**Iteración 3: Manejo transaccional de información**

María Paula Franco Guzmán, Camilo Andrés Sánchez Salamanca

Iteración

Universidad de los Andes, Bogotá, Colombia

{mp.franco10, ca.sanchez38}@uniandes.edu.co

Fecha de presentación: Octubre 29 de 2017

**Tabla de contenido**

[1 Introducción 1](#_Toc497062574)

[2 Etapa 1: Análisis 1](#_Toc497062575)

[3 Etapa 1: Diseño de la aplicación 2](#_Toc497062576)

[4 Etapa 2: Construcción de la aplicación 6](#_Toc497062577)

[5 Bibliografía 6](#_Toc497062578)

# Introducción

En el ámbito de los restaurantes en grandes centros comerciales, los sistemas transaccionales son componentes importantes para realizar las operaciones más importantes de la organización e incrementar la productividad de estas. Estos permiten controlar y administrar múltiples transacciones manteniendo seguridad, persistencia y consistencia de los datos manejados. En el caso de “RotondAndes”, empresa que actúa como intermediaria entre restaurantes y usuarios finales, los sistemas transaccionales permiten a los usuarios consultar, comprar y pagar servicios de alimentación de acuerdo a sus intereses y a las restricciones de los proveedores.

Para realizar una implementación correcta de un sistema transaccional es necesario realizar procesos de diseño y análisis del sistema. El proceso inicial de diseño consiste en identificar los requerimientos funcionales del sistema, conocer las restricciones o condiciones de este y realizar modelos conceptuales que permitan representar el sistema e identificar entidades y relaciones entre estas. Posteriormente, se deben realizar modelos de datos y relaciones a partir de los modelos conceptuales. Finalmente se debe realizar la implementación del diseño realizado en un sistema transaccional y las características de este que le permiten tener las propiedades A.C.I.D. En el presente documento se proponen las modificaciones a los modelos conceptuales de la aplicación utilizada por la organización “RotondAndes” así como a los modelos de datos y relacionales de la aplicación. Se presentan los resultados de la implementación de la aplicación transaccional y se documentan los requerimientos pedidos.

# Etapa 1: Análisis

A continuación se presenta el modelo del mundo:

