key constraint was placed in the Location table. Lastly for the M:N relationship Application/ Server there was a requirement to reify the relationship a new entity App\_Server\_Association was placed in the ERD to maintain the relationship. This new entity has the foreign keys to both the Application and Server entity resulting in a 1:M relationship between the App\_Server\_Assocation and Application and Server entities.

For the specialization generalization relationship that I have for my database design “The type of storage is SAN Storage, NAS Storage, or none of these” I have placed the primary key for the type\_of\_Storage which is the supertype as a foreign key for each of the subtype “SAN Storage” and “NAS Storage” which is also the primary key for each of the subtype as each subtype is the identity of the supertype.

Apart from the primary and foreign key constraints I have added the below attributes to each of the entity the datatype and reasoning for each is listed in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Entity | Attribute | Datatype | Reasoning |
| IT owner | first\_name | VARCHAR(64) | Every IT owner will have a firstname. I am allowing up to 64 characters for this. |
| IT owner | last\_name | VARCHAR(64) | Every IT owner will have a lastname. I am allowing up to 64 characters for this. |
| IT owner | Email | VARCHAR(255) | Each IT owner email will be added for contact information. I am allowing up to 255 characters for something exceptionally long and unusual. |
| Application | app\_name | VARCHAR(255) | Every Application will have a name associated with it. I am allowing for 255 characters for something exceptionally long and unusual. |
| Server | server\_name | VARCHAR(64) | Every Server will have a hostname (name) which will be hosted by allowing 64 characters for the attribute. |
| Server | server\_IP | VARCHAR(20) | Typically, IPv4 addresses are like 19 characters 16 digits and 3 dots so the I am allowing VARCHAR of up to 20 characters. |
| Environment | environment\_type | VARCHAR(16) | As the environment is like PROD, UAT or DEV so I am allowing 16 characters for the attribute. |
| Location | location\_name | VARCHAR(64) | Location name is held by this attribute, so I am allowing up to 64 characters. |
| Location | Street | VARCHAR(255) | This is the street address for the location allowing 255 characters for this incase there is a long street address. |
| City | City\_name | VARCHAR(64) | This is the city name so allowing 64 characters. |
| State | State\_name | VARCHAR(16) | This is the state name which are typically smaller than 16 characters. |
| Location | ZipCode | DECIMAL(12) | This is the zip code which holds the digits so allowing 12 decimals for this. |
| Type\_of\_storage | storage\_type | CHAR(1) | As there are at least two types of storage SAN Storage and NAS Storage so this attribute will identify what it is so allowing a single character for this. |
| Replication\_type | type | VARCHAR(32) | This attribute is the type of replication that is associated with this so allowing 32 characters for this. |
| Invoice | Cost | DECIMAL(8,2) | This attribute is for the cost of storage requested which cannot be in millions so allowing 8 digits with two decimal places. |
| Request | Amount\_of\_storage | DECIMAL(8,2) | As the amount of storage requested for each request cannot be more than 100TB for each so I am allowing 8 digits with two decimal places to be put in the Amount\_of\_storage field to be stored in GB of storage. |
| Request\_type | type | VARCHAR(32) | This is the type of request Addition or removal so allowing for 32 characters. |
| Person | first\_name | VARCHAR(64) | Every Person will have a firstname. I am allowing up to 64 characters for this. |
| Person | last\_name | VARCHAR(64) | Every Person will have a lastname. I am allowing up to 64 characters for this. |
| Request | Request\_date | DATE | This is the date when the request was put in, so it is DATE datatype. |
| Cost\_Change | Cost\_Change\_ID | DECIMAL(12) | This is the primary Key for the table so using DECIMAL (12) to allow for many values. |
| Cost\_Change | Old\_Cost | DECIMAL(8,2) | This is the cost of the Invoice before the change. The datatype for this is the same as Cost in the Invoice table. |
| Cost\_Change | New\_Cost | DECIMAL(8,2) | This is the cost of the Invoice after the change. The datatype for this is the same as Cost in the Invoice table. |
| Cost\_Change | Invoice\_ID | DECIMAL(12) | This is the foreign Key to the Invoice table a reference to the invoice with the change. |
| Cost\_Change | Change\_date | DATE | This is the date the cost change occurred with a DATE datatype. |

The final normalized DBMS physical ERD is shown below.



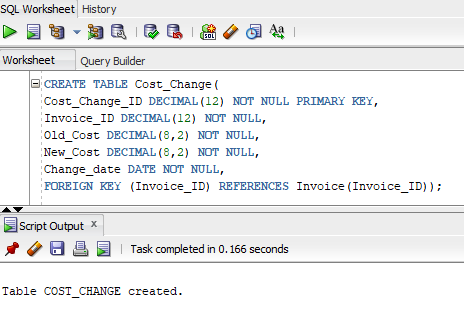
Using the above DBMS physical ERD I have defined the tables, sequences, columns, and constraints in SQL for my project I will be using Oracle SQL. I have created a script with all the CREATE TABLE and SEQUENCE commands. The SQL script is attached.

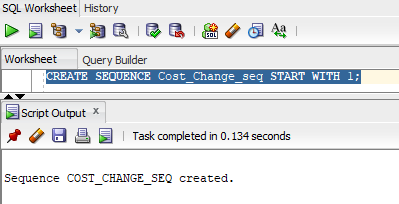
# History Table and Trigger

Update, if necessary, your history table and trigger which maintains it, and proof that the table and trigger work.

