CS360 Term Project 2 Report

<HOEP: Helper for Outbound Exchange Program>

# Team 11, 20160171 김진우 / 20150274 류자영 / 20160756 박현철

1. **Introduction**

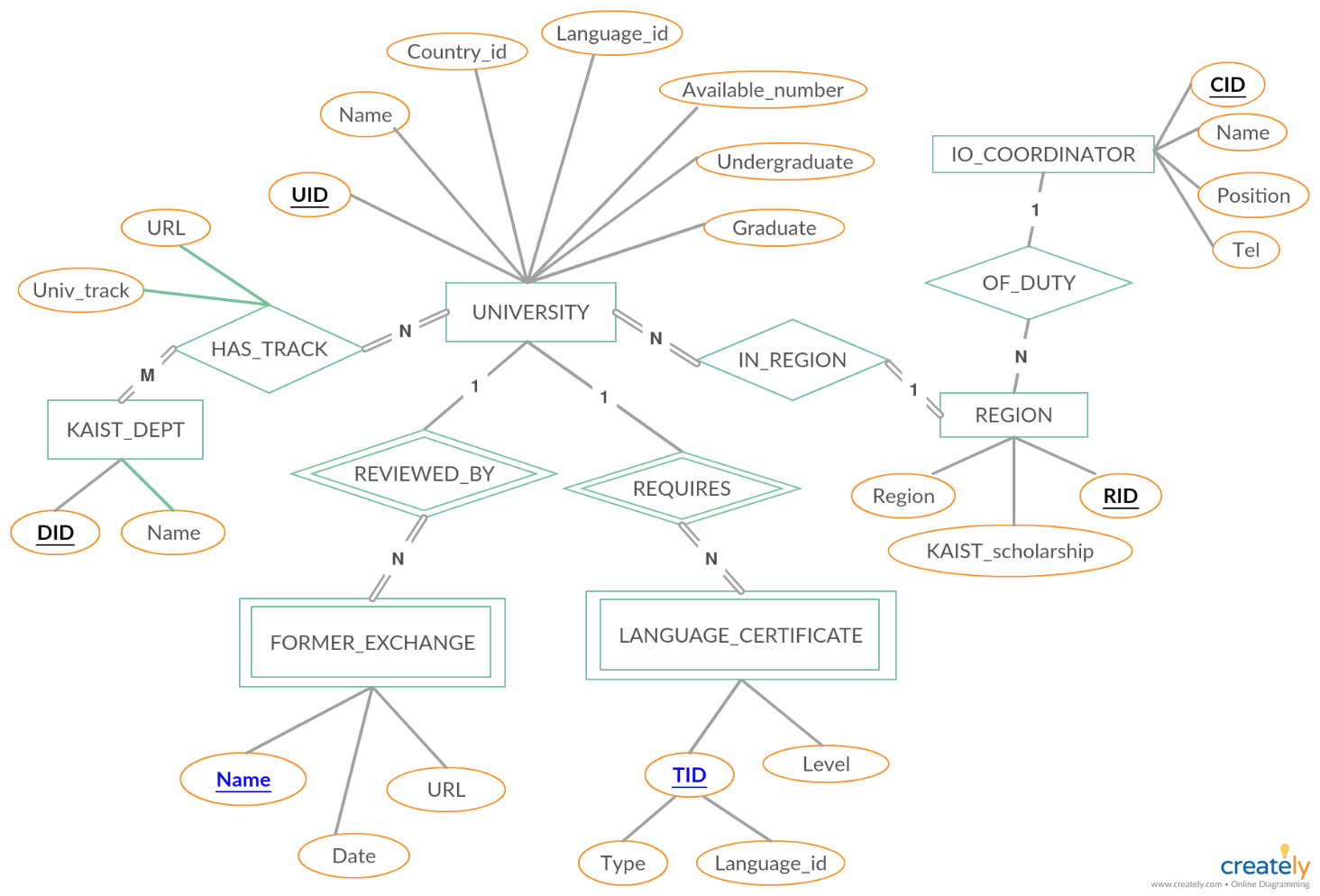
Despite **KAIST students’ recent high interest in outbound exchange programs**, data relevant to partner universities are currently fragmented and time consuming to retrieve.

Therefore, we developed **a web service allowing easy retrieval of well-organized information on outbound exchange program partner universities**. Out service provides **conditional search given user input** (geographical location and KAIST department), therefore meeting each user’s specific interest.

Moreover, the provided information includes **not only the university’s basic features, but also specific ones** like financial support, former exchange students’ reviews, IO coordinator of duty, and language test certificate.