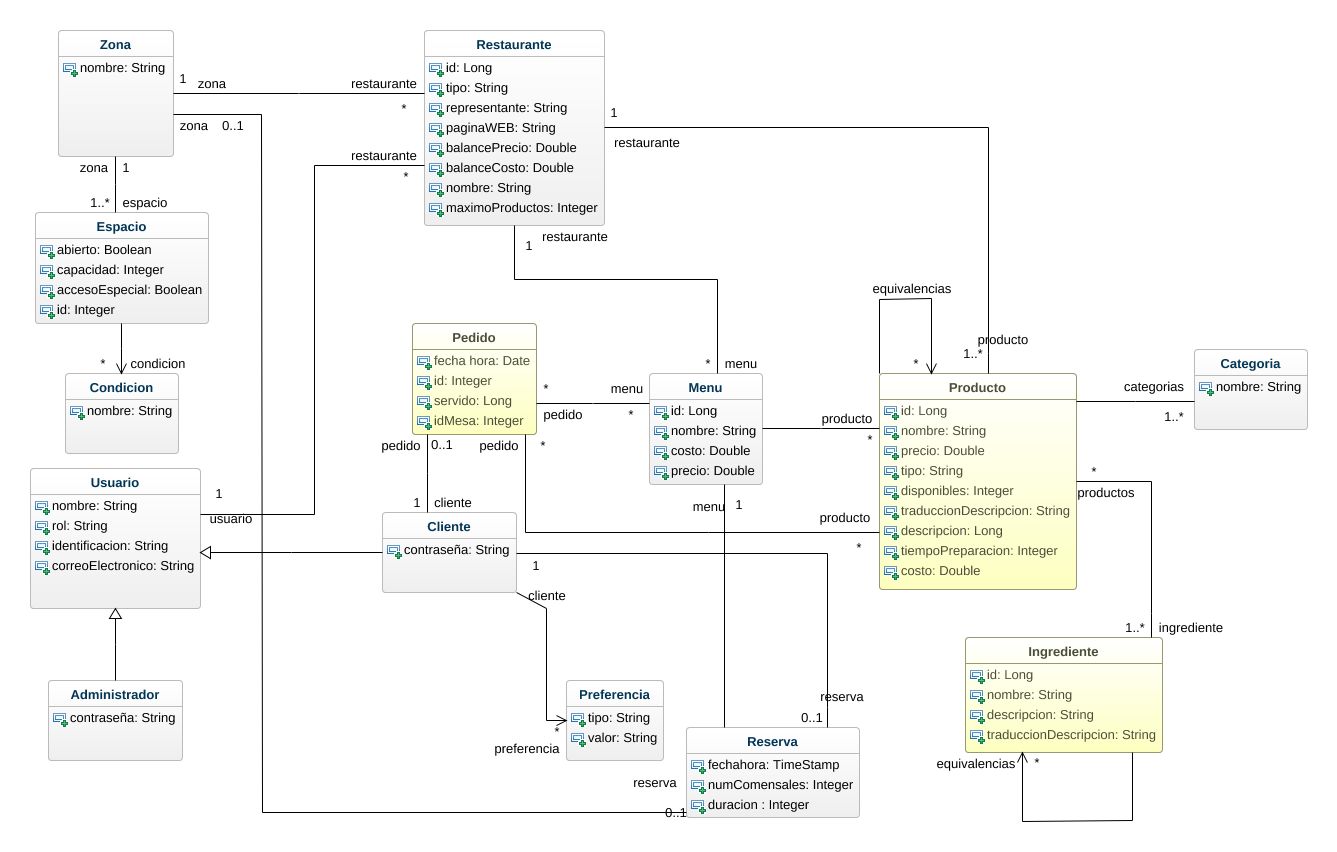


Figura 1 Modelo del mundo modificado

Se ajustó el modelo del mundo como se muestra en la figura. Las clases en amarillo representan las clases en donde se hicieron modificaciones. A pedido se le agregó un atributo idMesa que representa la mesa desde la que se hace el pedido. Para modelar las equivalencias se introdujeron relaciones en la clase producto e ingrediente. Estas representan que un producto tiene 0 o más productos, y así mismo un ingrediente. Si la lista es vacía significa que el producto/ingrediente no tiene equivalencias, y si tiene elementos significa que esos son las equivalencias.

# Etapa 1: Diseño de la aplicación

A nivel del modelo conceptual los nuevos requerimientos modifican las clases que ya se especificaron. Como se introduce la presencia de equivalencias se tienen que tener las relaciones en el modelo que modelan esto. Las restricciones A.C.I.D. no influyen significativamente en el modelo de UML, sino en las transacciones y en el modelo relacional.

A nivel del modelo relacional, se insertó el nuevo atributo de pedido en la tabla de pedido. En la tabla de pedido\_menus que modela los menús que se han pedido se agregaron 5 columnas para modelar cada producto contenido en el menú. La convención es la siguiente: si todas las columnas contienen null significa que se toma el menú original. Si alguna columna contiene un número de id, significa que en ese tipo de producto se toma el producto con id dado como equivalencia. Mediante una transacción debe revisarse que ese producto si corresponda a una equivalencia del producto original del menú.

Se agregaron las tablas de equivalencia de productos e ingredientes, que tienen el id del producto original y el id del producto equivalente. La llave primaria de ambas tablas es la combinación id original con id equivalente. A continuación, se presenta el modelo relacional y se destacan las tablas con cambios con respecto a la iteración anterior.

