1. Relational Database Development

152-156

SQL: Updating Data

| Notes | Activity |
| --- | --- |
| 1. Text References    * [Copying Table Structure](#CopyTable) Pages 162    * [Copying Data Across Tables](#CopyRecords) Page 163    * [Updating Existing Data](#UpdateData) Pages 163 – 165  Pages 167 – 168    * [Transaction Rollback and Commit](#Rollback) Pages 172 – 174    * [Deleting Rows](#DeleteRows) Pages 166 – 167    * [Adding Columns to a Table](#AddColumn) Page 168 – 169    * [Deleting a Column](#DropColumn) Page    * [Changing a Column Definition](#ModifyColumn) Pages 171 – 172    * [Rename a Column](#RenameColumn) Page    * [Remove a Key Column](#RemoveKey) Page    * [Add a Key Column](#AddKey) Page    * [Rename a Table](#RenameTable) Page    * [Delete a Table](#DeleteTable) Page 175 | |
| 1. Copying a Table’s Structure    * MySQL makes it easy to create a new table based on another table’s structure  **Create** **Table** newTableName **Like** currTableName    * The new table will have the same set of fields and field definitions as the current table    * The new table will NOT have data    * Other versions of SQL do not have the **Like** command. In those versions you have to create the new table manually with the same fields as the original table. | Create table tblFebTrips with the same structure as tblTrips  Display the structure of the new table |
| 1. Copying Data from One Table to Another    * The **Insert** command you learned about in Chapter 2 can also be used to copy data from one table to another.      + The two tables must be of similar structure but the field names or even the field sizes don’t have to be the same        - Errors could occur if field sizes are different      + Don’t include auto number fields; new numbers will be assigned to the inserted record automatically.        - Note: this could cause problems if the table needs to be linked with others.   **Insert Into** *tablename* **Select** *fieldlist or \** **From** *tablename2* [**Where** *condition*]   * + - The **Where** clause is optional. If it is omitted all records from the source table are copied to the destination table. | Copy the February trips from tblTrips into tblFebTrips  (like “%-02-%”) |
| 1. Updating Existing Data    * The **Update** command allows you to change existing data in a table without deleting the record and re-entering it.    * **Update** *TABLENAME* **Set** *fieldname* = *newvalue* [,*fieldname = newvalue*] **Where** *condition*      + **Set** keyword and clause are not optional      + To change multiple fields at once, separate each field and value with commas      + **Where** clause is optional, but if omitted all rows are changed.      + **Where** clause often dependent on primary key (though not required to be)    * To update a field value to **Null**,  **Set** *fieldname* = **Null**       + No apostrophes, no **Is**    * New value can be a calculated value using any database fields.    * Note the MySQL Workbench prohibits Delete and Update commands that don’t have a Where clause or where the Where clause does NOT include the primary key. You can turn off this feature:      + EditPreferencesSQL Queries”Safe Updates”      + Or, you can enter the following MySQL command: set sql\_safe\_updates = 0; | Change the foreman of the Polonia plant to Gaul, Volker  Remove the foreman phone number of the Polonia plant. |
| * + You can also update using fields from multiple tables by joining the tables or using subqueries. |  |
| (Assume a inventoryvalue column has been added to tblinventory  in Henry Books)  update tblinventory i  inner join tblbooks b on i.bookcode=b.bookcode  set inventoryvalue = onhand \* price;  (Assume a totalonhand column has been added to tblbooks in Henry  Books)  Update tblbooks b  Set totalonhand =  (Select sum(onhand) from tblinventory i  Where i.bookcode=b.bookcode); | |
| 1. Rollback    * Any changes to the data in the database are temporary.      + Changes to table structure are immediate.    * Changes can be undone any time before you log off the database.    * **Rollback** command restores all data to what it was when the transaction began.      + The MySQL Workbench is in *auto-commit* mode by default. All commands are committed as soon as they are entered correctly (no rollback)        - To allow rollbacks, you’ll have to turn auto-commit off by clicking the *Toggle auto-commit mode* toolbar button (see more info below)        - Alternatively, you can type the command  Set Autocommit = 0      + If a **Commit** command has been issued (see next section), **Rollback** restores the data to what is was just after the **Commit** | Use this for all further examples  Turn off auto-commit  Remove the foreman name from Polonia  Rollback (verify) |
| 1. Commit    * **Commit** makes all changes to the data during this transaction permanent and ends the transaction | Change the foreman name in Polonia to Smith, Steve  Commit  Rollback (verify Smith) |
| * + When the *Toggle auto-commit mode* toolbar button  is OFF, the MySQL Workbench provides Rollback  and Commit  buttons that work exactly the same as the MySQL commands.     - I use these buttons to determine if auto-commit is off. If this button is active, auto-commit if off (auto-commit button appearance for on/off is not that different). |  |
| 1. Deleting Rows    * **Delete** **From** *TABLENAME* **Where** *condition* (optional)    * If no condition, deletes all records.      + If primary key not part of Where clause, need to turn off “Safe Update” in MySQL Workbench (see above) | From tblFebTrips, delete all trips to Miami  Display the number of trips to each city (FebTrips)  Rollback (and verify) |
| 1. Changing Table Structure    * All the commands above change the data in the database. This command changes the database’s underlying structure.    * **Alter Table** *TABLENAME* *clauses*      + *Clauses* designates what kind of changes to make. |  |
| * + Note: multiple *clauses* (described next) can be added to one alter command, separated by commas   Example:  **Alter Table** *TABLENAME*  **Add country varchar(30),**  **Drop continent,**  **Change postalCode zipCode char(5);** |  |
| * + **Add** clause     - Allows you to add a new field to the table **Add** *fieldname fieldtype*     - Use same field types as in **Create** command | Add the *Airline* field to FebTrips (T15) |
| * + **Drop** *columnname*     - Removes a field from the table     - If the column is a primary key, you’ll have to remove the primary key constraint (see below) before you’ll be able to drop the column.     - Note: you can drop (alter actually) multiple columns by repeating the action alter table tblName drop column1, drop column2, drop column3; | Delete column Airline |
| * + **Modify** *fieldname* *newtype* [**Not** **Null**]     - Modifies the type of a field     - Optionally, you can also designate the field as not null | Change the type of expenses to Dec 5,2 |
| * + **Change** *colname newcolname coltype*     - Changes the name of a column | Change the name of the employeeId to empId |
| * + Removing a primary key     - Alter Table *tablename* Drop Primary Key     - Removes all keys from a table | Remove the keys from tblFebTrips |
| * + Add a primary key to a table     - Alter Table *tablename*  Add Primary Key (*fieldnames)*       * Parenthesis are required | Restore keys |
| 1. Changing a Table Name    * **Alter** **Table** *oldTableName* **Rename** *newTableName* or  **Rename** **Table** *oldTableName* **To** *newTableName* | Rename tblFebTrips  Rename back to tblFebTrips using To  Note camelCase retained (sort of) |
| 1. Deleting a Table    * **Drop Table** *tablename* | Drop it  Note Rollback doesn’t work for these commands. |
| 1. Complex Changes    * If the changes required for a table are too complex, you might be better off:      + creating a new table with a slightly different name,      + then **Insert**ing the records from the old table into the new table,      + then **Drop**ping the old table      + then **Rename** the table to the original name |  |