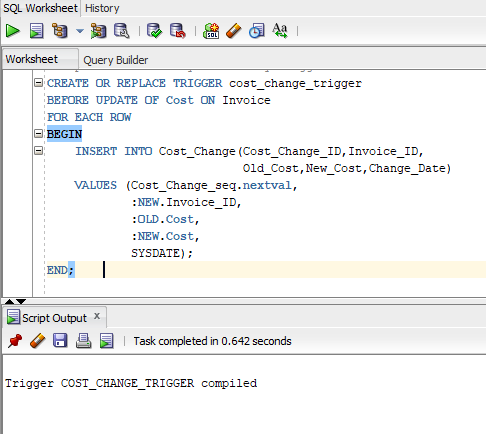
For my project I am adding a history table to the Invoice entity where the cost of each invoice will change over the period as there will be a change in the cost of storage per GB maybe my organization decide to charge more per GB of storage after a certain period so the Cost for each Invoice will need to be updated to reflect the new cost. For that purpose, I am adding a history table for the cost change to track any changes in cost and the date that change was made or occurred.

I have used the below CREATE TABLE and CREATE SEQUENCE statements to create the new history table and the sequence for the history table.





The trigger that maintains the history table is shown as below.

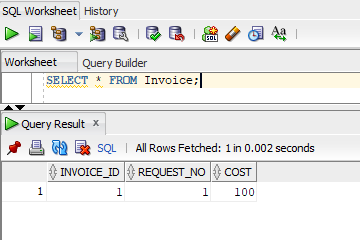


It will add a new record to the Cost\_Change table whenever there is an UPDATE to the Cost of the Invoice for each row.

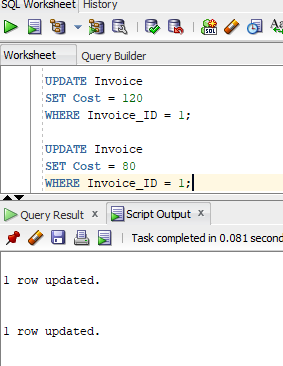
The code is explained line by line.

|  |  |
| --- | --- |
| CODE | DESCRIPTION |
| CREATE OR REPLACE TRIGGER cost\_change\_trigger BEFORE UPDATE OF Cost ON Invoice | This names the trigger “cost\_change\_trigger” and links it to the Invoice table. Further, it specifically indicates that the trigger will run before any update on Invoice table(ignoring deletes and inserts), and it only runs if the Cost column is updated. |
| FOR EACH ROW  BEGIN | This indicates that the trigger will run for each row that is updated which is necessary to get the specific invoices that get changed. |
| INSERT INTO Cost\_Change(Cost\_Change\_ID,Invoice\_ID,  Old\_Cost,New\_Cost,Change\_Date)  VALUES (Cost\_Change\_seq.nextval,  :NEW.Invoice\_ID,  :OLD.Cost,  :NEW.Cost,  SYSDATE); | This is a insert statement that records the cost change by adding a new row to the Cost\_Change table. The Cost\_Change\_ID column is updated using the Cost\_Change\_seq sequence. The Invoice\_ID is obtained from the :NEW pseudo table and the Old\_Cost and New\_Cost is obtained from the :OLD and the :NEW pseudo tables accordingly. The date is obtained using the built in SYSDATE variable. |
| END; | This ends the trigger. |

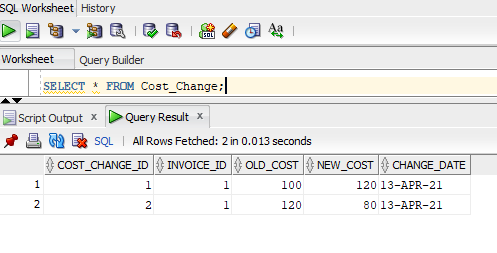
To test the trigger works I have added some value to the Invoice table as shown below.



I updated the value of the cost from 100 to 120 and down to 80 then as shown below.



When we look at the history table (Cost\_Change table) we can see that it has recorded two values in it for the two updates that we ran above.



# Index Identification and Creations

Update, if necessary, your indexes identifications useful to your database, explanations as to why they help, and screenshots of their creations.

For my database for the project, I know that I will be indexing the below:

1. All the primary keys are already indexed.

List of all the primary keys is as below which are already indexed and as these are unique, so it is unique index.

IT\_owner.owner\_ID

Application.app\_ID

App\_Server\_Association.app\_server\_ID

Server.server\_ID

Environment.environment\_ID

State.state\_ID

City.city\_ID

Location.location\_ID

Type\_of\_storage.storage\_type\_ID

SAN\_Storage.storage\_type\_ID

NAS\_Storage.storage\_type\_ID

Replication\_type.replication\_ID

Person.person\_ID

Request\_Type.request\_type\_ID

Request.Request\_No

Invoice.Invoice\_ID

Cost\_Change.Cost\_Change\_ID

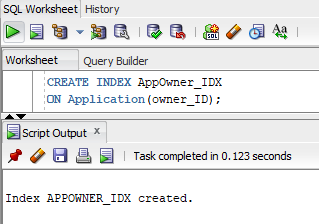
1. All the foreign keys for all the tables will need to be indexed.

The foreign keys that will need to index are shown below in the table along with the explanation of whether it will be unique index or not.