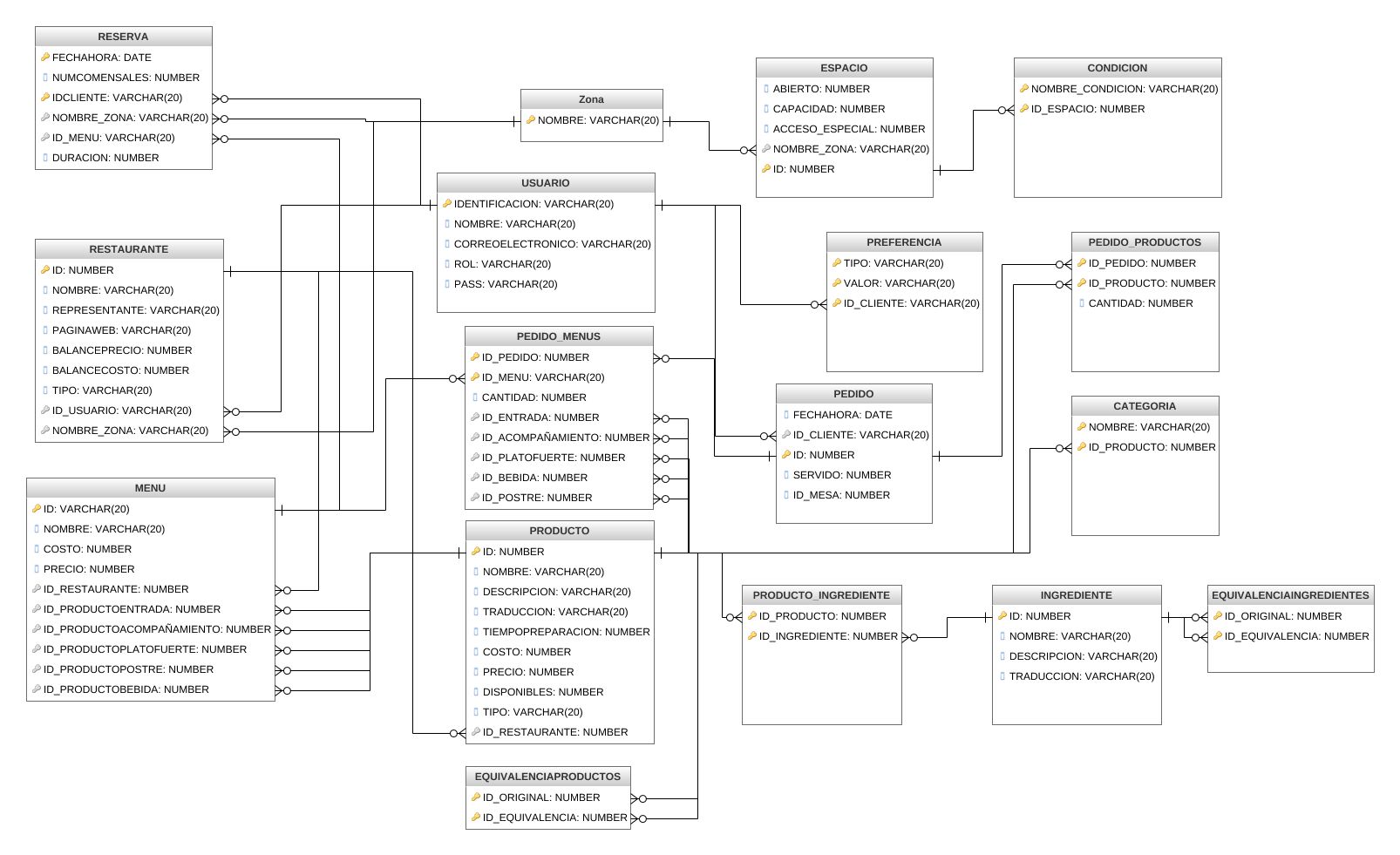


Figura 2 Modelo relacional

En la iteración anterior las tablas se habían diseñado en BCNF, por lo que no se realizaron cambios de calidad. La anormalidad que hay es la de contener los productos del menú en el pedido, pero esto es debido a que el restaurante necesita conocer qué es lo que el cliente quiere pedir. Mediante transacciones se manejan estas columnas.

