**System Analysis**

KYAW THIHA NAING

Firstly, data accessibility is greatly improved for the CRS users incorporating effective sharing of data to other users selectively. The CRS organization's workflow and operations will also be more apparent in a clear picture as it would be easier to realize how a certain segment of the organization affects the other interconnected segment. Furthermore, the CRS organization will be able to effectively manage data related to users like their documents, personal information, and application information of volunteers etc. A database management system will also allow users to selectively filter out relevant data without going through an overwhelming amount of data that is being stored. Additionally, due to information with good quality being produced readily by the database management system, it will result in the CRS organization's decision-making to be much better and quicker. This in turn leads to an increase in productivity of the CRS users, especially the staff.

When it comes to differences between traditional files and database systems, they tend to be in terms of their functionality and usage. A few examples include how the security system is better in database systems and how there's a data backup recovery function for database systems while there is none for traditional file systems. However, there are also some advantages in using traditional file systems notably in how it costs less and how it is simpler compared to database systems. It is also important to not overlook how database systems offer so much more when it comes to flexibility and effectiveness. According to the BioModels database Journal, the database system can be customized to have automated tasks like for instance checking the data for consistency in a serial manner which in turn enforces the validity of the data model. There are also some other features mentioned regarding how different types of models are retrievable just based on the associated attributes and elements like id, name etc. However, it also mentions a downside related to fast retrieval of data in terms of how the user needs to know the exact identifiers that are created and used by the data curators to annotate the model. Another downside being mentioned is how their database model's improvements are limited due to many data curators needed (Park, Yoon, Koo, Yoo, Choi, Beck, & Kim, 2018). This implies that without enough workforce the database's full potential cannot be utilized. On the other hand, hiring more people to increase the workforce can also result in more costs for the company.

For challenges in database system implementation, the primary consideration for standardized requirements of OLTP, industry standard benchmarks and complex query processing are being considered in order to choose the hardware with high performance. Even though the hardware prices are rapidly falling, robustness and decrease of ease in maintenance would become an issue as most embedded systems just typically perform simple queries and not necessarily require high performing hardware (Suhr, Chang, Shaw, & Lee, 2020). So, it is important to understand the business framework and requirements in order to decide on what kind of hardware to invest on without creating the necessity for extra workforce in maintenance. Another challenge being highlighted is the automatization of certain tasks in order for users to trust the database system and easily use it without the need for them to perform or take part in any system administration. Some of the tasks being described are data reorganization, and backup (Suhr, Chang, Shaw, & Lee, 2020). Data reorganization is necessary in a way to avoid unnecessary database inflation. As new data is updated very frequently in business organizations, data reorganization needs to be automated in order to reduce cost and not hire more people to do it manually. Backup is necessary in order to recover lost data in case something happens. When dealing with many users, it is not possible to manually backup their individual bank of data which is why it needs to be automated as well.

KIM SANGA

Company employees can exchange information quickly and accurately. Although there would be multiple simultaneous staff accesses to the database to modify the database, the information processing results are always accurate and up to date. An organization like CRS must help and support the victims swiftly in the event of a natural disaster, thus an efficient database system can accurately locate the disaster and quickly notify the volunteers

A file processing system is a file management system without hierarchical order that governs the relationship between files in a computer. The most significant advantage of a flat file system is its ease of usage, and it does not require complicated IT technology. It is best suited for generating small amounts of data and can be processed quickly and easily. Another advantage is the cost of creating and maintenance. Licensing and maintenance costs are much lower than DBMS. It can be created without requiring specialized skills. Thus, it reduces the cost of hiring IT professionals (Gröger, 2018). Unfortunately, there are disadvantages to this system. Because the file is a fixed-format file, thus a single modification can damage the data. The biggest drawback is data redundancy, which results in unnecessary data and takes up unnecessary storage space. This redundancy poses another problem which is the difficulty of system update. Since updates are made through many records and data, errors are more likely than other methods (Gröger, 2018). Unlike flat file systems, DBMS controls data redundancy by consolidating and organizing data. Besides, it can be publicized because it can support the same data in different structures. A single DBMS can effectively control its databases' management and access because it centrally manages and oversees its databases. Verification of legitimate users can ensure tight security for all data. However, there are also disadvantages: increased operating costs, the complexity of specific applications, and system vulnerabilities. Several different types of data are related to each other in the database. Certain programs can be written and performed with several limitations under these circumstances. Special purpose application systems may experience performance degradation because they require more extended design periods and need to be more professional and technical. Furthermore, since database systems are integrated systems, some failures can disrupt system reliability and availability by shutting down the entire system. As a result, it can be a fatal weakness in environments with high protocol in the database.