Taken together, our service aims to **provide efficient initial assistance for choosing the best university** to apply as an exchange student, even for those with near-zero background information.

1. **Conceptual modeling**



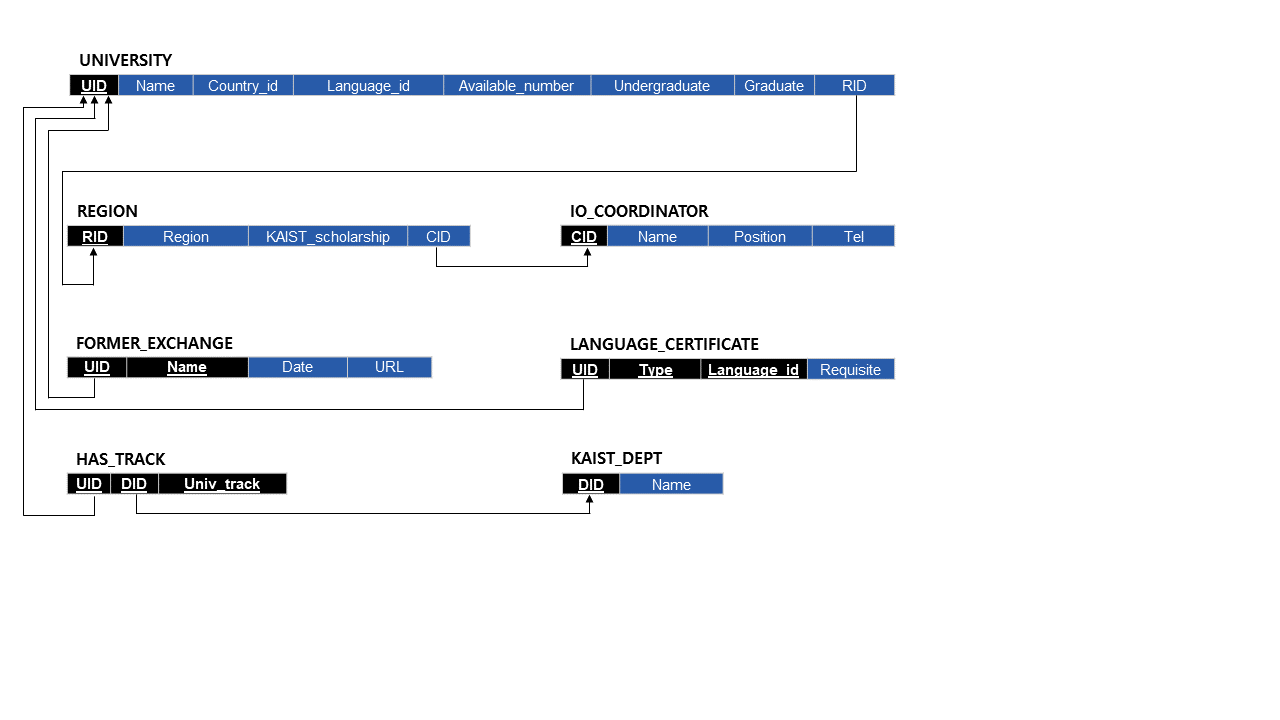
**<Figure 1. ER Diagram>**

(**Black + underlined attribute: primary key** / **Blue + underlined attribute: partial key**)

1. **Relational Model**
2. **Table schema**

As some candidate key attributes like UNIVERSITY.Name were inadequate to be used as tuple identifier (too long, may cause time lag and error during computation), we introduced simple integer ID attributes (UNIVERSITY.UID, REGION.RID, IO\_COORDINATOR.CID and KAIST\_DEPT.DID) as primary key for many tables.

With this adjustment, we were able to implement our conceptual model in a clear and simple way.



