Hyper T-Rex

# Group Members:

Twila Habab C20361521  
Jaycel Estrellado C20372876

# Domain Types

|  |  |  |  |
| --- | --- | --- | --- |
| Domain Type Name | Type Definition | Reason Introduced | Example |
| Description | VARCHAR(50) | For attributes which are character-based descriptions. It is a character string defined by user, specifying the maximum length. | Cust\_name in customer\_details\_tbl,  staff\_address in staff\_tbl |
| Amount | INT(10,20) | For attributes which represent monetary amounts. It will be used for ids, age and any other number values used throughout the database. | Manager\_id in manager\_tbl,  Cust\_age in customer\_details\_tbl |
| Date | TO\_DATE WITH LOCAL ZONE | Date and time attributes which require a very clearly defined time element, Date will be displayed as ‘YYYY-MM-DD HH24:MI:SS' in the database. This will be used to store the booked and check out time in the database. | Stat\_booked in table\_status\_tbl,  Stat\_checkout in table\_status\_tbl |

# Major Decisions

## General Idea

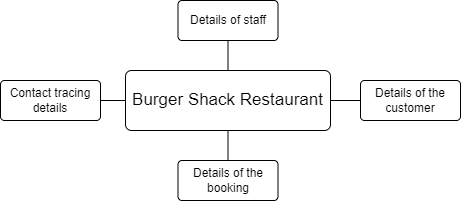
It was understood that the purpose of this project is to make a database that can store relevant details on customers, waiting staff and managers at every restaurant branch around the country. The goal is to make it so that the tables would be related to each other, hence a relational database.

In this way, relevant data about the restaurant, people, etc., would be easily tracked and be kept consistent. Furthermore, the design and purpose of this database would be relevant to the current pandemic by keeping track of any contacted personnel.

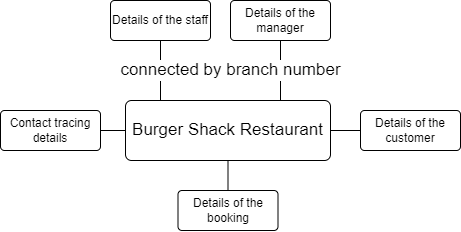
## Design / Creation of entities

Conceptual level

Information from the database must be relevant to the case study in place. The main details to be stored into the database were as follows:

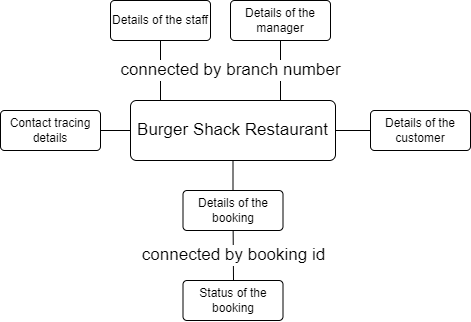


There will be a dedicated table for the staff, customers, booking details, and contact tracing details. However, if the entities were done this way, then there will be potential data redundancy (repeat of fields and records) such as the branch number for a staff and manager. As a solution, the staff entity was split into two and would be the staff entity and the manager entity as they are two distinct types of staff personnel, which can be both linked by the branch number of the restaurant.

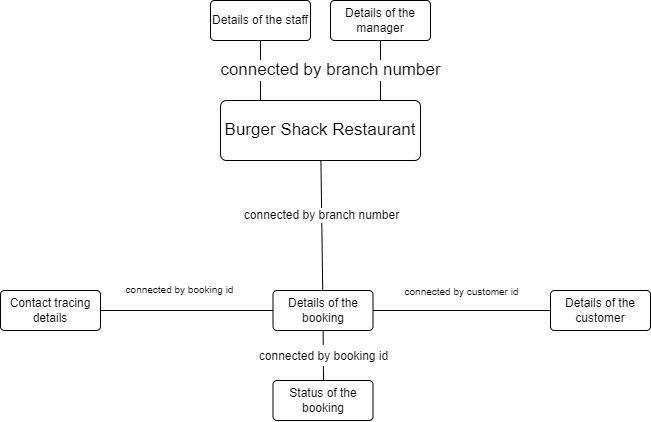


This also helps solve the problem of “what if the customer books onsite, a manager approval is needed” and therefore would be easier to navigate through the database by looking into the manager entity. This also results in giving more identity to managers, separated from the other staff.

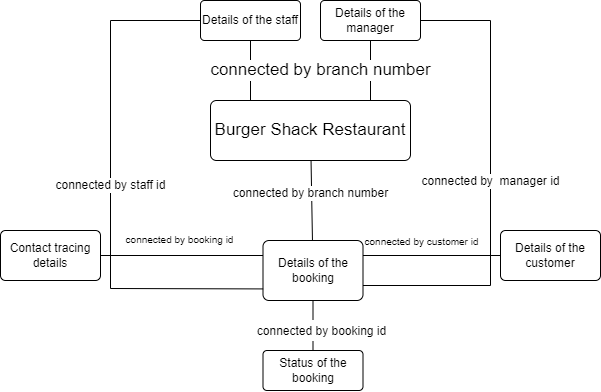
With the current concept, it was clear that the entity for booking details would be large and would contain too many attributes. The booking details would not only contain the booking id, time of booking, etc., but also the status of the booking. Due to the government regulations, customers will be fined if they stay longer than the 2-hour maximum time limit. It would be more logical to have a separate entity for statuses for each booking detail linked by a booking id. This in return would simplify the database and still would be able to retrieve relevant information whilst keeping the data integrity and relation in between tables.



At this point of the design, it was decided that the details of the customer should be linked to the details of the booking through customer id, and the details of the booking should be connected to contact tracing details through booking id. The booking details should also contain the branch number to avoid potential data redundancy (repeating booking id) when there are multiple bookings (especially under the same customer).



Details of the waiter and manager (provided they have booked onsite) who served a certain booking must also be recorded.

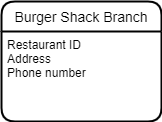


As a result, the basic entities are now all connected and now ready for more detailed concepts.

Basic models of the entities/tables

* Burger Shack Branch Entity

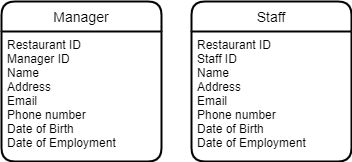
Each branch must have a unique numeric restaurant identifier, address, and a phone number.



- Manager and Staff Entities

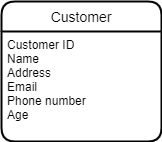
Each manager and staff must have their own unique ID, assigned to a particular branch, and has their name, address, phone number, email address, date of birth, and start date of employment stored in the database.