Not to mention the database becomes a very attractive target for hackers(Toapanta, Escalante Quimis, Gallegos & Maciel Arellano, 2020). The database contains sensitive and valuable information. This ranges from property rights, financial enterprise data to personal user data. Cybercriminals can hack and damage databases and breakthrough corporate servers to earn profits. Therefore, a database security test is a must. The second challenge will be the cost, as mentioned above. With DBMS, high-speed processors and high-capacity memory are fundamental because large amounts of data are present, which requires expensive hardware and expensive software. Both hardware and software require maintenance including all operations, training (at all levels, including programming, application development, and database management), licensing, and compliance. These are high cost in upkeeping.

**Logical Design**

**CRS table**

|  |  |
| --- | --- |
| CRS | crsID |
| Primary Key | crsID |
| Foreign Key | - |

**User table**

|  |  |
| --- | --- |
| User(supertype) | userID, password, first\_name, last\_name, phone, crsID |
| Primary Key | userID |
| Foreign Key | crsID REFERENCES CRS (crsID) |

**Volunteer table**

|  |  |
| --- | --- |
| Volunteer(subtype) | userID, password, first\_name, last\_name, phone, crsID |
| Primary Key | userID |
| Foreign Key | crsID REFERENCES CRS (crsID) |

**CRS staff table**

|  |  |
| --- | --- |
| CRS\_staff(subtype) | userID, password, first\_name, last\_name, phone, position, date\_joined, crsID |
| Primary Key | userID |
| Foreign Key | crsID REFERENCES CRS (crsID) |

**Document table**

|  |  |
| --- | --- |
| Document | docID, doc\_type, expiry\_date, image, volunteerID |
| Primary Key | docID |
| Foreign Key | volunteerID REFERENCES volunteer (userID) |

**Trip table**

|  |  |
| --- | --- |
| Trip | tripID, trip\_date, destination, crisis\_type, req\_vol, emergency\_phone, staffID, crsID |
| Primary Key | tripID |
| Foreign Key | staffID REFERENCES CRS\_staff (userID)  crsID REFERENCES CRS (crsID) |

**Application table**

|  |  |
| --- | --- |
| Application | applicationID, application\_date, status, remarks, staffID, volunteerID, tripID |
| Primary Key | applicationID |
| Foreign Key | staffID REFERENCES CRS\_staff (userID)  volunteerID REFERENCES volunteer (userID)  tripID REFERENCES Trip (tripID) |

**Submission Record**

|  |  |
| --- | --- |
| Submission\_record | docID, applicationID |
| Primary Key | docID, applicationID |
| Foreign Key | docID REFERENCES document (docID)  applicationID REFERENCES Application (applicationID) |

**Relations identified:**

**KYAW THIHA NAING**

Each CRS director may record many users, but one user must be recorded by one CRS director. (1-M)

Each CRS director may organize many trips, but each trip must be organized by one CRS director. (1-M)

Each staff may handle many trips, but each trip must be handled by one staff (1-M)

Each volunteer may upload one or many documents, but each document must only be uploaded by one volunteer (1-M)

Each volunteer may submit zero or many applications, but each application must only be submitted by one volunteer (1-M)

**KIM SANGA**

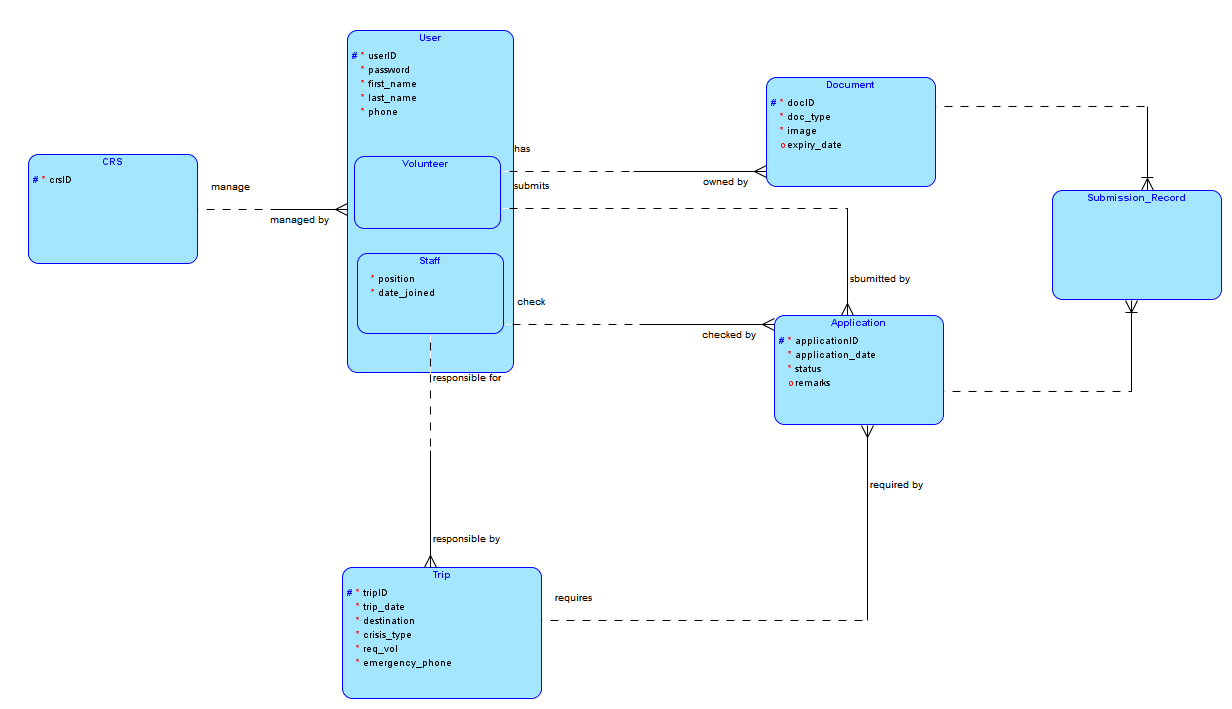
One volunteer may sign up for many trips and one trip may also be signed up by many volunteers [Application acts as an intersection entity] (M-M)

