CODERHOUSE

Proyecto Final

Segunda entrega

1. Vistas

Vista1: promedios_alumnos

Objetivo: Obtener una tabla que posea los datos de los alumnos y los promedios obtenidos en el total de sus calificaciones.

Tablas/Datos: alumnos, calificaciones

Descripción: Se realiza un join entre las tablas *alumnos* y *calificaciones* para obtener las calificaciones que los alumnos han obtenido, luego se calcula el promedio agrupando por cada alumno.

Vista2: alumnos_por_provincia

Objetivo: Obtener una tabla que muestre la provincia de origen de la totalidad de alumnos, puntualmente, la cantidad de alumnos asociados a cada provincia.

Tablas/Datos: provincias, alumnos

Descripción: Se realiza un join entre la tabla *provincias* y *alumnos* para obtener todos los alumnos relacionados con cada provincia. Luego se agrupa por provincia, contando la cantidad de registros. Se realiza un LEFT JOIN para que el resultado contengan la totalidad de provincias, sin importar si tienen o no alumnos.

Vista3: facturacion_historica_alumnos

Objetivo: Obtener el total facturado por alumno.

Tablas/Datos: matriculas_facturadas, facturas, matriculas, alumnos

Descripción: Se realiza un join de las tablas mencionadas anteriormente, para lograr obtener las *matriculas_facturadas* y los datos de los alumnos (como nombre y apellido). La unión se realizó en 4 tablas debido a que las mismas se relacionan en cadena. Con los datos obtenidos, se agrupa por alumno y se obtiene la suma del importe obtenido de la tabla *matriculas_facturadas*.

Vista4: matriculas_pendientes_facturar

Objetivo: Obtener un listado de aquellas matrículas generadas por los alumnos, pero que aún no fueron facturadas a los mismos (es decir, no se emitió un comprobante de factura).

Tablas/Datos: matriculas, matriculas_facturadas

Descripción: Se realizó un join entre las *matrículas* y las *matrículas_facturadas*, y luego se descartaron aquellas matriculas que ya se encontraban facturadas, quedando solamente las pendientes de facturar. Luego, se totaliza por periodo lectivo, semestre y se obtiene la suma del importe pendiente de facturar.

Vista5: facturas_impagas

Objetivo: Obtener aquellas facturas que a la fecha no poseen cobranzas asociadas.

Tablas/Datos: facturas, cobranzas

Descripción: Se realizó un join entre ambas tablas para obtener todas las cobranzas asociadas a facturas, y se descartaron aquellas facturas que tenían cobranzas. No se usaron criterios de agregación o agrupamiento, sino que se trata de una lista detallada de facturas pendiente de cobro.

Vista6: facturacion_duplicada

Objetivo: Obtener un detalle de aquellas matrículas que fueron incluidas erróneamente en más de una factura.

Tablas/Datos: matriculas_facturadas, facturas

Descripción: Se procedió a realizar un join entre las tablas mencionadas para tener detalle que matrícula se incluye en cada factura. Luego se filtraron aquellas que figuraban en más de una factura. Para realizar dicho filtro, se utilizó una subconsulta que obtiene aquellos id de matrículas que se encuentran duplicados.

Vista7: alumnos_promocionados

Objetivo: Listar todos los alumnos que hayan obtenido la condición de "Promoción" en sus materias cursadas.

Tablas/Datos: cursadas, alumnos, profesores, materias

Descripción: Se unieron las tablas mencionadas, y se filtró aquellos alumnos que poseían la condición de "Promoción". Se reportan datos del alumno y del profesor, así como periodo lectivo, y semestre.

2. Funciones

Función1: f_nota_texto

Objetivo: Esta función devuelve un texto asociado a la nota obtenida por el alumno. Se obtuvo la escala de ejemplo de la siguiente página (Facultad Ciencias Económicas). https://www.eco.unc.edu.ar/reglamentaciones/#escala-de-calificaciones-ordenanza-n-482-2009

Tablas/Datos: no aplicable.

Descripción: usando una estructura case, se evalúa en que rango se encuentra la nota obtenida, para retornar distintos textos.

Función2: f_materias_aprobadas

Objetivo: Esta función cuenta la cantidad de materias aprobadas para un alumno dado, en la determinada carrera. Se agrega la carrera porque el alumno puede estar cursando más de una carrera.

Tablas/Datos: alumnos, carreras, calificaciones

Descripción: En primer lugar, se verifica si los parámetros de legajo de alumno y el id de carrera existen en las tablas correspondientes. En caso de que no, se lanza una excepción. En caso de que existan, se guarda en una variable la cantidad de notas mayores a 4 (estado aprobado) que el alumno tiene en la carrera determinada.

3. Stored Procedures

Procedure1: OrdenarTabla

Objetivo: Consultar a una tabla pasada por parámetro, ordenando por un campo pasado también por parámetro. Asimismo, debe especificarse ASC o DESC para indicar el modo de orden. En caso de estar vacío el modo de orden, se realizará un orden ascendente.