(Note in the assignment that staff\_table is meant for servers only, the structure of this database is simplified for the purpose of having a functional database that can model real life problems such as the pandemic where we can keep track of close contacts.)



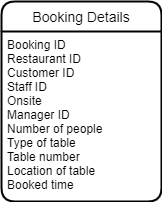
- Customer Entity

Each customer must have their own unique ID, and their name, address, email, phone number, and age.



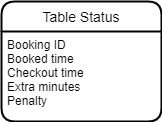
- Booking Details Entity

Each booking must have a unique booking ID and the ID of the Restaurant as there could be multiple bookings with the same booking ID (but in a different restaurant. It must contain the ID of the customer who booked and the ID of the staff who served the booking. The booking details must also indicate whether it was booked onsite or not, and if so, manager ID must be included for proof of approval. The number of people will also be listed, and so is the type of table (whether if it is a 2, 4, 6 or 8 seater), the table number, location of the table (window or interior), and the booking time.



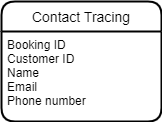
- Table Status Entity

Each booking must have their own table status entity. This would contain details such as the booking ID, the booked time, checkout time, extra minutes (if applicable) and the penalty (if applicable).



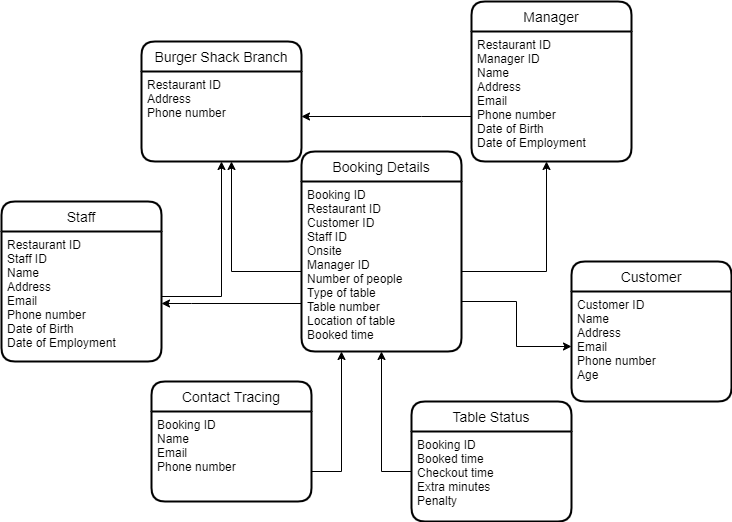
- Contact Tracing Entity

An entity dedicated for people under a certain booking. Must include the booking ID, customer ID, their name, email, and phone number.



Conceptual model

Below would be the conceptual model of the database.



Attributes

All ID’s are given VARCHAR instead of INT because there are a couple of IDs used in the database. For each ID to have its own uniqueness, an extra 2 chars will be used at the beginning of the ID and will end with an INT (e.g., Burger Shack Branch ID will be BS01 and Customer ID will be CT01 etc).   
All phone numbers are given VARCHAR instead of INT because there are numbers that start with 0.

Burger Shack Branch Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Restaurant ID | VARCHAR2(10) NOT NULL | To identify the unique restaurant branch address by their id. It is varchar so that it is easier to identify the restaurant id over the other existing id in the other tables. |
| Address | VARCHAR2(100) |  |
| Phone number | VARCHAR2(15) |  |

Manager Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Restaurant ID | VARCHAR2(10) NOT NULL | Required |
| Manager ID | VARCHAR2(10) NOT NULL | Required |
| Manager name | VARCHAR2(50) NOT NULL | Required |
| Manager address | VARCHAR2(100) |  |
| Manager email | VARCHAR2(50) |  |
| Manager phone number | VARCHAR2(15) |  |
| Date of birth | DATE |  |
| Date of employment | DATE |  |

Staff Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Restaurant ID | VARCHAR2(10) NOT NULL | Required |
| Staff ID | VARCHAR2(10) NOT NULL | Required |
| Staff name | VARCHAR2(50) NOT NULL | Required |
| Staff address | VARCHAR2(100) |  |
| Staff email | VARCHAR2(50) |  |
| Staff phone number | VARCHAR2(15) |  |
| Date of birth | DATE |  |
| Date of employment | DATE |  |

Customer Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Customer ID | VARCHAR2(20) NOT NULL | Required |
| Customer name | VARCHAR2(50) NOT NULL | Required |
| Customer address | VARCHAR2(100) |  |
| Customer email | VARCHAR2(50) |  |
| Customer age | NUMBER(\*,0) | To be able to check if the booker is over 18 or under 18. They must be over the age of 18 in order to book successfully. |

Booking Details Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Booking ID | VARCHAR2(10) NOT NULL |  |
| Restaurant ID | VARCHAR2(10) NOT NULL |  |
| Customer ID | VARCHAR2(20) NOT NULL |  |
| Staff ID | VARCHAR2(10) NOT NULL |  |
| Onsite | VARCHAR2(1) | Oracle does not support Boolean datatype, therefore if customer books Onsite, it will be 1, otherwise 0 |
| Manager ID | VARCHAR2(10) | This will optional and will depend if **onsite** is True or False (can be empty if booked online or on call) |
| Number of people | NUMBER(\*,0) |  |
| Type of table | NUMBER(\*,0) | 2, 4, 6 or 8 seater |
| Table number | NUMBER(\*,0) |  |
| Location of table | VARCHAR2(10) | ‘Window’ or ‘Interior’ |
| Booked time | DATE |  |

Table Status Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Booking ID | VARCHAR2(10) NOT NULL | Required |
| Booked time | DATE | Regardless of what time the customer comes in to check in, the calculation will depend on the time it was booked at |
| Checkout time | DATE |  |
| Extra | NUMBER(\*,0) |  |
| Penalty | NUMBER(\*,0) |  |

Contact Tracing Table

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Datatype** | **Reasoning** |
| Booking ID | VARCHAR2(10) NOT NULL | Required |
| Customer ID | VARCHAR2(10) NOT NULL | Required |
| Email | VARCHAR2(50) NOT NULL | Required. Default way of communicating to close contact personnel |
| Name | VARCHAR2(20) |  |
| Phone Number | VARCHAR2(15) |  |

To avoid ambiguity in terms of column names, each column name would start with a part of the name of their respective entity (e.g., there are distinct types of emails throughout the database, therefore it would be more specific if each email would be named to their corresponding entity, such as customer email would be cust\_email and the person to be contacted for contact tracing would be trace\_email).