One trip can have many applications, but one application is only for one trip. (1-M)

One staff may handle many applications, but one application must only be handled by one staff (1-M)

One application may have multiple documents and one document can be submitted to multiple applications too. [Submission record is created as an intersection entity] (M-M)

**Physical Database Design**

****

Picture 1: CRS ERD Diagram

**ii.**

For implementing the supertype-subtype scenario of user, volunteer, and staff, we used the two-table method as there are more relationships on the subtype level compared to the supertype level.

**CRS table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| CRS | The CRS table stores the information of the board of directors who keeps the record of staff and volunteers. There is no additional information being provided for the CRS board of directors, resulting in only the primary key being present in the table | crsID primary key not null |
| - |
| - |

**Volunteer table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| Volunteer | Volunteer table stores the personal information of volunteers who are categorized as users. The volunteer table retains only the general information shared by all users and has no other attributes. | volunteerID primary key not null |
| - |
| crsID foreign key references CRS(crsID) on delete cascade |

**CRS staff table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| CRS\_staff | CRS\_staff table stores the personal information of staff from the CRS organization who falls under the user category. The staff table not only retains the general information shared by all users but also has two additional attributes namely position and data joined | staffID primary key not null |
| - |
| crsID foreign key references CRS(crsID) on delete cascade |

**Document table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| Document | Document table stores several information on the documents submitted by volunteers. Some of the attributes for this table are optional as they depend on the type of document being submitted. Images are also stored as volunteers have to submit them to prove their skills | docID primary key not null |
| document\_type check value IN (“PASSPORT”, “CERTIFICATE”, “VISA”) |
| volunteerID foreign key references Volunteer(volunteerID) on delete cascade |

**Trip table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| Trip | Trip table stores the information related to the trips organized by the CRS and handled by its staff. All the information stored in the trip table is mandatory to ensure smooth workflow of the CRS organization. The types of crisis are limited to certain values | tripID primary key not null |
| crisis\_type check value IN (“FLOOD”, “EARTHQUAKE”, “WILDFIRE”) |
| staffID foreign key references Staff(staffID) on delete cascade  crsID foreign key references CRS(crsID) on delete cascade |

**Application table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| Application | Application table stores information on applications made by volunteers when applying for trips. The trips being applied along with supporting document for the application are stored in this table in the form of foreign attributes | applicationID primary key not null |
| status check value IN (“NEW”, “APPROVED”, “REJECTED”) |
| staffID foreign key references Staff(staffID) on delete cascade  tripID foreign key references Trip(tripID) on delete cascade  volunteerID foreign key references Volunteer(volunteerID) on delete cascade |

**Submission record table**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Constraints** |
| Submission\_ record | Submission\_record table stores information on documents submitted for specific applications as multiple documents correspond to multiple applications at the same time | docID primary key not null  applicationID primary key not null |
| - |
| docID foreign key references Document(docID) on delete cascade  applicationID foreign key references application(applicationID) on delete cascade |