Tablas/Datos: tabla a ordenar.

Descripción: Se va creando de a partes una prepared statement, indicando la tabla, campo de ordenación y sentido de orden (ascendente/descendente). Se validan los parámetros ingresados, en caso de que no sean los esperados, se lanza una excepción. Una vez construida la sentencia preparada, se ejecuta y se libera de la memoria.

Procedure2: CalcularPromedioAlumno

Objetivo: Devuelve en un parámetro de salida el promedio de las notas obtenidos por un alumno.

Tablas/Datos: calificaciones

Descripción: Se realiza una consulta a la tabla mencionada, filtrando por el alumno recibido por parámetro, y se guarda en una variable de salida, el promedio de dichas notas obtenidas.

Procedure3: RegistrarCobranza

Objetivo: Registra una cobranza de una factura existente en la tabla facturas.

Tablas/Datos: cobranzas

Descripción: Realiza una inserción de un registro en la tabla mencionada, con los parámetros recibidos (fecha, factura que se cobró y medio de pago utilizado).

Procedure4: CambiarDirectorCarrera

Objetivo: Asigna otro profesor como director de carrera. Se le debe indicar la carrera a actualizar y el nombre y apellido del profesor que asumirá el rol de director.

Tablas/Datos: profesores, carreras

Descripción: Dado que se poseen nombre de alumno y descripción de carrera, ambos campos que no son claves primarias de sus tablas, se procede a determinar los mismos haciendo consultas a las tablas mencionada para tener el legajo y el id correspondiente. Con dichos datos, se hace una actualización del registro de la carrera para asignar el campo director con el legajo del profesor que se obtuvo a través del nombre pasado por parámetro.

4. Triggers

Trigger1: notas

Objetivo: Analiza la validez de la nota (rango entre 0 y 10), y setea el campo aprobado, considerando que con 4 ya se obtiene dicha condición.

Tablas/Datos: calificaciones

Descripción: En primera instancia se valida si la nota se encuentra fuera del rango, seteando alguno de los valores extremos (0 o 10) en caso de estar por fuera del intervalo. Luego, si la nota es mayor o igual a 4, se setea el campo aprobado en 1, o 0 en su defecto. Una vez realizados todos estas validaciones y definiciones, se continua el proceso de inserción en la tabla calificaciones.

Trigger2: log_modif_alumnos

Objetivo: Registra sólo las modificaciones que sufre cada registro y campo de la tabla alumnos. Aclaración: no registra inserciones ni eliminaciones.

Tablas/Datos: alumnos, log_modif_alumnos

Descripción: Este trigger se ejecuta luego de cada actualización de un registro de la tabla alumnos, no teniendo efecto sobre inserciones o eliminaciones. El procedimiento consiste en validar uno a uno los atributos de la tabla alumnos, y en caso de que el valor anterior difiera del actualizado, se procede a insertar un registro en la tabla *log_modif_alumnos* con detalle del campo cambiado, valor anterior, valor nuevo, fecha y hora de la actualización, y usuario que ejecutó la sentencia.

Trigger3: log_insert/update/delete

Objetivo: Registra los eventos realizados sobre cada id de la tabla materias. Aclaración: no registra valores anteriores y nuevos.

Tablas/Datos: log_tablas, y a modo de ejemplo: materias

Descripción: En este caso se trata de un conjunto de triggers (uno para cada evento de inserción, actualización y eliminación) que registran en la tabla *log_tablas* el detalle de que operación se hizo, para que registro de una entidad particular. Con el objetivo de lograr este cometido, se procedió a crear un stored procedure denominado RegistrarEntradaLog que recibe por parámetro la entidad (tabla), el id de la novedad y la operación realizada (insert/update/delete). Este SP facilita la inserción de registros en la tabla de log. Luego, como ejemplo, se creó para la tabla *materias* un trigger para cada operación, haciendo llamada del SP con los datos necesarios con posterioridad a cada operación.

5. Archivos SQL

4.1 Script creación de objetos:

Colocar link a archivo .sql

- Views: https://drive.google.com/file/d/1sek7tcpBNcLSyuwmFqL7Lqm9JSR9EAto/view
- Functions: https://drive.google.com/file/d/1LnfDRsBp3BWjfXjWMWimBAnWX-_d9-Zi/view
- Stored Procedures: https://drive.google.com/file/d/1qb5S8vES2sgfq7hhY-ZNlN9o4fMkUOxf/view
- Triggers: https://drive.google.com/file/d/1fJluvDcMdXYyMYELJYCvucTaK_1L3y2n/view

4.2 Script inserción de datos:

Archivo completo (definición tablas, inserción datos y sentencias punto 4.1).
Link: https://drive.google.com/file/d/11jYCBleXRzqW1fw7f_ASDYIeAf3dSwzT/view

Carpeta con los 5 archivos (puntos 4.1 y 4.2).
Link: https://drive.google.com/drive/folders/1qJuIvjSpYy1_HNDS_Ae6eVB4J6I_rfmZ