Normalization check

1NF

* Each column has a unique name.
* Each record would be unique because of the primary keys included in each entity.
* Each value is atomic.

2NF

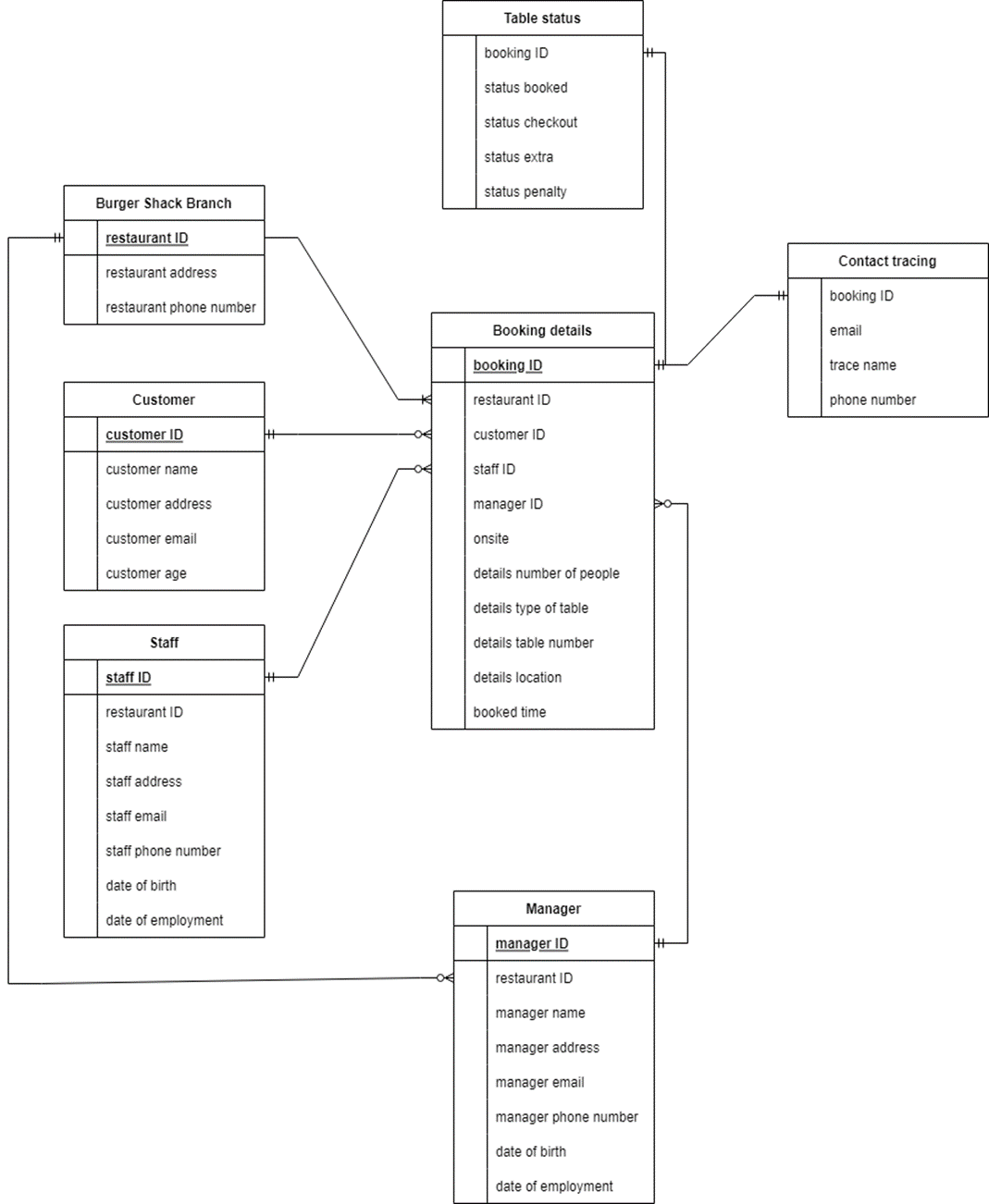
* 1NF has been applied.
* Each non-key attribute is functionally dependent on the primary key.
* There are no compound-primary keys.

3NF

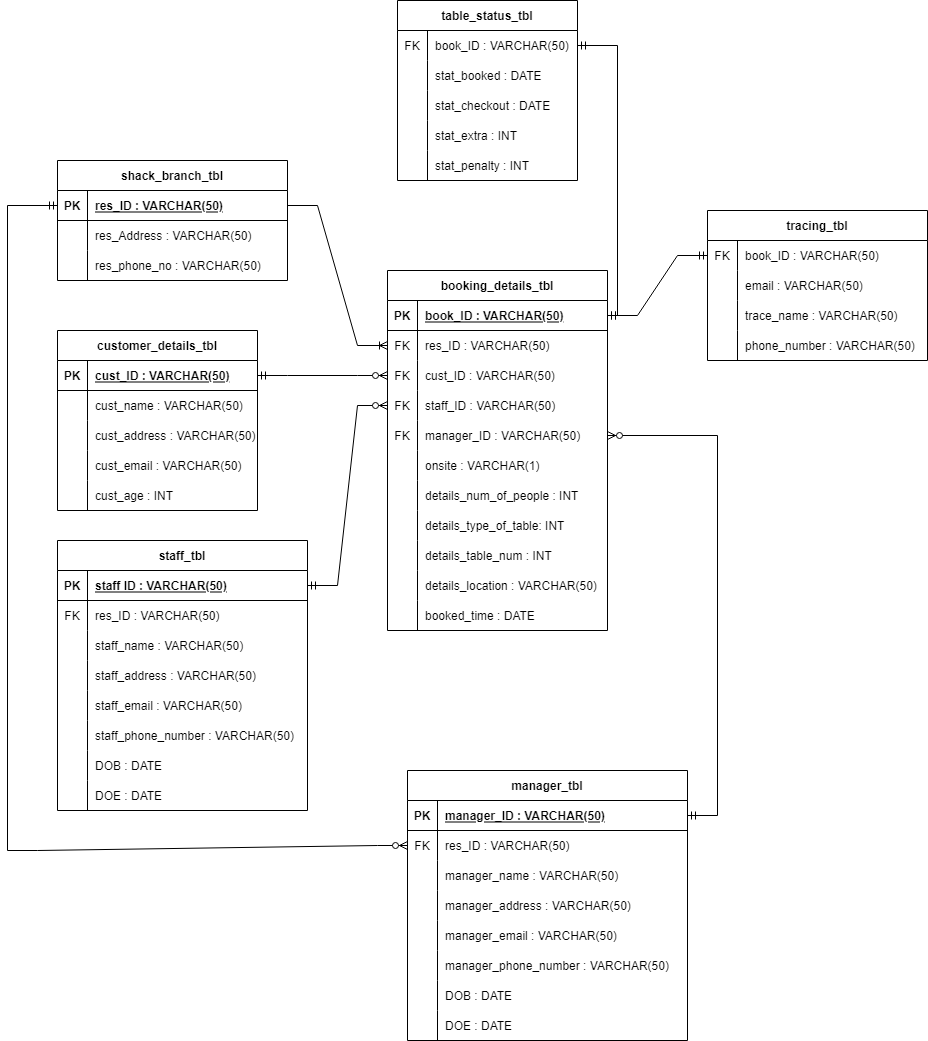
* 2NF has been applied.
* No attribute is transitively dependent on the primary key.
* Every attribute that is not the primary key must depend on the primary key and the primary key only.