**Implementation with SQL**

|  |  |
| --- | --- |
| Name | KIM SANGA |
| List of tables | CRS |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE CRS (  crsID VARCHAR (10) NOT NULL,  CONSTRAINT crc\_PK PRIMARY KEY (crsID)  );  INSERT INTO CRS VALUES('C0001');  INSERT INTO CRS VALUES('C0002');  INSERT INTO CRS VALUES('C0003');  INSERT INTO CRS VALUES('C0004');  INSERT INTO CRS VALUES('C0005');  INSERT INTO CRS VALUES('C0006');  INSERT INTO CRS VALUES('C0007');  INSERT INTO CRS VALUES('C0008');  INSERT INTO CRS VALUES('C0009');  INSERT INTO CRS VALUES('C0010'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KIM SANGA |
| List of tables | CRS\_Staff |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE CRS\_staff(  staffID VARCHAR (10) NOT NULL,  crsID VARCHAR (10) NOT NULL,  password VARCHAR (20) NOT NULL,  first\_name VARCHAR (10) NOT NULL,  last\_name VARCHAR (10) NOT NULL,  phone VARCHAR (30) NOT NULL,  position VARCHAR (20) NOT NULL,  date\_joined DATE NOT NULL,  CONSTRAINT crs\_staff\_PK PRIMARY KEY (staffID) ,  CONSTRAINT staff\_crs\_FK FOREIGN KEY (crsID) REFERENCES CRS(crsID) ON DELETE CASCADE  );  INSERT INTO CRS\_staff VALUES('S0001','C0001','alfldnle','Barbara','Smith','0167569436','Manager',DATE'2001-05-24');  INSERT INTO CRS\_staff VALUES('S0002','C0001','fda3d2f','Mike','Smith','0136458921','Team Leader',DATE'2005-05-24');  INSERT INTO CRS\_staff VALUES('S0003','C0002','fd1a3d1','Heily','Kim','0145397561','IT Support',DATE'2009-07-04');  INSERT INTO CRS\_staff VALUES('S0004','C0003','aiefsdf54','Sam','Kyaw','0964522310','IT Support',DATE'2003-07-29');  INSERT INTO CRS\_staff VALUES('S0005','C0004','fd1a3dfs5','Henry','Brown','0336599325','Manager',DATE'2008-02-24');  INSERT INTO CRS\_staff VALUES('S0006','C0005','fdagbf3d','Kathrine','Jang','0117555432','Team Leader',DATE'2007-10-01');  INSERT INTO CRS\_staff VALUES('S0007','C0009','yhkgd23','Emy','White','0114558793','IT Support',DATE'2007-06-24');  INSERT INTO CRS\_staff VALUES('S0008','C0004','231refsdfs','Alice','Swift','0635598732','Assistant',DATE'2013-06-24');  INSERT INTO CRS\_staff VALUES('S0009','C0006','fdad1fs321','Joel','Ng','0145578302','Assistant',DATE'2011-05-24');  INSERT INTO CRS\_staff VALUES('S0010','C0007','kyfdhd23','Sara','Lee','0123697589','Assistant',DATE'2019-08-24'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KIM SANGA |
| List of tables | Volunteer |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE Volunteer (  volunteerID VARCHAR (10) NOT NULL,  crsID VARCHAR (10) NOT NULL,  password VARCHAR (20) NOT NULL,  first\_name VARCHAR (10) NOT NULL,  last\_name VARCHAR (10) NOT NULL,  phone VARCHAR (30) NOT NULL,  CONSTRAINT volunteer\_PK PRIMARY KEY (volunteerID),  CONSTRAINT vol\_crs\_FK FOREIGN KEY (crsID) REFERENCES CRS(crsID)ON DELETE CASCADE  );  INSERT INTO VOLUNTEER VALUES('V0001','C0003','dtver','Selly','Smith','0132225684');  INSERT INTO VOLUNTEER VALUES('V0002','C0008','dfa3fd','Bily','Mark','0132666975');  INSERT INTO VOLUNTEER VALUES('V0003','C0003','Q3WRD','Sheirley','Park','0145574632');  INSERT INTO VOLUNTEER VALUES('V0004','C0004','fhswts','Bella','Goth','0133458552');  INSERT INTO VOLUNTEER VALUES('V0005','C0009','fhgvj3455','Johnson','Baby','1035569754');  INSERT INTO VOLUNTEER VALUES('V0006','C0005','agrja3gr','Oliver','Taylor','0.15665412');  INSERT INTO VOLUNTEER VALUES('V0007','C0004','fkdtsfadsef','Jack','Wilson','0114457360');  INSERT INTO VOLUNTEER VALUES('V0008','C0001','garau4567','Jacob','Evans','0102554785');  INSERT INTO VOLUNTEER VALUES('V0009','C0002','razhf34','William','Thomas','0123456789');  INSERT INTO VOLUNTEER VALUES('V0010','C0004','srz86t4','Oscar','Roberts','0198745632'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KIM SANGA |
| List of tables | Trip |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE Trip (  tripID VARCHAR (20) NOT NULL,  staffID VARCHAR (20) NOT NULL,  crsID VARCHAR (10) NOT NULL,  trip\_date DATE NOT NULL,  destination VARCHAR (20) NOT NULL,  crisis\_type VARCHAR (20) NOT NULL,  req\_vol NUMBER(5) NOT NULL,  emergency\_phone VARCHAR (20) NOT NULL,  CONSTRAINT trip\_PK PRIMARY KEY (tripID),  CONSTRAINT trip\_staff\_FK FOREIGN KEY (staffID) REFERENCES CRS\_staff(staffID) ON DELETE CASCADE,  CONSTRAINT trip\_crs\_FK FOREIGN KEY (crsID) REFERENCES CRS(crsID) ON DELETE CASCADE  );  INSERT INTO TRIP VALUES ('T0001','S0001','C0001',DATE'2021-03-01','Korea','FLOOD','50','0111234567');  INSERT INTO TRIP VALUES ('T0002','S0001','C0002',DATE'2021-05-01','Japan','EARTHQUAKE','100','0123548320');  INSERT INTO TRIP VALUES ('T0004','S0008','C0009',DATE'2021-06-06','Colombia','EARTHQUAKE','50','01122469532');  INSERT INTO TRIP VALUES ('T0005','S0003','C0008',DATE'2021-04-08','Comoros','Tsunamis','150','0235645393');  INSERT INTO TRIP VALUES ('T0006','S0005','C0004',DATE'2021-03-01','Congo','FLOOD','50','0132655952');  INSERT INTO TRIP VALUES ('T0007','S0009','C0007',DATE'2021-03-06','Malaysia','FLOOD','50','1023544585');  INSERT INTO TRIP VALUES ('T0008','S0010','C0006',DATE'2021-03-21','Africa','EARTHQUAKE','150','82645596531');  INSERT INTO TRIP VALUES ('T0009','S0004','C0006',DATE'2021-03-11','Mali','FLOOD','50','6015548963');  INSERT INTO TRIP VALUES ('T0010','S0002','C0010',DATE'2021-04-01','Mozambique','EARTHQUAKE','100','1025568632');  INSERT INTO TRIP VALUES ('T0003','S0002','C0005',DATE'2020-05-04','Australia','WILDFIRE','100','0225675563'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KYAW THIHA NAING |
| List of tables | Document |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE Document(  docID VARCHAR (20) NOT NULL,  volunteerID VARCHAR (10) NOT NULL,  doc\_type VARCHAR (15) NOT NULL,  expiry\_date DATE,  image VARCHAR (30) NOT NULL,  CONSTRAINT doc\_PK PRIMARY KEY (docID),  CONSTRAINT doc\_vol\_FK FOREIGN KEY (volunteerID) REFERENCES VOLUNTEER (volunteerID) ON DELETE CASCADE,  CONSTRAINT doctype\_check CHECK (doc\_type IN ('PASSPORT','CERTIFICATE', 'VISA'))  );  INSERT INTO Document VALUES  ('D0001', 'V0002', 'PASSPORT', DATE'2025-07-04', 'THIS IS A PASSPORT');  INSERT INTO Document VALUES  ('D0002', 'V0002', 'CERTIFICATE', NULL, 'SWIMMING CERTIFICATE');  INSERT INTO Document VALUES  ('D0003', 'V0001', 'PASSPORT', DATE'2026-08-04', 'THIS IS ANOTHER PASSPORT');  INSERT INTO Document VALUES  ('D0004', 'V0001', 'VISA', NULL, 'VISITOR VISA FOR THAILAND');  INSERT INTO Document VALUES  ('D0005', 'V0005', 'PASSPORT', DATE'2024-08-06', 'A RED PASSPORT');  INSERT INTO Document VALUES  ('D0006', 'V0005', 'CERTIFICATE', NULL, 'CERTIFICATE IN SOCIAL SERVICES');  INSERT INTO Document VALUES  ('D0007', 'V0007', 'PASSPORT', DATE'2024-08-14', 'A GREEN PASSPORT');  INSERT INTO Document VALUES  ('D0008', 'V0007', 'VISA', NULL, 'VISITOR VISA FOR GAMBIA');  INSERT INTO Document VALUES  ('D0009', 'V0004', 'PASSPORT', DATE'2022-10-04', 'THIS IS ALSO ANOTHER PASSPORT');  INSERT INTO Document VALUES  ('D0010', 'V0004', 'CERTIFICATE', NULL, 'ACROBATICS CERTIFICATE'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KYAW THIHA NAING |
| List of tables | Application |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE Application (  applicationID VARCHAR (20) NOT NULL,  volunteerID VARCHAR (10) NOT NULL,  staffID VARCHAR (10) NOT NULL,  tripID VARCHAR (20) NOT NULL,  application\_date DATE NOT NULL,  status VARCHAR (20) NOT NULL,  remarks VARCHAR (50),  CONSTRAINT app\_PK PRIMARY KEY (applicationID),  CONSTRAINT app\_vol\_FK FOREIGN KEY (volunteerID) REFERENCES VOLUNTEER (volunteerID) ON DELETE CASCADE,  CONSTRAINT app\_staff\_FK FOREIGN KEY (staffID) REFERENCES CRS\_staff(staffID) ON DELETE CASCADE,  CONSTRAINT app\_trip\_FK FOREIGN KEY (tripID) REFERENCES TRIP(tripID) ON DELETE CASCADE,  CONSTRAINT status\_check CHECK (status IN('NEW', 'APPROVED', 'REJECTED'))  );  INSERT INTO Application VALUES  ('A0001', 'V0002', 'S0001', 'T0001', DATE'2021-1-3','NEW', NULL);  INSERT INTO Application VALUES  ('A0002', 'V0002', 'S0003', 'T0002', DATE'2021-1-3','REJECTED', 'Not eligible');  INSERT INTO Application VALUES  ('A0003', 'V0001', 'S0004', 'T0003', DATE'2021-1-13','NEW', NULL);  INSERT INTO Application VALUES  ('A0004', 'V0003', 'S0005', 'T0001', DATE'2021-1-6','APPROVED', 'Welcome on board!');  INSERT INTO Application VALUES  ('A0005', 'V0004', 'S0002', 'T0005', DATE'2021-1-18','NEW', NULL);  INSERT INTO Application VALUES  ('A0006', 'V0008', 'S0005', 'T0006', DATE'2021-1-23','APPROVED', 'Hope to see you soon!');  INSERT INTO Application VALUES  ('A0007', 'V0007', 'S0004', 'T0001', DATE'2021-1-7','NEW', NULL);  INSERT INTO Application VALUES  ('A0008', 'V0004', 'S0008', 'T0002', DATE'2021-1-19','REJECTED', 'Not enough supporting documents');  INSERT INTO Application VALUES  ('A0009', 'V0005', 'S0006', 'T0009', DATE'2021-1-18','NEW', NULL);  INSERT INTO Application VALUES  ('A0010', 'V0007', 'S0005', 'T0002', DATE'2021-1-7','NEW', NULL); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