En cuanto a los comentarios de la iteración anterior, se nos había dicho que el modelado de preferencia y condición de los espacios no era correcto. El enunciado no es muy claro con cuáles valores posibles hay, por lo que no habíamos puesto ninguno. Sin embargo, para mejorar esto implementamos dos restricciones de check en preferencia y en condición para que no se pueda introducir un valor cualquiera sino uno de los dados por nosotros. Esto se puede modificar fácilmente si RotondAndes decide darle la oportunidad a los usuarios de tener nuevas preferencias o de dar nuevas condiciones para los espacios. Los checks utilizados son los siguientes:

En condición:

**NOMBRE\_CONDICION IN ('Barra ensalada','Pago en efectivo','Guardería','Iluminación','Meseros','Música','Parrilla','Pago con tarjeta')**

En preferencia:

**TIPO IN ('Condicion','Espacio','Ingrediente','Precio','Restaurante','Zona')**

En cuanto a la información de las tablas se hicieron consultas en los metadatos como en el taller SQL. A continuación, se presenta la sentencia SQL (que puede ser verificada) y los resultados obtenidos. Como se obtuvieron 80 tuplas los datos se ven muy pequeños, pero si hace zoom puede ver en detalle los resultados obtenidos.

Sentencia SQL:

1. **SELECT** TABLE\_NAME **AS** NOMBRE\_TABLA, COLUMN\_NAME **AS** NOMBRE\_COLUMNA, DATA\_TYPE **AS** TIPO\_DATO, PK\_NAME **AS** NOMBRE\_PK, CK\_NAME **AS** NOMBRE\_CK, FK\_NAME **AS** NOMBRE\_FK
2. **FROM**
3. (**SELECT** \*
4. **FROM**
5. (**SELECT** \*
6. **FROM**
7. (**SELECT** TABLE\_NAME, COLUMN\_NAME, DATA\_TYPE
8. **from** ALL\_TAB\_COLUMNS c
9. **WHERE** c.OWNER = 'ISIS2304A311720'
10. **ORDER** **BY** TABLE\_NAME)
11. NATURAL LEFT OUTER JOIN
12. (**SELECT** C.TABLE\_NAME, C.COLUMN\_NAME, C.CONSTRAINT\_NAME **AS** PK\_NAME
13. **FROM**  ALL\_CONS\_COLUMNS C, ALL\_TAB\_COLUMNS T
14. **WHERE** C.OWNER = 'ISIS2304A311720' AND NOT (C.TABLE\_NAME LIKE 'BIN%') and C.CONSTRAINT\_NAME LIKE '%PK' AND C.OWNER = T.OWNER
15. AND C.TABLE\_NAME = T.TABLE\_NAME AND C.COLUMN\_NAME = T.COLUMN\_NAME
16. **ORDER** **BY** C.TABLE\_NAME))
17. NATURAL LEFT OUTER JOIN
18. (**SELECT** C.TABLE\_NAME, C.COLUMN\_NAME, C.CONSTRAINT\_NAME **AS** CK\_NAME
19. **FROM**  ALL\_CONS\_COLUMNS C, ALL\_TAB\_COLUMNS T
20. **WHERE** C.OWNER = 'ISIS2304A311720' AND NOT (C.TABLE\_NAME LIKE 'BIN%') and C.CONSTRAINT\_NAME LIKE '%CHK%' AND C.OWNER = T.OWNER
21. AND C.TABLE\_NAME = T.TABLE\_NAME AND C.COLUMN\_NAME = T.COLUMN\_NAME
22. **ORDER** **BY** C.TABLE\_NAME))
23. NATURAL LEFT OUTER JOIN
24. (**SELECT** C.TABLE\_NAME, C.COLUMN\_NAME, C.CONSTRAINT\_NAME **AS** FK\_NAME
25. **FROM**  ALL\_CONS\_COLUMNS C, ALL\_TAB\_COLUMNS T
26. **WHERE** C.OWNER = 'ISIS2304A311720' AND NOT (C.TABLE\_NAME LIKE 'BIN%') and C.CONSTRAINT\_NAME LIKE '%FK%' AND C.OWNER = T.OWNER
27. AND C.TABLE\_NAME = T.TABLE\_NAME AND C.COLUMN\_NAME = T.COLUMN\_NAME
28. **ORDER** **BY** C.TABLE\_NAME)

Resultados obtenidos:



Tabla 1Información del modelo relacional

En la tabla se ven todas las columnas de cada tabla, el tipo de dato de las columnas, el nombre de la PK a la que pertenece una columna. Si no pertenece a ninguna aparece vacío (en sqldeveloper aparece null). También se puede ver el nombre del check y el nombre de la llave foránea.