# Present ERD

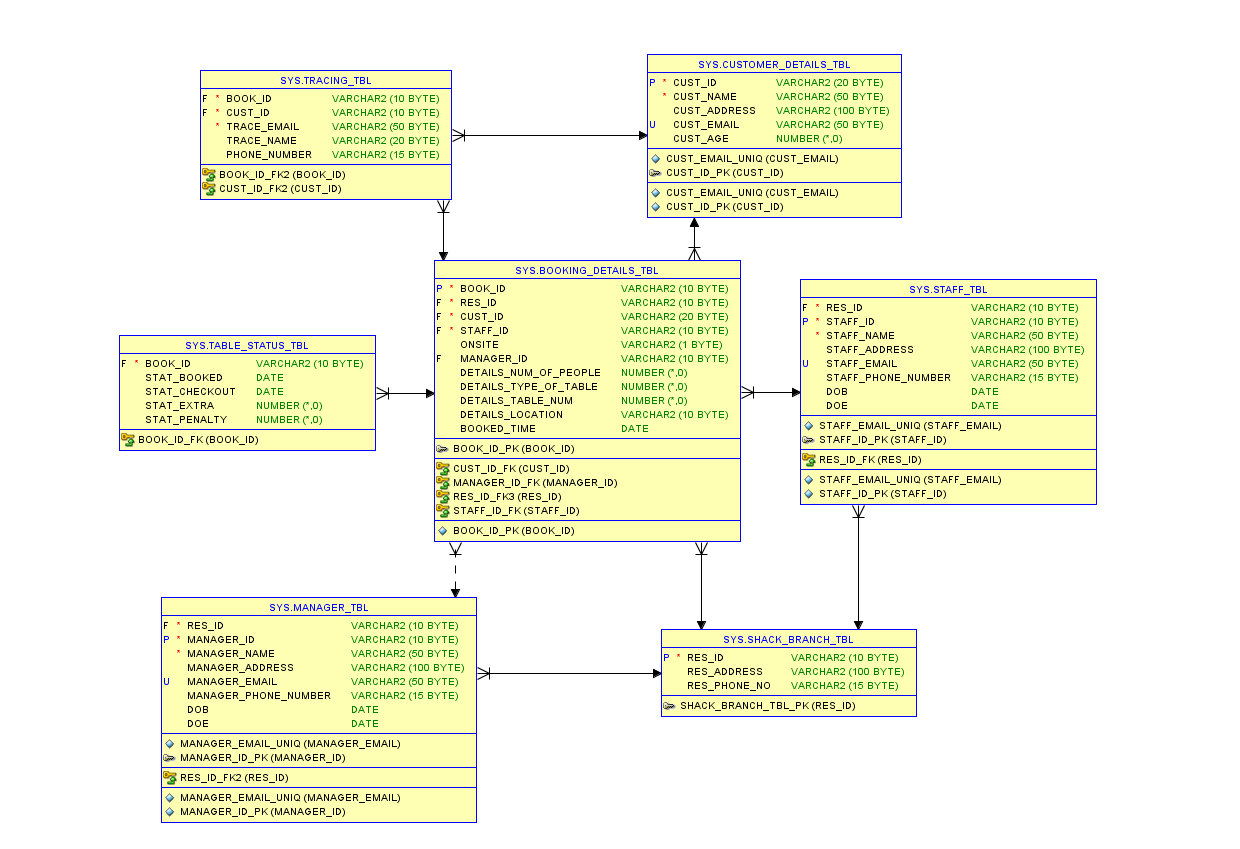
Logical Model:



Physical Model:

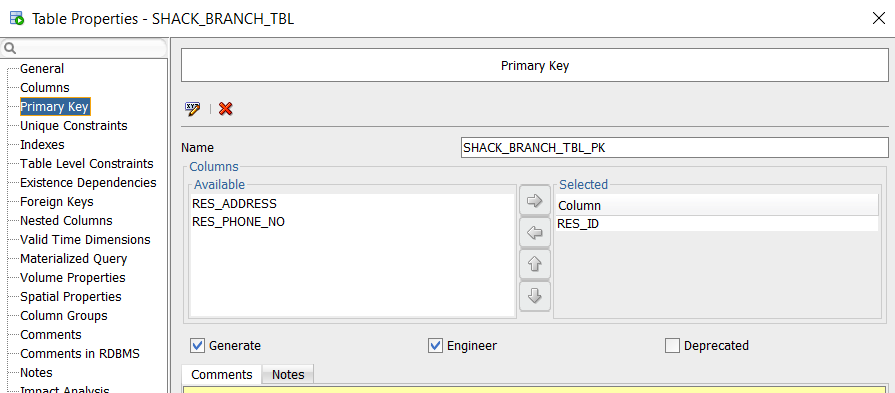


**Oracle ERD:**

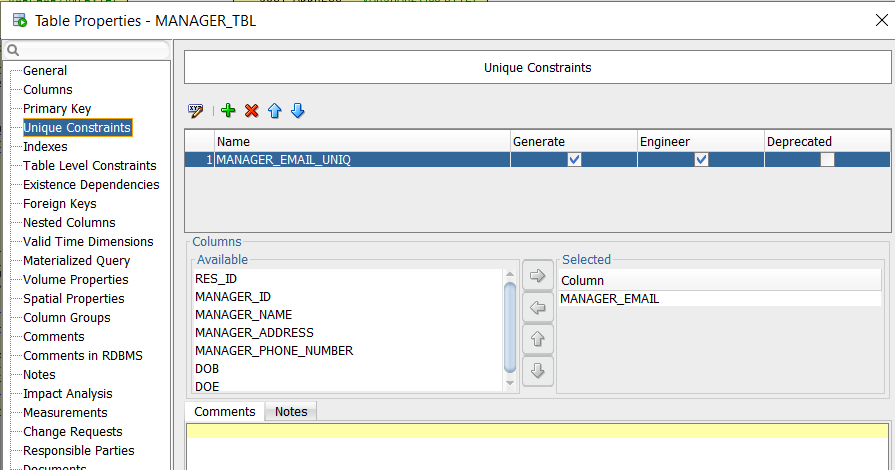
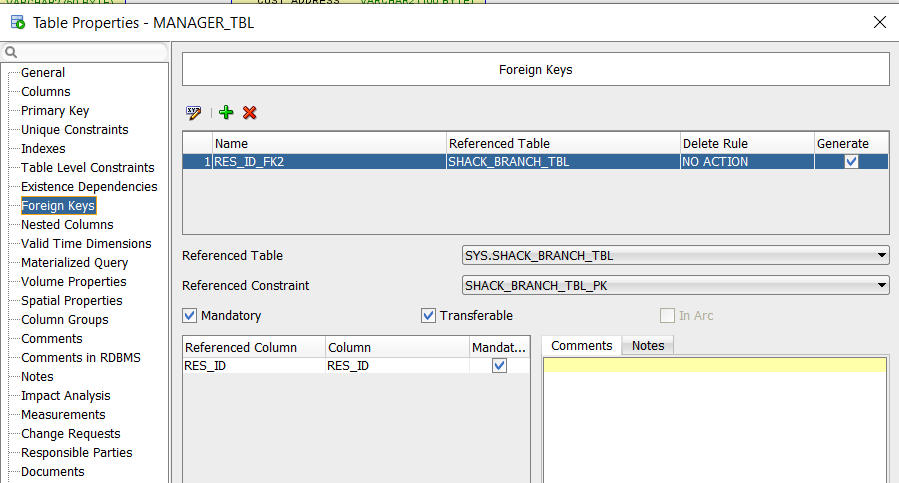
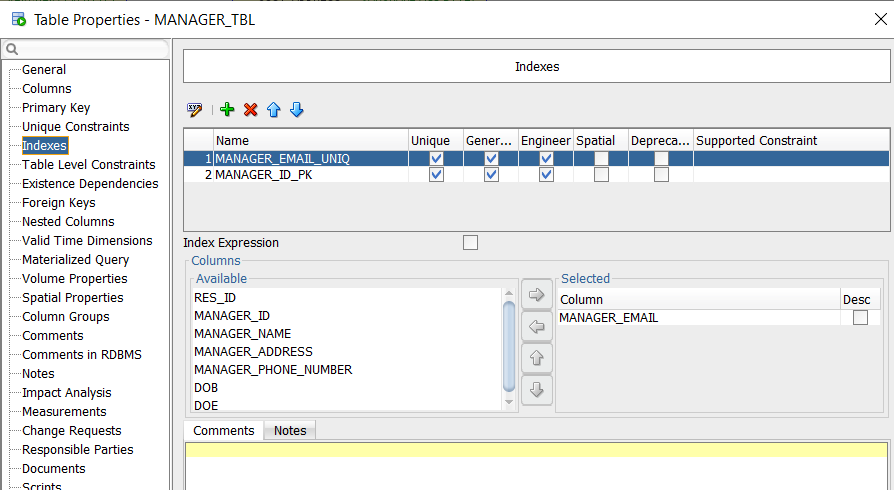
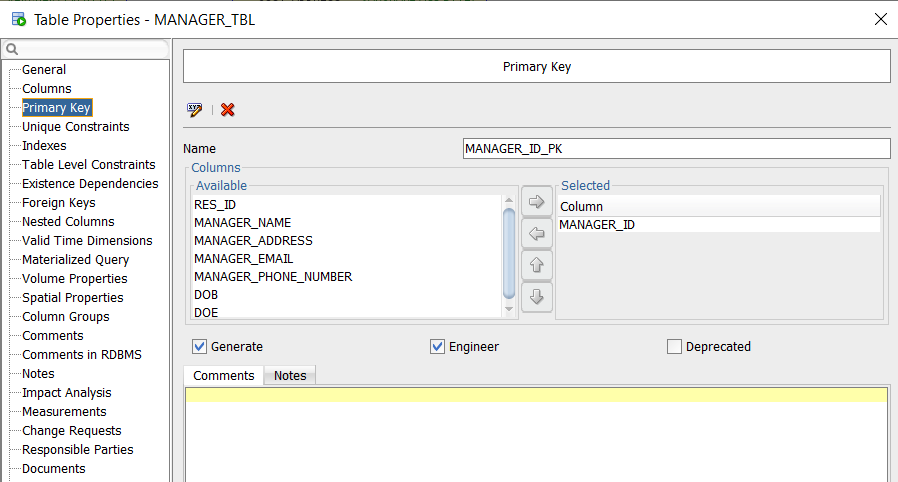