|  |  |
| --- | --- |
| Name | KYAW THIHA NAING |
| List of tables | Submission\_Record |
| SQL Statements  (All queries to create tables and insert records | CREATE TABLE Submission\_Record (  docID VARCHAR (20) NOT NULL,  applicationID VARCHAR (20) NOT NULL,  CONSTRAINT sub\_rec\_PK PRIMARY KEY (docID, applicationID),  CONSTRAINT sub\_doc\_FK FOREIGN KEY (docID) REFERENCES DOCUMENT (docID) ON DELETE CASCADE,  CONSTRAINT sub\_app\_FK FOREIGN KEY (applicationID) REFERENCES APPLICATION (applicationID) ON DELETE CASCADE  );  INSERT INTO Submission\_Record VALUES  ('D0001','A0010');  INSERT INTO Submission\_Record VALUES  ('D0002','A0009');  INSERT INTO Submission\_Record VALUES  ('D0003','A0008');  INSERT INTO Submission\_Record VALUES  ('D0004','A0007');  INSERT INTO Submission\_Record VALUES  ('D0005','A0006');  INSERT INTO Submission\_Record VALUES  ('D0006','A0005');  INSERT INTO Submission\_Record VALUES  ('D0007','A0004');  INSERT INTO Submission\_Record VALUES  ('D0008','A0003');  INSERT INTO Submission\_Record VALUES  ('D0009','A0002');  INSERT INTO Submission\_Record VALUES  ('D0010','A0001'); |
| Results  (Provide snapshot for each table showing the inserted records) |  |

**Data Manipulation**

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of Query** | This query is useful when the CRS users want to see whether each individual staff is handling enough trips. If the labor is not divided equally, some of the staff will have more workload than the others. So, by doing this, new staff can be assigned to trips that are only handled by one staff in order to prevent over workload. |
| **SQL Statement** | SELECT staffID, COUNT(tripID) AS "Number of trips"  FROM crs\_staff JOIN trip USING (staffID)  GROUP BY staffID  HAVING COUNT(tripID) >= 1  ORDER BY staffID; |
| **Output**  **(Provide snapshot)** |  |

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of Query** | This query is useful when the CRS users want to filter out the trips that most of the volunteers applied for. If there are other trips that are not getting enough attention due to volunteers not making an application, the CRS organization can reanalyze their marking strategy and change it if necessary, so that the other trips get more exposure and applications |
| **SQL Statement** | SELECT trip.tripID,destination,crisis\_type, COUNT(applicationID) AS "Number of Applications"  FROM trip,application  WHERE trip.tripID = application.tripID  GROUP BY trip.tripID,destination,crisis\_type  HAVING COUNT(applicationID) > 1  ORDER BY trip.tripID; |
| **Output**  **(Provide snapshot)** |  |