Para garantizar las propiedades ACID de cada requerimiento se toman por defecto los valores de aislamiento y lectura de Oracle, ya que se tiene el “READ COMMITED” por defecto, que solo lee cambios que se han confirmado, lo cual brinda datos consistentes, pues ninguna transacción hace cambios sin el autocommit. Adicionalmente, al realizarse validaciones en el mundo del problema se evitan inconsistencias. Además como las tablas están en BCNF no se presentan muchos problemas al realizar una utilización concurrente.

# Etapa 2: Construcción de la aplicación

Para las pruebas los datos se generaron manualmente. No hay una gran cantidad de datos, pero hay suficientes datos para revisar los requerimientos y son consistentes entre ellos. En la carpeta docs está el archivo sql DB.sql que se ejecuta para reiniciar la base de datos. No se ejecuta directamente desde eclipse sino que se copia y se pega a sqldeveloper, y luego este se ejecuta como script.

A continuación, se presentan las pruebas (escenarios de prueba), especificaciones y demás datos sobre cada requerimiento. Se presentan los datos requeridos en este documento y no en un Excel para tener más orden.

**RF1 – Registrar Usuario**

En este requerimiento simplemente se agrega un usuario a RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/usuarios

* Registrar nuevo usuario

JSON:

{

"rol": "No registrado",

"name": "Andres",

"identificacion": "40",

"contraseña": null,

"correoEletronico": "andresca2@hotmail.com"

}

* Registrar un usuario existente

JSON:

{

"rol": "No registrado",

"name": "Andres",

"identificacion": "40",

"contraseña": null,

"correoEletronico": "andresca2@hotmail.com"

}

Con el GET puede obtener todos los usuarios en RotondAndes.

**RF2 – Registrar Cliente**

En este requerimiento se registra a un usuario con rol cliente en RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/administradores/clientes

* Agregar un cliente nuevo

JSON:

{

"rol": "Cliente",

"name": "Andy",

"identificacion": "41",

"contraseña": "alzatucerveza",

"correoEletronico": "andy\_sobrio@gmail.com"

}

* Agregar un cliente existente

JSON:

{

"rol": "Cliente",

"name": "Andy",

"identificacion": "3",

"contraseña": "alzatucerveza",

"correoEletronico": "andy\_sobrio@gmail.com"

}

Con GET puede obtener los usuarios con rol cliente en RotondAndes.

**RF3 – Registrar Restaurante**

En este requerimiento se registra a un restaurante en RotondAndes con una sentencia de insert.

URL Para obtener los restaurantes: localhost:8080/RotondAndes/rest/restaurantes

URL Para registrar:

* Registrar un nuevo restaurante

JSON: localhost:8080/RotondAndes/rest/administradores/restaurantes

{

"id":20,

"nombre": "Chick And Chips2",

"tipo": null,

"representante": "Camilo Sánchez",

"paginaWeb": "chicaachips.com",

"balancePrecio": 400000,

"balanceCosto": 200000,

"usuario": 17,

"nombreZona": "Oriental"

}

* Registrar restaurante existente

JSON:

{

"id":5,

"nombre": "Chick And Chips",

"tipo": null,

"representante": "Camilo Sánchez",

"paginaWeb": "chic&chips.com",

"balancePrecio": 400000,

"balanceCosto": 20000,

"usuario": 17,

"nombreZona": "Oriental"

}

**RF4 – Registrar Producto**

En este requerimiento se registra a un producto en RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/restaurantes/{id}/productos

En este caso id=1

* Registrar nuevo producto

JSON:

{

"id": 30,

"nombre": "Hamburguesa Gigante",

"descripcion": "Con carne",

"traduccion": "With meat",

"tiempoPreparacion": 18,

"costo": 14000,

"precio": 25000,

"disponibles": 10,

"tipo": "Plato fuerte",

"idRestaurante": 1,

"maximoProductos":20

}

* Registrar un producto existente

JSON:

{

"id": 2,

"nombre": "Hamburguesa Gigante",

"tipo": "Plato fuerte",

"disponibles": 10,

"tiempoPreparacion": 18,

"precio": 25000,

"costo": 14000,

"descripcion": "Con carne",

"idRestaurante": 1,

"traduccion": "With meat",

"maximoProductos" : 20

}

Con GET puede obtener los productos de ese restaurante.