# Unique / Check constraints

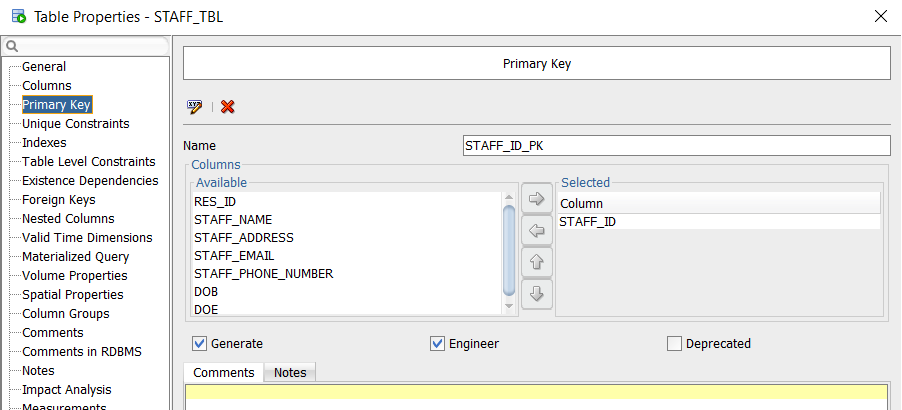
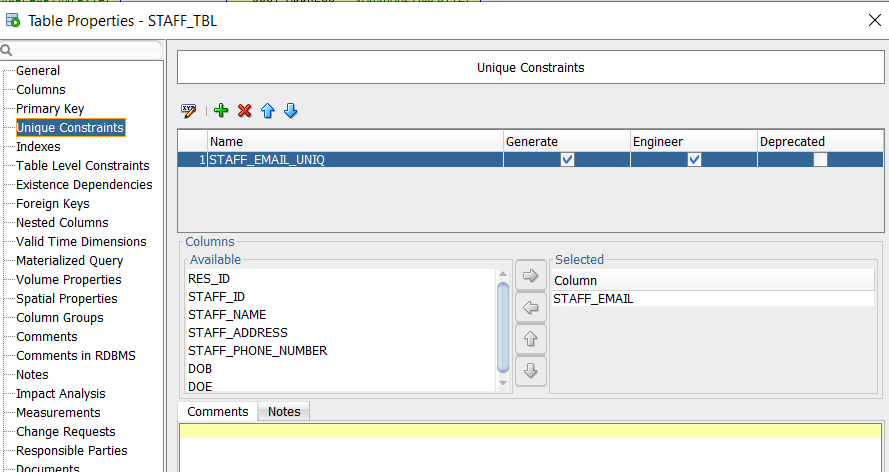
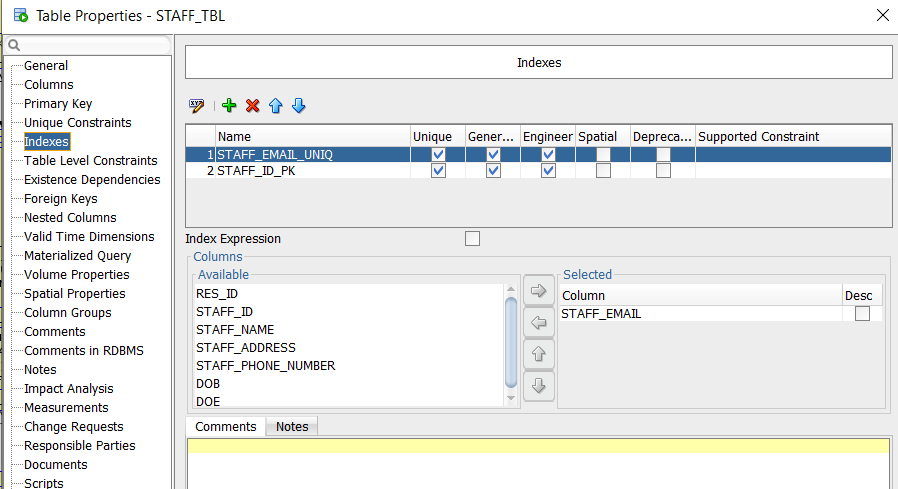
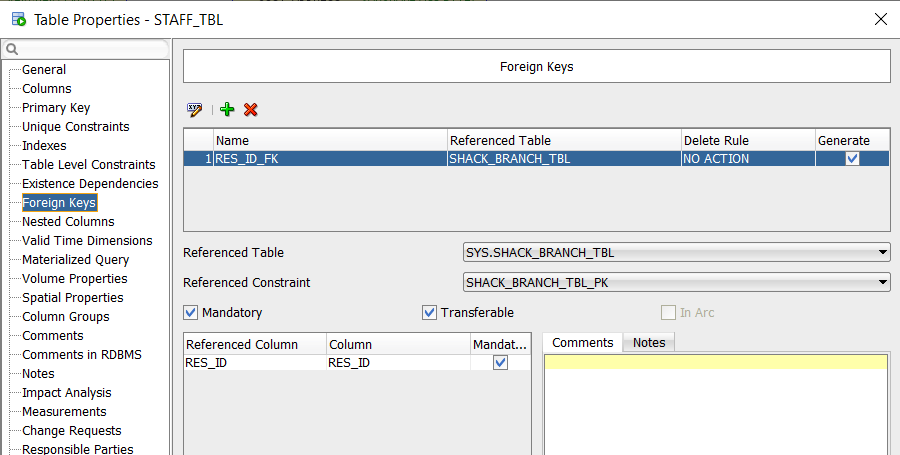
**Burger Shack Branch Table**