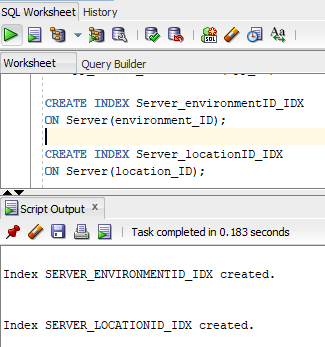
|  |  |  |
| --- | --- | --- |
| Foreign Key | Unique? | Description |
| Application.owner\_ID | Not Unique | The foreign key for the owner in Application table is not unique as an owner can own many application |
| App\_Server\_Association.server\_ID | Not Unique | The foreign key for the server in the table is not unique as the server can run many applications |
| App\_Server\_Association.app\_ID | Not Unique | The foreign key for the application in the table is not unique as an application can run many on many servers |
| Server.environment\_ID | Not Unique | The foreign key for the environment in the table is not unique as an environment can be associated with many servers |
| Server.location\_ID | Not Unique | The foreign key for the location in the table is not unique as a location can have many servers |
| Location.city\_ID | Not Unique | The foreign key for the city in the table is not unique as a city can have many locations |
| Location.state\_ID | Not Unique | The foreign key for the State in the table is not unique as a state can have many locations |
| Request.storage\_type\_ID | Not Unique | The foreign key for the Storage type in the table is not unique as a storage type can have many requests |
| Request.replication\_ID | Not Unique | The foreign key for the replication type in the table is not unique as a replication can have many requests |
| Request.app\_ID | Not Unique | The foreign key for an application in the table is not unique as an application can have many requests |
| Request.server\_ID | Not Unique | The foreign key for a server in the table is not unique as a server can have many requests |
| Request.request\_type\_ID | Not Unique | The foreign key for the request type in the table is not unique as a request type can have many requests |
| Request.person\_ID | Not Unique | The foreign key for the person in the table is not unique as a person can open many requests |
| Invoice.Request\_No | Unique | The foreign key for the Request in the table unique as one request will have one invoice |
| Cost\_Change.Invoice\_ID | Not Unique | The foreign key for the Invoice in the table is not unique as an Invoice can have multiple cost changes. |

1. I have identified the below three columns where query driven indexes that will need to be created.
   1. Request.Amount\_of\_Storage: There can be queries on the table for a certain amount of storage to find what requests were raised by which applications. This will be non-unique index because many requests can have same amount of storage requested.
   2. Request.Request\_date: There can be queries on the table to pull the data for the requests raised for a particular date so therefore require indexing based on that column. This will be a non-unique index as the same date will have many requests raised.
   3. Invoice.Cost: There can be queries on the Invoice table to pull the data for Invoices where the cost is exceeding a certain threshold or limit. This is non-unique index as many invoices can have the same cost associated.

The screen shots of the creation of the indexes are shown below.



# 



# 

# 

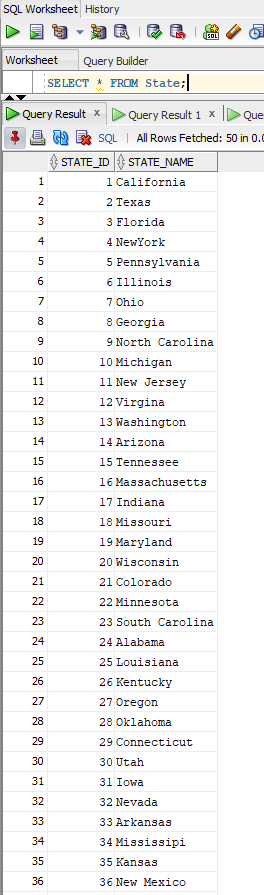
# 

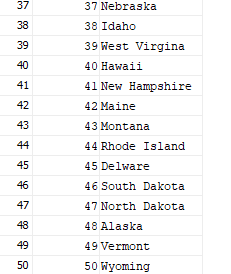
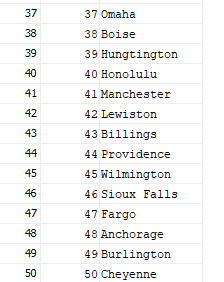
# Stored Procedure Execution and Explanations

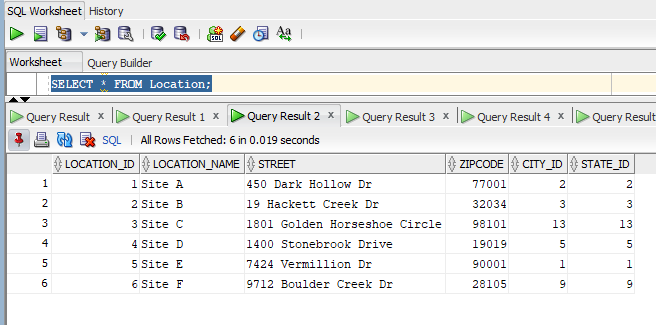
Replace this with screenshots and explanations of defining and executing your stored procedures to transactionally add data to your database.

