Unit 10 – Structured Query Language

Relational database schemas and ER diagrams

Basic CRUD operations

Aggregate Functions

Sub-queries

Joins

1. “ER” – Entity Relationship
2. SQL was originally designed for console.
   1. CRUD operations
3. READ – “SELECT”
   1. SELECT column-names
   2. FROM table-names
   3. WHERE condition
   4. ORDER BY sort-order
   5. “LIKE” – first name begins with “E”
      1. Where name LIKE ‘E%’
         1. % - can be “any” number of chars long
      2. SELECT \* FROM customer WHERE name LIKE ‘E\_\_\_%’
4. CREATE – “INSERT”
   1. When we want to add records to a table, we use INSERT
   2. INSER INTO table-name (column-names)
      1. Kind of like passing in args into parameters
      2. The order of the column names provided should be consistent..
5. UPDATE – “UPDATE”
   1. UPDATE table-name
   2. SET column-name = column-value
   3. WHERE condition
      1. In Update, we definitely need a “where” condition.
6. Delete – “DELETE”
   1. DELETE FROM table-name
   2. WHERE condition
   3. You NEED a “WHERE” clause when you are deleting from the database.
   4. **Or YOU WILL DELETE EVERYTHING IN THAT COLUMN,**
   5. **\*note about conditionals\***
      1. **OR is the word we use to combine conditions.**
7. Aggregate functions
   1. Functions that return a single value from a column.
   2. GROUP BY
      1. This uses one parameter as a filter for the other.
      2. Should generally be used only with aggregate functions.
   3. HAVING
      1. Which has a value of… Something like “WHERE”
      2. Except “WHERE” is evaluated before aggregation – so aggregate functions are passed into “HAVING.”
8. Subqueries
   1. Queries inside of queries
   2. SELECT description
   3. FROM table
   4. WHERE price = (SELECT min(price) FROM item\_order)
      1. This is how you can nest selectors within other selectors.
9. What if we need data from more than one table??
   1. Joins are commands that combine data from two or more tables.
   2. Inner Join
      1. Value of one table matches the value of a specified column in right hand table.
   3. SELECT table1.displayColumn, table2.displayColumn
   4. FROM table 1 INNER JOIN table2 // highlight the values where we have a match, checking ->>
   5. ON table2.matchColumn= table1.matchColumn
10. Left Outer Join
    1. Complete set of records from left hand table
    2. With matching records in the right hand table.
       1. SELECT c.\_id AS customer\_id, c.name, i.\*
       2. FROM customer c LEFT OUTER JOIN item\_order i
       3. ON i.customer\_id = c.\_id
    3. In SQL, we read “FROM” first.
       1. So we give “c” and “i” as tags for “customer” and “item\_order” tables
       2. And we can then use this to select the “\_id” field from “customer”
       3. And we will set this AS “customer\_id”
       4. And we join when customer\_id in item\_order table matches the \_id in customer table.
       5. RIGHT OUTER JOIN is the inverse (preserves all items on the RIGHT side of statement)
    4. Finding what ISN’T there:
       1. Include the above, but we include the clause:
       2. WHERE nickname.columnName = null
11. Using a many-to-many relationship table
    1. SELECT s.name, m.method
    2. FROM shippers AS s
    3. INNER JOIN shipper\_shipping\_method AS sm ON s.\_id = sm.shipper\_id
       1. Trying to match up the shipper’s id and the shipper id stored in SM.
    4. INNER JOIN shipping\_method m ON m.\_id = sm.shipping\_method\_id.
12. SECURITY CONCERNS
    1. SQL Injection Attacks
    2. Tries to inject SQL database with a malicious SQL
       1. Remember that one XKCD strip where the school’s database gets wiped because of a kid’s name?
    3. Sanitize your inputs!
       1. Escape characters that have special meaning
       2. Pattern-checks and validate inputs to see if it conforms to a valid representation of a pattern…