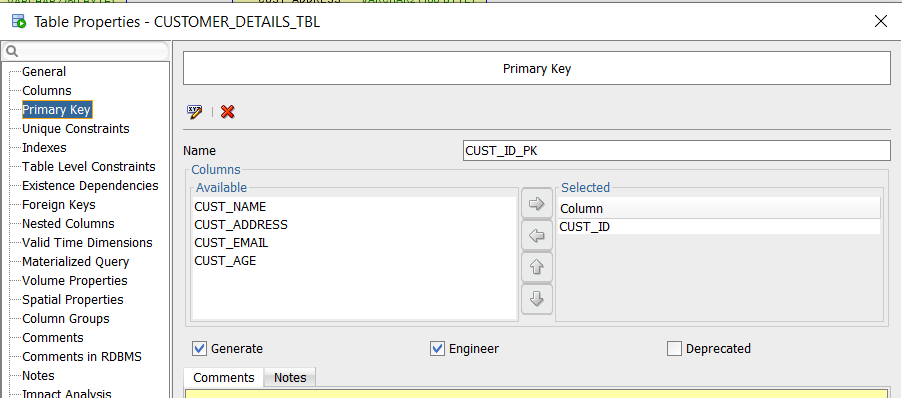
**Manager Table**

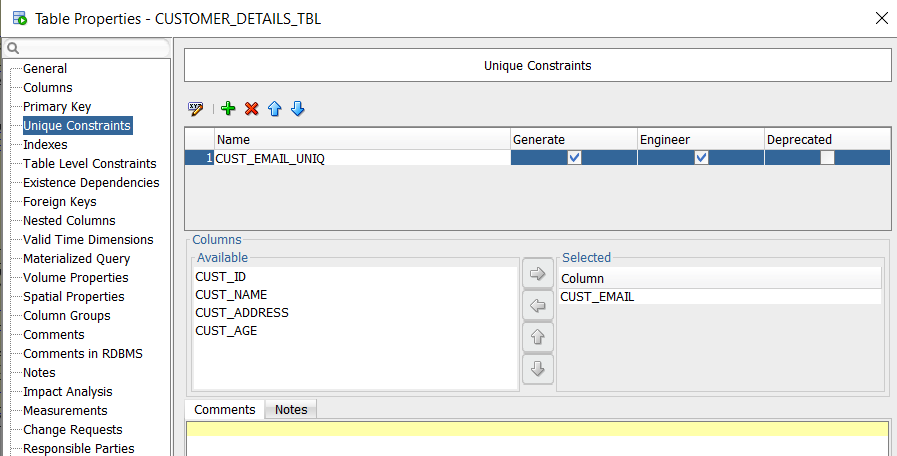


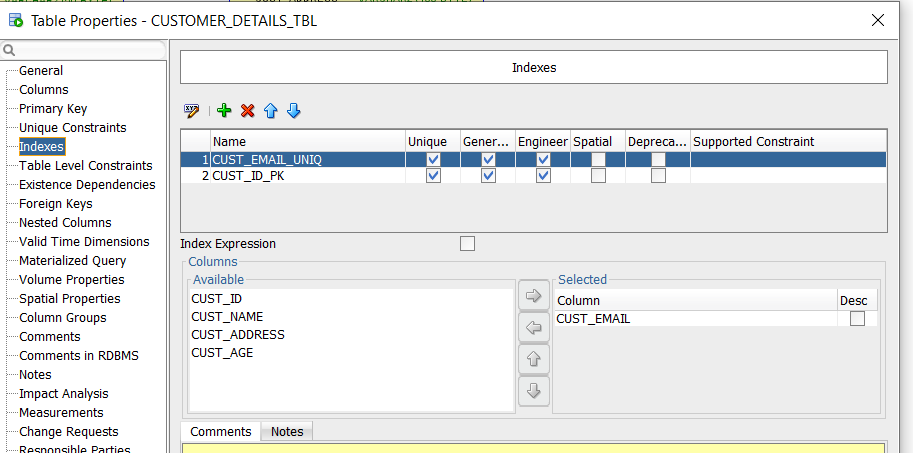
**Staff Table**



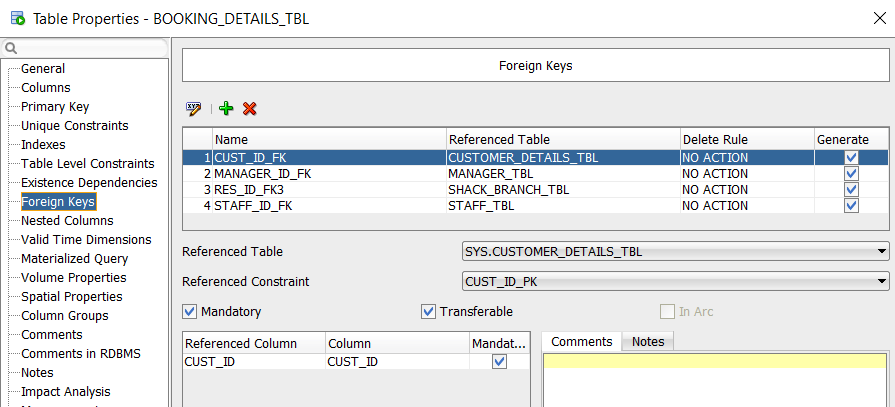
**Customer Details Table**

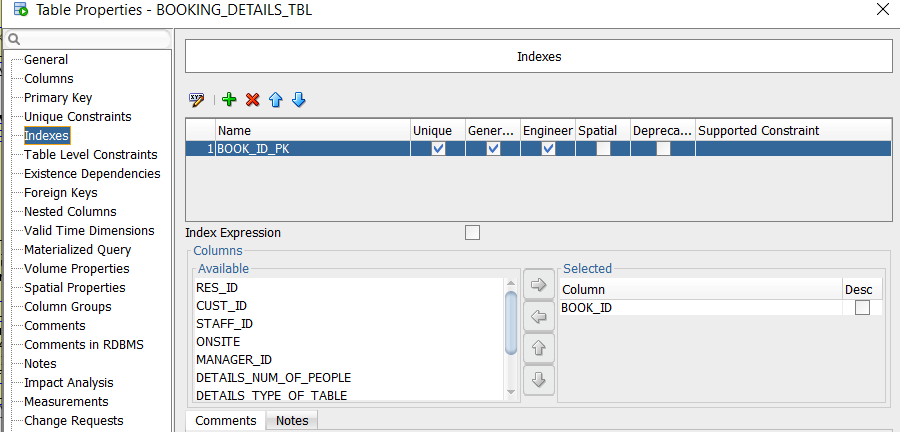


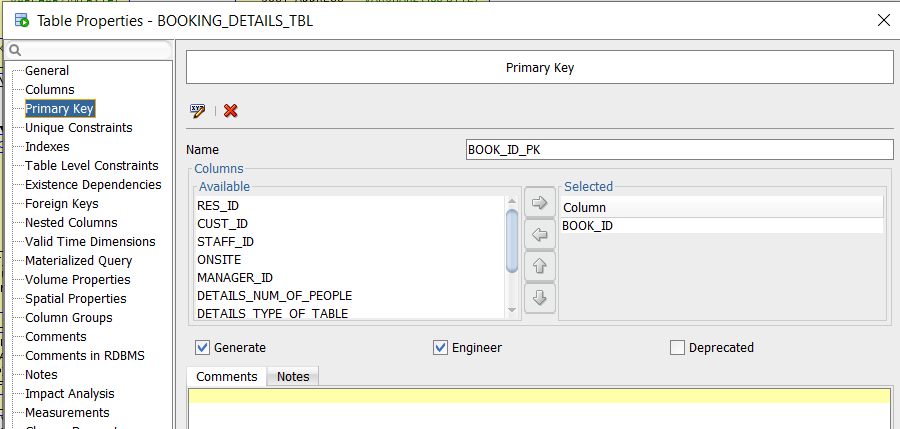




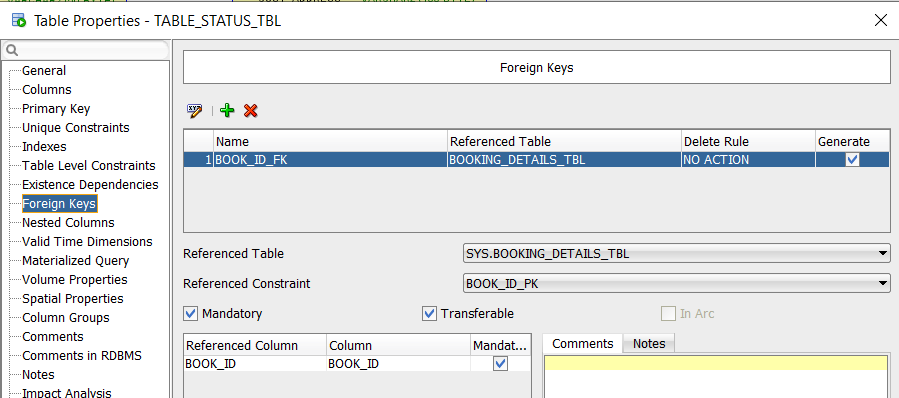
**Booking Details Table**



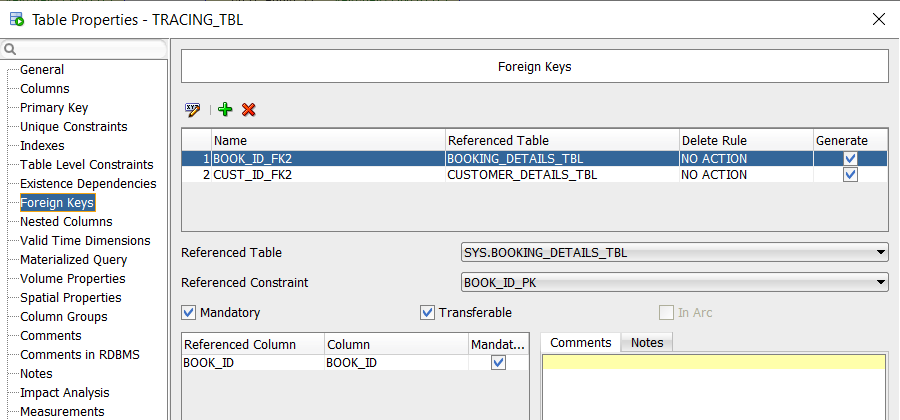




**Table Status Table**



**Contact Tracing Table**



# QUERIES

· One UPDATE/DELETE using a subquery

|  |
| --- |
| */\*Update the table type to 4 where the id corresponds to BI01.*  *Selecting book\_ID from book table where id = BI01 and change column details\_type\_of\_table to 4\*/*  UPDATE booking\_details\_tbl SET details\_type\_of\_table = 4  WHERE book\_ID = (SELECT book\_ID FROM booking\_details\_tbl WHERE book\_ID = 'BI01'). |
| Before:    After: |

· One query using a selection function (CASE/DECODE)

|  |
| --- |
| */\*Check if the customer age is over 18, is 18 or under 18 and display customer name with results. Select columns customer name and age from customer table and display results (over, equal or under 18)\*/*  SELECT cust\_name AS Name, cust\_age AS Age,  CASE WHEN cust\_age > 18 THEN 'Customer age is over 18'  WHEN cust\_age = 18 THEN 'Customer is precisely 18'  WHEN cust\_age < 18 THEN 'Customer is under 18'  END AS IsOverEighteen  FROM customer\_details\_tbl; |

· One INNER join using a GROUP Function

|  |
| --- |
| */\*Count the number of staff working at different branches.*  *Select shack branch and staff table where res\_ID is equal in both tables and group it by the restaurant address\*/*  SELECT shack\_branch\_tbl.res\_address AS Restaurant\_Address, COUNT(staff\_ID) AS Number\_Of\_Staff FROM staff\_tbl  INNER JOIN shack\_branch\_tbl  ON staff\_tbl.res\_ID = shack\_branch\_tbl.res\_ID  GROUP BY shack\_branch\_tbl.res\_address; |

· One LEFT OUTER Join

|  |
| --- |
| */\*Display the list of staff and manager who works for branch Phisborough where their restaurant id*  *corresponds with the restaurant branch\*/*  SELECT shack\_branch\_tbl.res\_address AS Restaurant\_Address,  s.staff\_name AS StaffName,  s.staff\_email, s.DOE AS Employment\_Date,  m.manager\_name, m.manager\_email, m.DOE AS Employment\_Date  FROM staff\_tbl s  LEFT JOIN shack\_branch\_tbl ON shack\_branch\_tbl.res\_ID = s.res\_ID  LEFT JOIN manager\_tbl m ON shack\_branch\_tbl.res\_ID = m.res\_ID  WHERE shack\_branch\_tbl.res\_ID = 'BS05'  ORDER BY shack\_branch\_tbl.res\_address; |