**<Figure 2. Table schema with referential integrity constraint>**

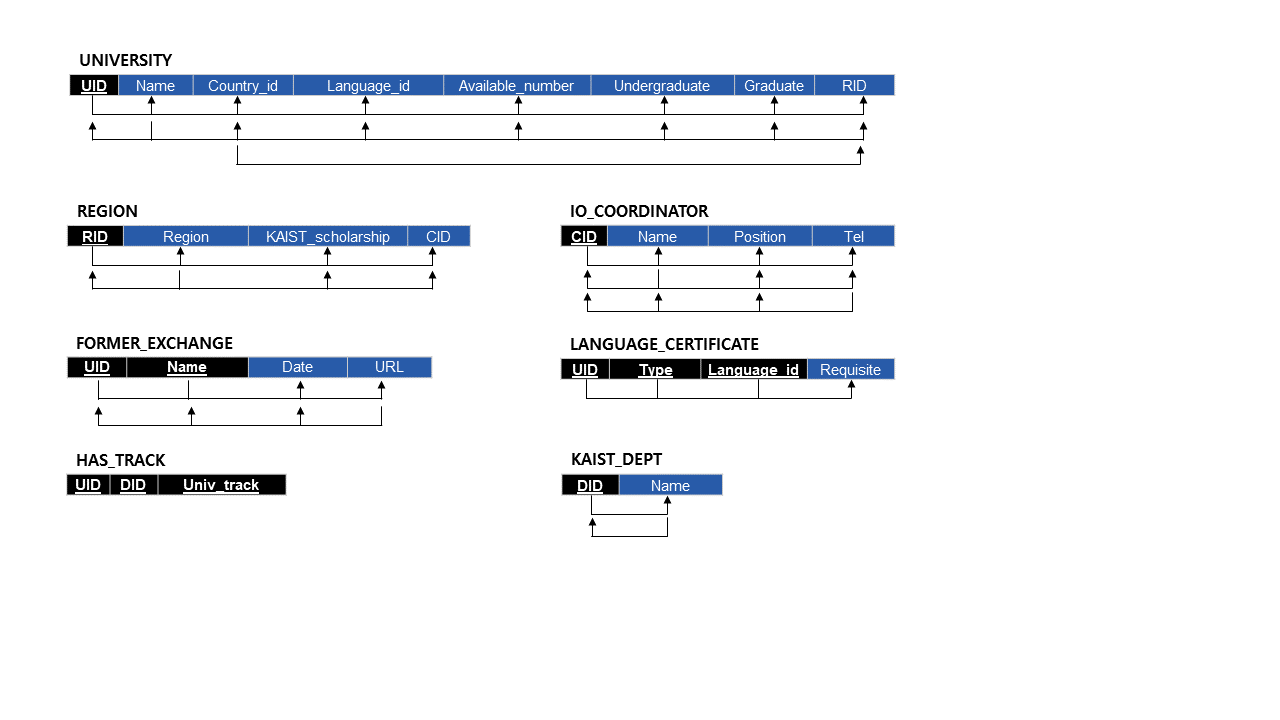
1. **Functional dependencies**

In this project, we used 2nd normal form. Following constraints were enforced for every tables.

* **1st normal form achieved** (only atomic values allowed for each attribute).
* **Every** **nonprime attribute is fully functionally dependent on the primary key.**

\* If we were to use 3rd normal form, many semantically important candidate keys like UNIVERSITY.Name, REGION.Region and IO\_COORDINATOR.Name would be paired into separate table with UNIVERSITY.UID, REGION.RID and IO\_COORDINATOR.CID.

As we want those representative attributes to remain in the same table which have specific information, we did not extend our schema to 3rd normal form.



**<Figure 3. 2nd normal form>**

1. **Web implementation (summarized)**

We used JavaScript code – connected to MySQL DBMS – to implement core functions, based on the provided tutorial code. Specifically,

* Initial base table construction from .csv files and some view table definitions were done in SQL code directly (“initial\_db\_construction.sql”).
* Major functions were implemented in Node.js code (“app\_univ.js”), sending queries to MySQL DBMS and receiving the result.

We implemented web UI in html and ran the service in the local server for demonstration.

1. **SQL Code (DML)**

**Selection and projection (line 124, app\_univ.js)**

* Given a keyword for the university name, select FORMER\_EXCHANGE tuples of which Univ\_name attribute contains the keyword.
* Project some of the attributes (full university name, former exchange student name, review URL) to be displayed.

**Code**

"**SELECT** Univ\_name, Name, URL

**FROM** FORMER\_EXCHANGE

**WHERE** Univ\_name **LIKE** '%" + req.body.univ+ "%';"

**Insert (line 171, app\_univ.js)**

* Given a keyword for the university name, check if a single matching UNIVERSITY tuple exists.
* With the found UNIVERSITY tuple’s UID and additionally given student name, URL and current date, make a new tuple for FORMER\_EXCHANGE table.
* Perform insertion, if it does not violate the entity integrity constraint (overlapping UID and student name).