**RF5 – Registrar Ingrediente**

En este requerimiento se registra a un ingrediente de un producto en RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/restaurantes/1/productos/3/ingredientes

Con esa ruta puede obtener todos los ingredientes del producto de un restaurante.

URL: localhost:8080/RotondAndes/rest/restaurantes/1/productos/14/ingredientes

* Registrar un ingrediente

JSON:

{

"id": 21,

"nombre": "Frijoles",

"descripcion": "Rojos",

"traduccion": "Red"

}

* Registrar un ingrediente existente

JSON:

{

"id": 21,

"nombre": "Frijoles",

"descripcion": "Rojos",

"traduccion": "Red"

}

**RF6 – Registrar Menú**

En este requerimiento se registra a un menú en RotondAndes con una sentencia de insert. Se revisa que sus productos pertenezcan al restaurante que posee al menú.

URL: localhost:8080/RotondAndes/rest/restaurantes/menus

* Registrar menu con FK existente

JSON:

{

"id": "20",

"nombre": "Burritox",

"costo": 90000,

"precio": 130000,

"idrestaurante": 5,

"idplatoFuerte": 6

}

* Registrar menu que ya existe

JSON:

{

"id": "1",

"nombre": "Ajiaquito",

"costo": 10000,

"precio": 25000,

"idrestaurante": 2,

"identrada": 0,

"idacompañamiento": 0,

"idplatoFuerte": 9,

"idpostre": 0,

"idbebida": 0

}

* Registrar menu con FK no existente

JSON:

{

"id": "17",

"nombre": "Burritoss",

"costo": 90000,

"precio": 130000,

"idrestaurante": 5,

"idplatoFuerte": 24

}

* Registrar menú que ya se agregó

JSON:

{

"id": "20",

"nombre": "Burritox",

"costo": 90000,

"precio": 130000,

"idrestaurante": 5,

"idplatoFuerte": 6

}

* Registrar menú con producto de otro restaurante

JSON:

{

"id": "25",

"nombre": "Burro carne",

"costo": 90000,

"precio": 130000,

"idrestaurante": 5,

"idplatoFuerte": 4

}

**RF7 – Registrar Zona**

En este requerimiento se registra a una zona con rol cliente en RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/administradores/zonas

* Agregar una zona nueva

JSON:

{

"name": "De prueba 3"

}

* Agregar una zona que ya existe

JSON:

{

"name": "Oriental"

}

* Agregar la misma zona 2 veces

JSON:

{

"name": "De prueba 3"

}

**RF8 – Registrar Preferencia de Cliente**

En este requerimiento se registra a una preferencia de un cliente en RotondAndes con una sentencia de insert.

URL: localhost:8080/RotondAndes/rest/clientes/0/preferencias

* Registrar una preferencia nueva

JSON:

{

"tipo": "Condicion",

"valor": "Tarjeta de credito",

"idCliente": 2

}

* Registrar una preferencia existente

JSON:

{

"tipo": "Condicion",

"valor": "Tarjeta de credito",

"idCliente": 2

}

Con GET puede obtener las preferencias.

**RF9 – Registrar Pedido de un Producto**

Como este requerimiento se editó, se presenta su desarrollo en el RF14.

**RF10 – Registrar el Servicio de un Producto**

En este requerimiento cambia el atributo de servido de un pedido que tenga al producto especificado.

URL:

JSON:

**RF11 – Registrar Equivalencia de Ingredientes**

En este requerimiento se registra una equivalencia de ingredientes en RotondAndes con una sentencia de insert.