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of Query** | This query checks for volunteers who applied for trips during holidays. Their contact info is also checked to notify that the staff will not be able to handle and approve their applications anytime soon and will have to wait till the holidays are over. The number of documents will give the staff an idea on how many documents to process in extra when going back to work once the holidays are over. |
| **SQL Statement** | SELECT volunteer.volunteerID, COUNT(docID) AS "Number of documents", first\_name, last\_name, phone, application\_date  FROM volunteer, document, application  WHERE application\_date >= TO\_DATE('2021-01-01','YYYY-MM-DD')  AND application\_date <= TO\_DATE('2021-01-14','YYYY-MM-DD')  AND volunteer.volunteerID = document.volunteerID  AND document.volunteerID = application.volunteerID  GROUP BY first\_name, last\_name, phone, application\_date, volunteer.volunteerID  HAVING COUNT(docID) > 1  ORDER BY volunteerID; |
| **Output**  **(Provide snapshot)** |  |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of Query** | To check the number of current registered volunteers for each trip in order to meet the no of required volunteers |
| **SQL Statement** | SELECT tripID, trip\_date, req\_vol, COUNT(volunteerID) as "Current Noof Volunteers"  FROM trip join application using(tripID)  GROUP BY tripID, trip\_date, req\_vol; |
| **Output**  **(Provide snapshot)** |  |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of Query** | To check and notify rejected volunteers via phone call/ SMS in order to ask them to re submit for urgent crisis trip |
| **SQL Statement** | SELECT ApplicationID, volunteerID, phone, first\_name||' '||last\_name AS "VolunteerName"  From Application JOIN volunteer using (volunteerID)  WHERE status='REJECTED'  GROUP BY ApplicationID, volunteerID, first\_name,last\_name, phone; |
| **Output**  **(Provide snapshot)** |  |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of Query** | To check any volunteers have life skills certification related to certain crisis type of trip |
| **SQL Statement** | SELECT tripID, crisis\_type,volunteerID, docID, doc\_type, image  FROM trip join application using (tripID) join document using (volunteerID)  WHERE crisis\_type = 'FLOOD' AND image LIKE 'SWIM%'  GROUP BY tripID, crisis\_type, volunteerID, docID, doc\_type, image; |
| **Output**  **(Provide snapshot)** |  |

**Procedural SQL**

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of PL/SQL** | This trigger is to keep an update when new staff is hired in the CRS to keep an audit trail |
| **PL/SQL Statement** | CREATE TRIGGER CRSstaffUpdate  AFTER INSERT ON CRS\_staff  REFERENCING NEW AS new  FOR EACH ROW  BEGIN  INSERT INTO CRSStaffAudit VALUES(:NEW.staffID, :NEW.crsID, :NEW.password, :NEW.first\_name, :NEW.last\_name, :NEW.phone);  END |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | The audit table needs to be created first  CREATE TABLE CRSStaffAudit (  staffID VARCHAR (10) NOT NULL,  crsID VARCHAR (10) NOT NULL,  password VARCHAR (20) NOT NULL,  first\_name VARCHAR (10) NOT NULL,  last\_name VARCHAR (10) NOT NULL,  phone VARCHAR (30) NOT NULL,  position VARCHAR (20) NOT NULL,  date\_joined DATE NOT NULL,  CONSTRAINT crs\_staff\_PK PRIMARY KEY (staffID),  CONSTRAINT staff\_crs\_FK FOREIGN KEY (crsID) REFERENCES CRS(crsID) ON DELETE CASCADE  );  The new CRS staff is to be added to the staff record which will activate the trigger and update the CRSStaffAudit table  INSERT INTO CRS\_staff  VALUES('S0013','C0010','fortriggerssake','Mathilda','Hargreeves','0168976546','Consultant',DATE'2015-07-26');  The following code is to check the CRSStaffAudit table after the activation of the trigger  SELECT \* FROM CRSStaffAudit; |

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of PL/SQL** | This function will display the total number of staff currently working for the CRS |
| **PL/SQL Statement** | CREATE OR REPLACE FUNCTION fngetTotalStaff RETURN NUMBER IS totalStaffnum NUMBER;  BEGIN    SELECT COUNT(staffID) INTO totalStaffnum FROM CRS\_staff;  RETURN totalStaffnum;  END fngetTotalStaff; |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | Due to the function not directly being executable, an anonymous SQL block is created to call the created function. Executing this anonymous SQL block will display the total staff members currently working for the CRS  DECLARE  vCRSStaffAllColumns CRS\_staff%ROWTYPE;  vtotalStaffnum NUMBER;  CURSOR CRSStaffCursor IS SELECT \* FROM CRS\_staff;  BEGIN  vtotalStaffnum:=fngetTotalStaff;  DBMS\_OUTPUT.PUT\_LINE('The total number of staff currently working is ' || vtotalStaffnum);  END |