· One RIGHT OUTER Join (using different tables to your left outer join)

|  |
| --- |
| */\*Displaying the list of the main booker and their email, depending on when they have booked. Sorted by their ascending date.\*/*  SELECT c.cust\_name AS Name, c.cust\_email AS Customer\_Email, t.stat\_booked AS Date\_Booked  FROM table\_status\_tbl t  RIGHT JOIN customer\_details\_tbl c ON t.cust\_ID = c.cust\_ID  ORDER BY t.stat\_booked ASC; |

· One UNION

|  |
| --- |
| */\*Display all close contact associated with the main booker if someone tested positive for COVID-19.*  *Selecting the main booker by their ID and displaying all their close contact from the tracing\_tbl\*/*  SELECT cust\_name AS Name, cust\_email AS Email FROM customer\_details\_tbl  WHERE customer\_details\_tbl.cust\_ID = 'CT01'  UNION  SELECT trace\_name AS Close\_Contact, trace\_email FROM tracing\_tbl  WHERE tracing\_tbl.cust\_ID = 'CT01'; |

· One INTERSECT

|  |
| --- |
| */\* Display whoever is on duty \*/*  SELECT staff\_id AS on\_duty FROM staff\_tbl  INTERSECT  SELECT staff\_id FROM booking\_details\_tbl |

· One VIEW (which can use any of the SQL you have previously created)

|  |
| --- |
| */\*Total Customer amount\*/*  CREATE VIEW cust\_ID AS SELECT COUNT(cust\_ID) AS CustNum  FROM customer\_details\_tbl;    */\*Average age\*/*  CREATE VIEW cust\_age AS SELECT AVG(cust\_age) AS Average  FROM customer\_details\_tbl; |

# PENALTY + TIME CALCULATION

|  |
| --- |
| */\* =================== Penalty + Time Calculation ===================\*/*  */\*Query to update the calculated extra time by minutes and the penalty to be paid for by the customer if they exceed the covid-19 policy of 2 hour stay\*/*  UPDATE table\_status\_tbl SET stat\_extra = (((stat\_checkout - stat\_booked) \* 24 \* 60) - 120), stat\_penalty = ((((stat\_checkout - stat\_booked) \* 24 \* 60)- 120) \* 5);    */\*Display the results where if the customer stayed for equal or more than 2 hours\*/*  SELECT c.cust\_name AS Name, s.stat\_extra AS Extra\_Time, s.stat\_penalty AS Euro FROM table\_status\_tbl s  INNER JOIN customer\_details\_tbl c ON s.cust\_ID = c.cust\_ID  WHERE stat\_penalty >= 0; |