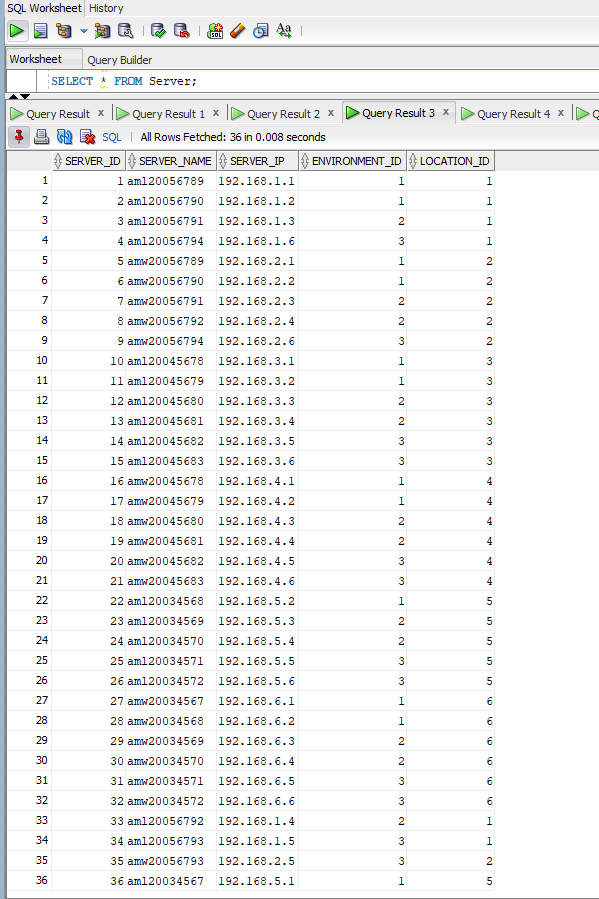
Before defining any stored procedure, I am predefining data into some of the tables like City, State, Location, Server, Replication\_Type, Request\_Type, Type\_of\_storage, NAS\_Storage, and SAN\_Storage.

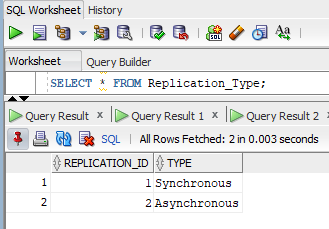
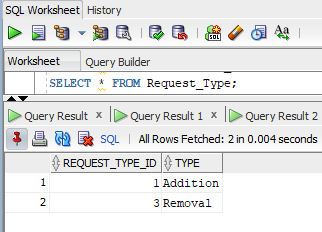
The data for each of them is shown in the below screenshots.

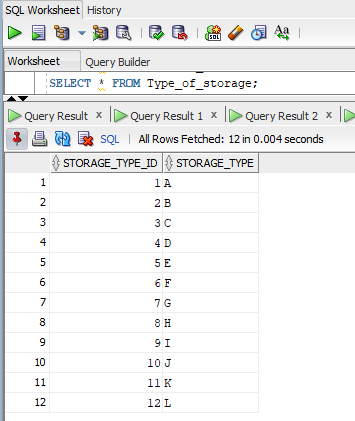
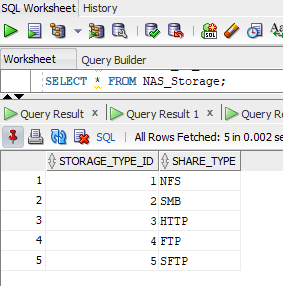
 

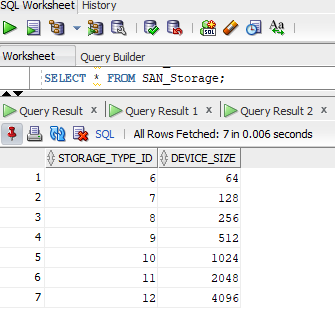
 





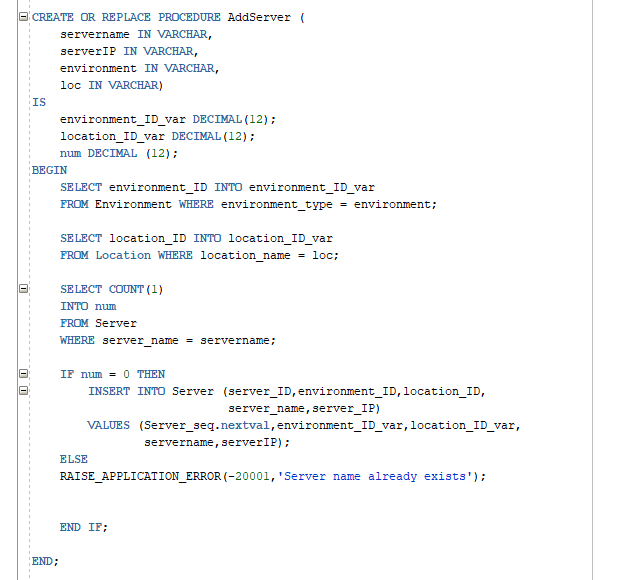
 

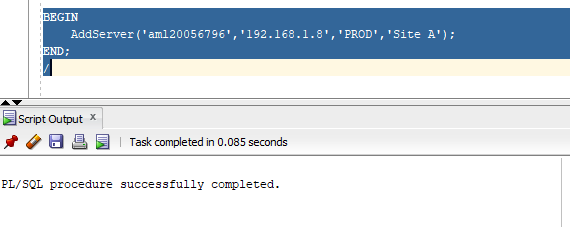


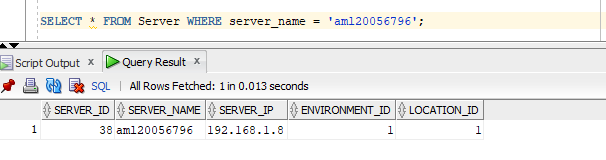
For my stored procedure for my database, I am creating three stored procedure named as below

* + - 1. AddServer this will be used to add server information into the Server table. This stored procedure takes 4 parameters as input which are server name , server IP address, environment the server belongs to and lastly the location the server is in. It selects the environment ID and the location ID which are foreign keys for the Server table by looking up the ID in the respective Environment and the Location table. The stored procedure contains mechanism to check if the server is already existing in the table if it is then it raises an error otherwise add the info to the table.