|  |  |
| --- | --- |
| **Name** | KYAW THIHA NAING |
| **Purpose of PL/SQL** | This procedure will display the number of trips that a specific staff is responsible for |
| **PL/SQL Statement** | CREATE OR REPLACE PROCEDURE getTotalTrip (vstaffID IN Trip.staffID%TYPE) IS  vtripTotal INTEGER;  BEGIN  SELECT COUNT(staffID) INTO vtripTotal FROM Trip WHERE staffID = vstaffID;  DBMS\_OUTPUT.PUT\_LINE('The number of trip that ' || vstaffID || ' is responsible for is ' || TO\_CHAR(vtripTotal));  END getTotalTrip; |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | The query below can be executed for a specific staff ID of interest  BEGIN  getTotalTrip('S0002');  END |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of Query** | To check newly joined volunteer |
| **SQL Statement** | CREATE **TRIGGER** VolAfterInsert  AFTER INSERT ON VOLUNTEER  REFERENCING NEW as new  FOR EACH ROW  BEGIN  INSERT INTO VolJoin  VALUES(:NEW.volunteerID, :NEW.crsID, :NEW.password, :NEW.first\_name, :NEW.last\_name, :NEW.phone);  END |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | We need to create a table and insert new values first then only can see the result  CREATE TABLE VolJoin (  volunteerID VARCHAR (10) NOT NULL,  crsID VARCHAR (10) NOT NULL,  password VARCHAR (20) NOT NULL,  first\_name VARCHAR (10) NOT NULL,  last\_name VARCHAR (10) NOT NULL,  phone VARCHAR (30) NOT NULL,  CONSTRAINT voljoin\_PK PRIMARY KEY (volunteerID),  CONSTRAINT voljoin\_crs\_FK FOREIGN KEY (crsID) REFERENCES CRS(crsID) ON DELETE CASCADE  );  INSERT INTO VOLUNTEER VALUES  ('V0012','C0010','triggertesting','Jack','Reacher','0665423201');  SELECT\* FROM VolJoin |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of Query** | To track current amount of applications in specific trip |
| **SQL Statement** | CREATE **PROCEDURE** getVolTotal (vtripID IN Application.tripID%TYPE) IS  vTotal INTEGER;  BEGIN  SELECT COUNT(tripID) INTO vTotal FROM Application WHERE tripID= vtripID;  DBMS\_OUTPUT.PUT\_LINE('Current amount of applications in this trip is ' || TO\_CHAR(vTotal));  END getVolTotal ; |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | We need to check this with following query by input tripID  BEGIN  getVolTotal ('T0002');  END); |

|  |  |
| --- | --- |
| **Name** | KIM SANGA |
| **Purpose of PL/SQL** | To determine how many volunteers are actually participate in the crisis trip |
| **PL/SQL Statement** | CREATE **FUNCTION** getActiveVol RETURN INTEGER IS  actVolunteer INTEGER;  BEGIN  SELECT COUNT(DISTINCT volunteerID) INTO actVolunteer FROM Application;  RETURN actVolunteer;  END getActiveVol; |
| **Output**  **(Provide snapshot)** |  |
| **Remark** | As Function cannot call directly, we need anonymous SQL blocks to call/utilize.  DECLARE  vAppAllColumns APPLICATION%ROWTYPE;  vActVolunteer INTEGER;  BEGIN  vActVolunteer:=getActiveVol;  DBMS\_OUTPUT.PUT\_LINE('Num of current active participants: ' || vActVolunteer);  END |

**REFERENCES**

Gröger, C. (2018). Building an Industry 4.0 Analytics Platform. *Datenbank-Spektrum*,

*18*(1). doi: 10.1007/s13222-018-0273-1. Retrieved from link.springer.com/article/10.1007/s13222-018-0273-1

Park, Y. R., Yoon, Y. J., Koo, H. Y., Yoo, S., Choi, C.-M., Beck, S.-H., & Kim, T. W.

(2018). Utilization of a Clinical Trial Management System for the Whole Clinical Trial Process as an Integrated Database: System Development. *Journal of Medical Internet Research*, *20*(4), e103. Retrieved from https://doi.org/10.2196/jmir.9312

Suhr, A., Chang, M.-W., Shaw, P., & Lee, K. (2020). Exploring Unexplored

Generalization  Challenges for Cross-Database Semantic Parsing. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 8372–8388. Retrieved from https://doi.org/10.18653/v1/2020.acl-main.742

Toapanta, S., Escalante Quimis, O., Gallegos, L., & Maciel Arellano, M. (2020). Analysis

for the Evaluation and Security Management of a Database in a Public Organization to Mitigate Cyber Attacks. *IEEE Access*, *8*. doi: 10.1109/access.2020.3022746. Retrieved from https://ieeexplore.ieee.org/abstract/document/9193956