URL:

JSON:

**RF12 – Registrar Equivalencia de Productos**

En este requerimiento se registra una equivalencia de productos en RotondAndes con una sentencia de insert.

URL:

JSON:

**RF13 – Surtir restaurante**

En este requerimiento se asigna al valor de unidades disponibles de un producto su valor máximo.

URL:

JSON:

**RF14 – Registrar pedido de un producto – Con equivalencias**

URL:

JSON:

**RF15 – Registrar pedido de productos de una mesa**

URL:

JSON:

**RF16 – Registrar el Servicio de una mesa**

URL:

JSON:

**RF17 – Cancelar Pedido**

URL:

JSON:

**RFC1 – Consultar los productos servidos en RotondAndes**

POST URL: localhost:8080/RotondAndes/rest/usuarios/productos

* Consultar productos servidos con filtro restaurante

JSON:

{

"idRestaurante": 1,

"categoria":null,

"fechas":null,

"precios":null

}

* Consultar productos servidos sin filtros

JSON:

{

"idRestaurante":null,

"categoria":null,

"fechas":null,

"precios":null

}

* Consultar productos servidos con filtro categoría

JSON:

{

"idRestaurante": null,

"categoria":"Comida rapida",

"fechas":null,

"precios":null

}

* Consultar productos servidos con filtro fecha

JSON:

{

"idRestaurante": null,

"categoria":null,

"fechas":"2017-10-01 00:10:00 y 2017-10-04 23:40:00",

"precios":null

}

* Consultar productos servidos con filtro precio

JSON:

{

"idRestaurante": null,

"categoria":null,

"fechas":null,

"precios":"5000:8000"

}

**RFC2 – Consultar una zona**

* Consultar una zona con parámetro de ordenamiento inválido

URL: localhost:8080/RotondAndes/rest/usuarios/zonas/Occidental/Numero

* Consultar una zona que no existe

localhost:8080/RotondAndes/rest/usuarios/zonas/Suroriental/NOMBRE\_PRODUCTO\_SERVIDO

* Consultar zona existente con parámetro válido

localhost:8080/RotondAndes/rest/usuarios/zonas/Suroriental/NOMBRE\_PRODUCTO\_SERVIDO

**RFC3– Consultar los clientes**

* Consultar un cliente con id existente

URL: localhost:8080/RotondAndes/rest/usuarios/clientes/1

* Consultar un usuario que no es cliente y existe

URL: localhost:8080/RotondAndes/rest/usuarios/clientes/12

* Consultar un usuario inexistente

URL: localhost:8080/RotondAndes/rest/usuarios/clientes/88

**RFC7 – Consultar el consumo de un cliente registrado**

* Consultar el consumo de un cliente desde administrador

URL: localhost:8080/RotondAndes/rest/administradores/consumo/1

* Consultar un cliente que no existe

URL: localhost:8080/RotondAndes/rest/administradores/consumo/90

* Consultar consumo cliente que no ha consumido

URL: localhost:8080/RotondAndes/rest/administradores/consumo/3

* Consulta de un cliente de su consumo

URL: localhost:8080/RotondAndes/rest/clientes/1/consumo

**RFC8 – Consultar pedidos**

* Ver pedidos hechos por usuarios registrados desde administrador

URL: localhost:8080/RotondAndes/rest/administradores/pedidos/registrado

* Ver pedidos hechos por usuarios no registrados desde administrador

URL: localhost:8080/RotondAndes/rest/administradores/pedidos/noregistrado

* Ver pedidos con parámetro inválido

URL: localhost:8080/RotondAndes/rest/administradores/pedidos/noregis

* Ver pedidos hechos por usuarios registrados desde restaurante

URL: localhost:8080/RotondAndes/rest/restaurantes/7/pedidos/registrado

# Bibliografía

|  |  |
| --- | --- |
| [1] | Oracle, «Oracle Docs,» Oracle Help Center , 2017. [En línea]. Available: https://docs.oracle.com/en/. |
| [2] | W3Schools, «SQL Tutorial,» W3Schools, 2017. [En línea]. Available: https://www.w3schools.com/sql/default.asp |