The screen shot for the stored procedure is shown below also an example of the execution is shown with the output of the table showing successful addition of the server in the table.



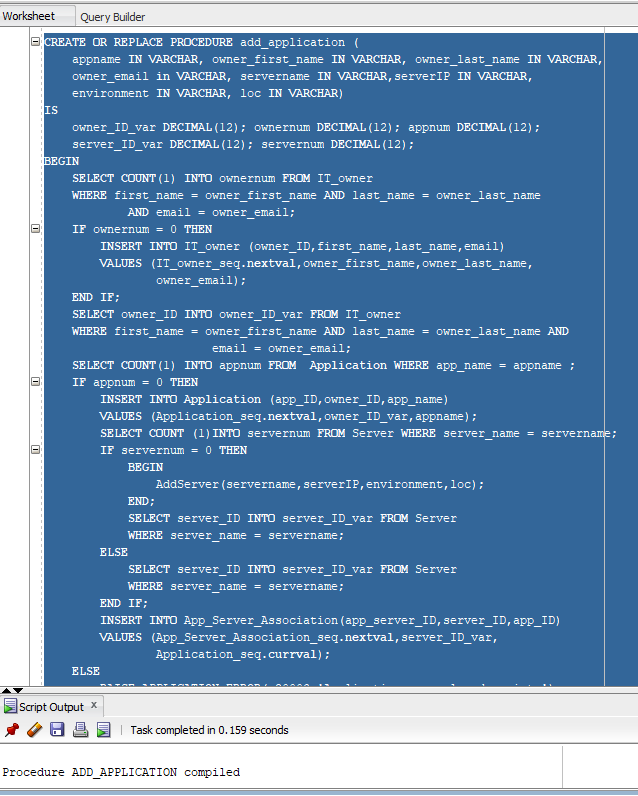


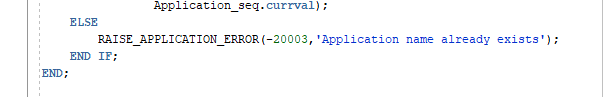


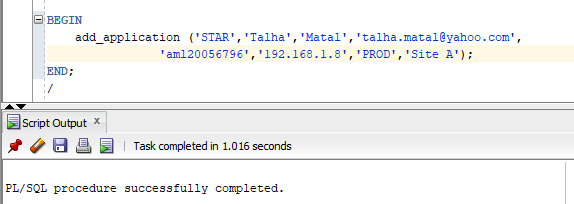
* + - 1. add\_application will be used with my use case of onboarding the application whenever a new application will need to onboard for my application for Online storage request. This stored procedure takes 8 parameters as input which are application name, IT owner first name, IT owner last name, IT owner email, server name , server IP address, environment the server belongs to and lastly the location the server is in. It first checks the IT owner table to see if the IT owner is already existing if it is then it fetches the ID for the owner which will be the foreign key for the Application table, if the owner does not exist it adds (inserts) the data int the IT owner table. Then it checks if the application name is already existing in the database table if it is raising an application error otherwise it walks through the steps of adding the application details within the database.

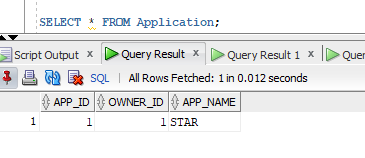
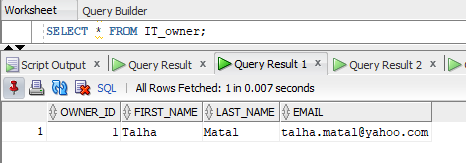
It adds the server into the Server database by calling the stored procedure above if the server is not already in the database. The stored procedure selects the server ID from the Server table using the server’s name and then populates the App\_Server\_Association table.

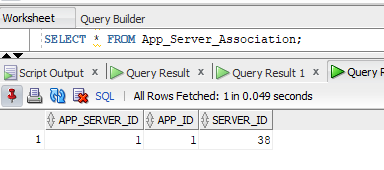
The screen shot for the stored procedure is shown below also an example of the execution is shown with the output of the table showing successful addition of the application in the table.