**Code**

"**INSERT INTO** FORMER\_EXCHANGE

**VALUES** ("+uid +", '"+univName+"', '"+req.body.author+"', '"+today+"', '"+req.body.URL+"')"

**Update (line 246, app\_univ.js)**

* Given a region key, find the matching REGION tuple.
* With newly given IO coordinator ID, update the REGION tuple’s CID attribute.

**Code**

"**UPDATE** Region **SET** CID = "+req.body.coord\_CID+"

**WHERE** RID = "+req.body.region\_CID+";";

**Delete (line 208, app\_univ.js)**

* Given a keyword for the university name, check if a single matching UNIVERSITY tuple exists.
* With found UNIVERSITY tuple’s UID and additionally given student name, check if a single matching FORMER\_EXCHANGE tuple exists.
* Delete the tuple from the FORMER\_EXCHANGE base table.

**Code**

"**DELETE FROM** FORMER\_EXCHANGE

**WHERE** UID = '"+uid+"' and Name = '"+req.body.author+"';"

**Join (line 105, initial\_db\_construction.sql)**

* Show all IO coordinators and their corresponding regions of duty**.**
* To achieve this, join REGION table and IO\_COORDINATOR table by CID attribute and create a view table.

**Code**

**CREATE VIEW** REGION\_COORD AS

**SELECT** REGION.RID, REGION.Region,

IO\_COORDINATOR.Name, IO\_COORDINATOR.Position, IO\_COORDINATOR.Tel

**FROM** REGION **JOIN** IO\_COORDINATOR **USING**(CID);

**Subquery (line 59, app\_univ.js)**

\* Perform all the below commands using a single SQL statement containing encapsulated subqueries (two inline views).

* First, inner-join TRACK\_N\_UNIV table to REGION by RID to assign KAIST scholarship values to each partner university’s tracks according to regions.
* Given a KAIST department id and a list of region ids, select corresponding TRACK\_N\_UNIV tuples (joined with REGION) and project UID, Univ\_track and KAIST\_scholarship attributes (subquery).
* Inner-join the table of projected TRACK\_N\_UNIV tuples with the UNIVERSITY table by UID attribute.
* Among the resulting tuples, select ones that accept undergraduate students (main query).
* Project some of the attributes to be displayed.

**Code**

'**SELECT** A.Name as Univ\_name, B.Univ\_track as Dept,

A.Country\_id, A.Language\_id, A.Available\_number, B.Scholarship

**FROM** UNIVERSITY AS A

**INNER JOIN**

(**SELECT** TRACK\_N\_UNIV.UID as UID, TRACK\_N\_UNIV.Univ\_track as Univ\_track,

REGION.KAIST\_scholarship as Scholarship

**FROM** (TRACK\_N\_UNIV **INNER JOIN** REGION **USING** (RID))

**WHERE** DID = ‘ + req.body.user\_dept + ‘ AND RID IN ‘ + region\_str\_set + ’) AS B

**USING** (UID)

**WHERE** Undergraduate = 1;’

**Aggregation (line 100, app\_univ.js)**

* Given theobtained table of TRACK\_N\_UNIV tuples in **Subquery**, perform aggregation (COUNT) to acquire the total number of retrieved tuples.

**Code**

"**SELECT** **COUNT(\*)** AS **Count**

**FROM** TRACK\_N\_UNIV

**WHERE** DID = “ + req.body.user\_dept + “ AND RID IN “ + region\_str\_set

**Additional: View (line 100, initial\_db\_construction.sql)**

* Create a view table containing partner universities’ name, region id, UID, each track’s name and corresponding KAIST department ids for later use (in **Subquery** code).
* For this, join UNIVERSITY table and HAS\_TRACK table by UID attribute and project the needed attributes.

**Code**

**CREATE VIEW** TRACK\_N\_UNIV AS

**SELECT** A.UID, A.DID, A.Univ\_track, B.RID, B.Name

**FROM** HAS\_TRACK AS A

**JOIN** UNIVERSITY AS B **USING**(UID);

1. **Future Work**

In here we list some further improvements or expansions that can be later made to our service.

* Recording users’ access frequency for each university data
* As KAIST IO does not allow access to past students’ preference on region / universities, user access data can be analyzed to provide an alternative clue about such tendency.
* Integrating university search function and language requirement check function
* The two functions are currently separated.
* The functions can be integrated so that language prerequisite can be accessed directly from the university search result page.
* Capability to update UNIVERSITY base table
* Required for service consistency over years (keeping miniworld information up-to-date)
* May call for many triggered SQL operations, as UNIVERSITY table references / is referenced by many other tables via foreign key.

1. **Program Code**

* **Added as supplementary**