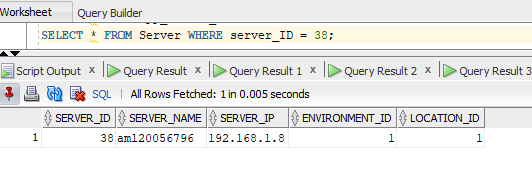






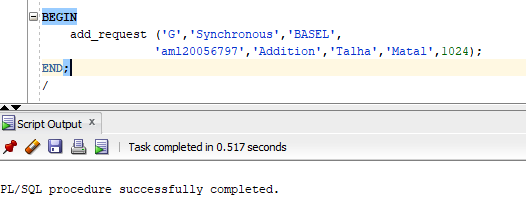
 

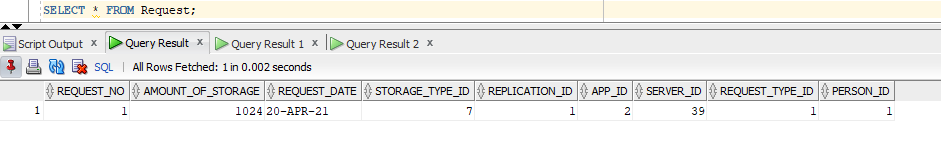


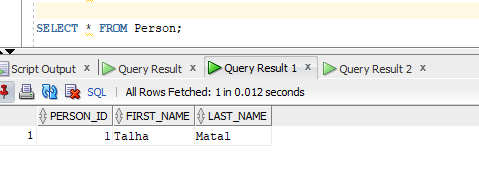


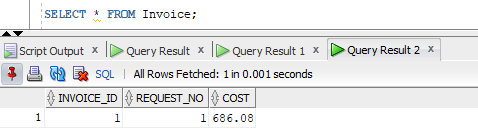
* + - 1. The third and final stored procedure for my database is called add\_request which is used to add the request information into the Request table. This stored procedure takes 8 parameters as input which are storage type, Replication type, application name requesting the storage, server for which the storage is requested, request type (Addition or removal), requestor first and last name and the amount of storage required. The stored procedure fetches the storage\_type\_ID, replication\_ID, app\_ID, server\_ID and the request\_type\_ID from the respective tables as these are all foreign keys to the Request table. It checks if the requestor person is already existing in the Person table if it is not it adds the Requestor details to the person table and fetches the person\_ID as it is the foreign key to the Request table. It inserts the details into the Request table also it finds the request date by using the SYSDATE function. The stored procedure also adds the cost of the requested storage to the Invoice table where it adds the Request no as a foreign key and the cost is calculated based on a value that is hardcoded to the procedure.

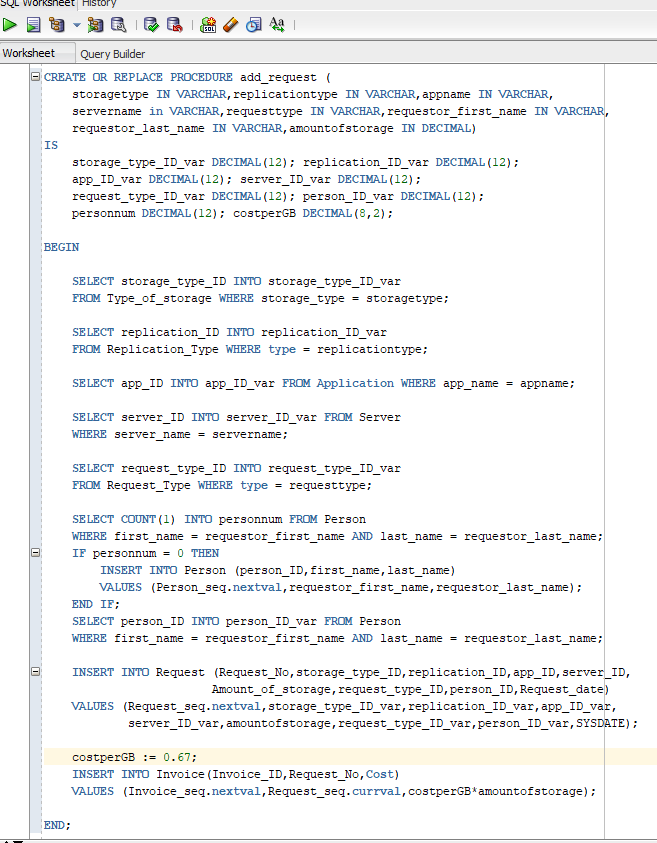
The screen shot for the stored procedure is shown below also an example of the execution is shown with the output of the table showing successful addition of the request in the table.











# Question Identification and Explanations

Replace this with questions useful to the organization and explanations as to why they are useful.

For my database, a few questions that are useful to the organization are as below.

1. What is the change in the cost of storage for an application over a specific period like past six months or so? This question is useful as it will give the idea as to what the change in the storage cost for an application over the past few months and the data is that can retrieve can be used by the application owners in determining the cost of storage of the application and doing analysis based on that and make decisions to order more storage or reduce the amount of storage so the cost can be reduced.
2. How many requests are raised for a particular type of storage and what is the total amount of storage requested for each storage type? This is another important question that is useful since it will give us an idea of how many requests are raised for a specific type of storage as it will help in analysis of what type of storage is in demand and what is not being used as much as it was anticipated before. This is also helpful in decision making process as to what storage to order more in future.
3. What is the amount of storage that is there allocated for each of the server? This is a useful question as it will help us in identifying the type of storage, amount of storage, replication type, and the cost of storage that is associated with each server so if we are in the process of decommissioning a server it can be analyzed how much storage will be removed and cost that will be saved from storage.
4. What are the requests that have been raised for an application and provide details for each of the requests like amount of storage, replication type, request type, requestor info, server name etc? This is an important question as it will help the application owner to lookup the data for the application as to how many requests were raised for the application and lookup details around each storage as to the type of storage, amount of storage etc.

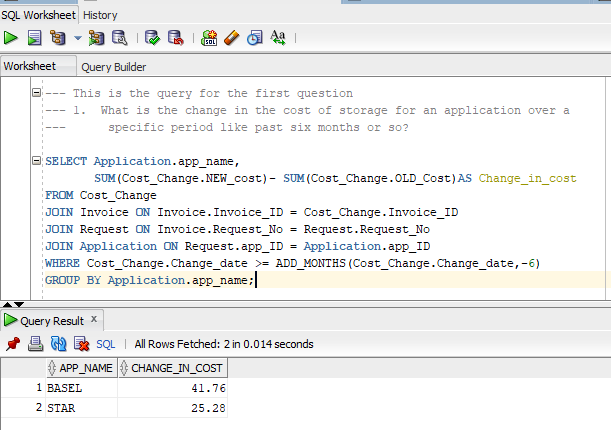
# 

# Query Executions and Explanations

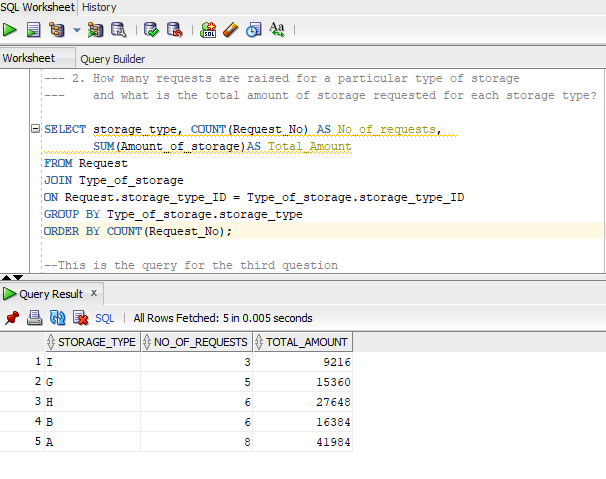
Replace this with queries answering the questions, along with screenshots and explanations.

Below are the screen shots for the queries answering the above questions and the explanations.

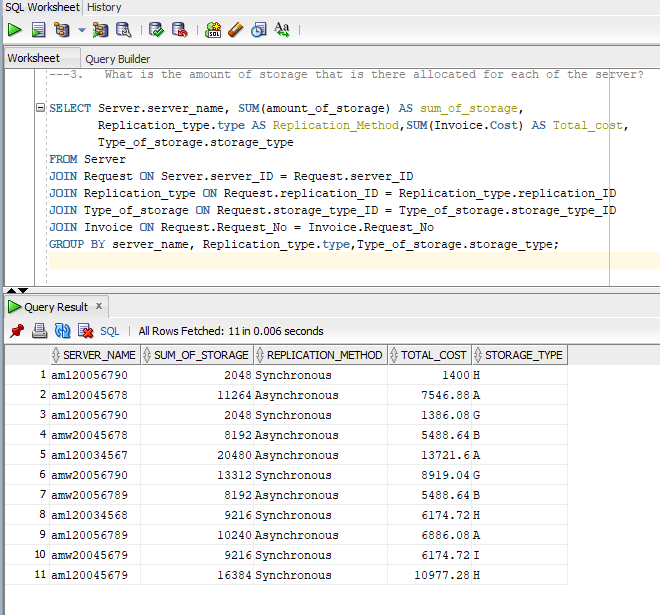
The below query answers the first question as described above. The query takes the Cost\_Change (History) table and joins with the Invoice, Request and Application tables and selects the application name and groups the information on the application name to find the difference between the old and the new cost for each of the groups by summing the total new cost and taking the difference from the total old cost.



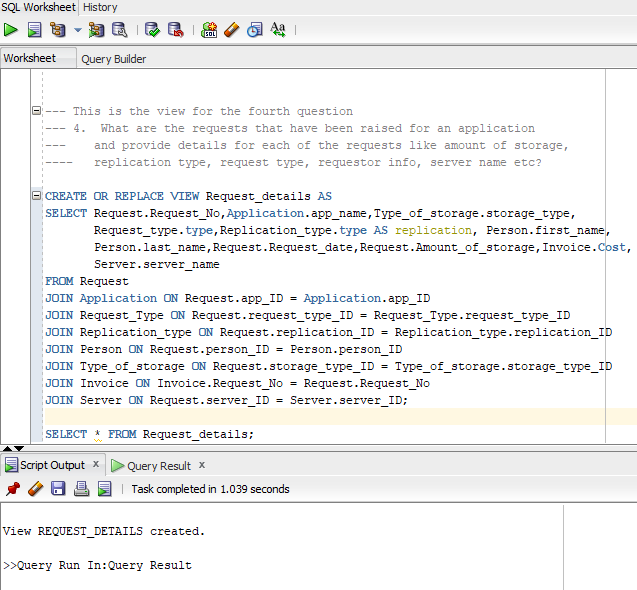
The below query answers the second question as described above. The query takes the Request table and joins with the Type of storage table, counts the number of requests using the COUNT function and finds the sum of Amount of storage for each group of Type of storage and then orders the result in Ascending based on the number of requests for each storage type.

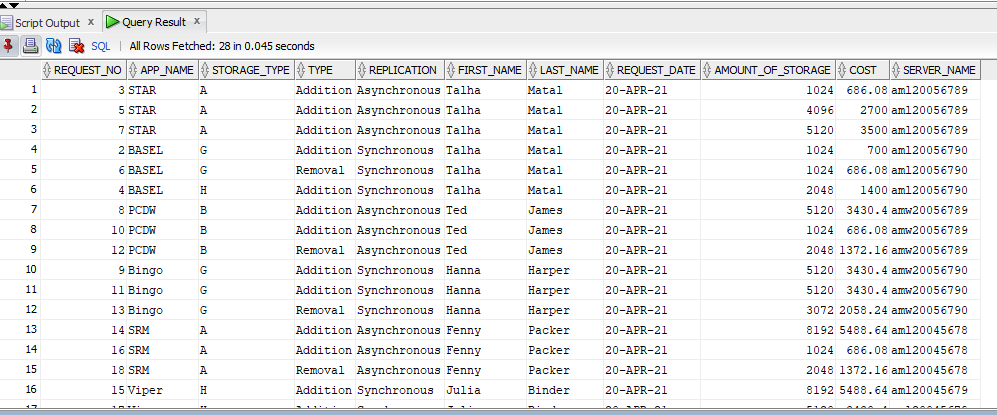


The below query answers the third question as described above. The query takes the Server table and joins with the Request, Replication\_type, Type\_of\_storage and Invoice tables finds the sum of amount of storage and the cost using the SUM function groups by server name, Type of storage and replication type to give the information for each server.



The fourth query is basically a view that defines or finds the information from 7 different tables into a single view so it can be used repeatedly as required and it addresses the needs for the database as it helps reusability and easier access to information. The view is basically joining the Request, Application, Request\_type, Replication\_type, Person, Type\_of\_storage, Invoice and Server table on the appropriate fields as shown below. This gives a single pane of view for a database.

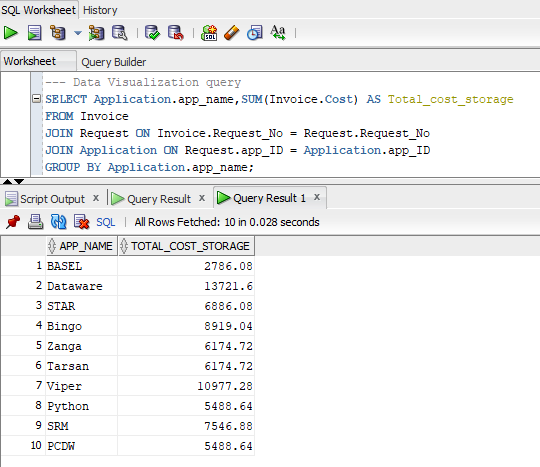




# Data Visualizations

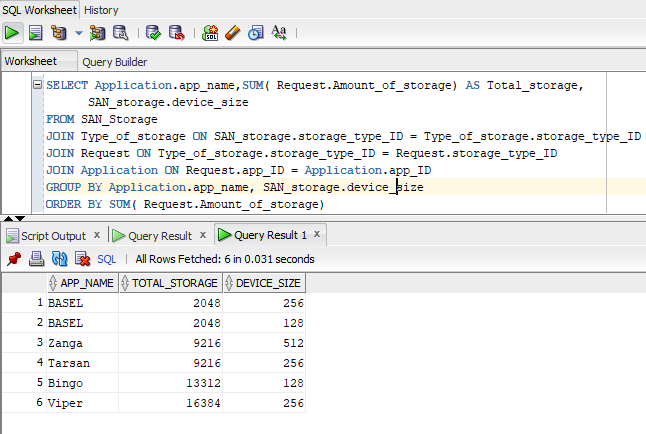
Replace this with effective data stories and visualizations.

For this part of I have selected the below query to find the total cost of storage for each of the application that have requested storage based using the application. As there is a single measure total cost of storage so I am using a bar char to represent the data visualization.



This chart gives us an important information that the biggest consumer of storage is the Dataware application which can lead us to the information that the Dataware application is a capacity requiring application which tells us that maybe it is an archival based application with high storage demand. Whereas on the other hand BASEL is least consuming application leading us to believe that it is not high capacity demanding.

For the second data visualization I am running the below query to find the device size and the total amount of storage used by each application. As there are two measures total storage amount and device size so I will be using a scatter plot to represent the visualization.



This chart shows that the representation between the device size and the total SAN storage that is in use by each of the application. This visualization gives us some important information it represents that the Viper application has the most amount of SAN storage that it consumes but the device size for each of the logical devices it uses is not large it is medium sized devices. Whereas although Zanga application uses the largest size of devices but the amount of storage it consumes is not exceptionally large.

# Summary and Reflection

Update the concise summary of your project and the work you have completed thus far, and additionally update your questions, concerns, and observations, so that you and your facilitator or instructor are aware of them and can communicate about them.

The database that I am planning to design for my project contains the data for a web application for the purpose of onboarding a new application to be able to request online storage, remove online storage, track the requests, find the cost associated with those requests and the overall cost associated with the storage that is allocated for that application. The database will have fields for the application name, IT owner , contact details, request number for the storage, amount and type of storage required, server the storage is requested for, replication type, environment, location the storage is required in, and the cost associated with each request.

The structural database rules and the ERD that I have created for my database design contains entities like Application, IT owner, Request, Server, Location, Type of storage, Replication type and the Invoice associated with the request. I have defined database rules for these entities and represented them diagrammatically using crow’s foot representation. I have also defined the specialization – generalization rules which have subtypes for Type of storage. The DBMS physical ERD for my database design shows the same entities and relationships with the use of synthetic primary keys for each table and the attributes required for each of the entities.

Based on the DBMS physical ERD I have written down the SQL for creating my tables with the columns as defined with the attributes in the DBMS physical ERD. I have created sequences for all the tables for the primary keys and indexed all the foreign keys along with any query driven columns which I identified can be used in WHERE and JOIN clause. I have also created and defined a history table called Cost\_Change which updates whenever there is an update to the cost in the Invoice table. The SQL script defines a trigger which is invoked whenever there is an update to the Invoice table. I have created stored procedures to insert data into the database and populated the database with data. Some useful questions have been answered using the queries that I have written in SQL.

Reflecting over the project and the accomplishments so far I think so I have come a long way by designing and implementing my own database. There are still a lot of improvements and enhancements that can be made to my database.