**Assignment 4 calls on you to apply what you’ve learned about RDBMS query performance tuning through Indexes. You are provided a 1-table database composed of the Titanic passenger “manifest” (well, kinda) – one that we have already introduced in class.**

|  |
| --- |
| ASSIGNMENT GOALS:   1. Gain more experience with:    1. Applying Indexes to the appropriate Table fields/attributes.    2. The SQL (DDL) CREATE INDEX statement    3. Reviewing the output of “explain query plan” – with and without Table Indexes |

**Required Resources**

* SQLite Browser: <https://sqlitebrowser.org/dl/>
* Titanic SQLite Database File:
  + BB 🡪 Content 🡪 SQLite Databases 🡪 Titanic DB - single table

**Prep: Textbook Reading Assignment**

* Chap 17 pp 601 ~ 636
* Note: the textbook provides a very different introduction to Relational Table Indexes than was covered in class. It’s a good idea to get exposed to multiple views, but only the material that we covered in class will appear on Quiz 3 and the Final Exam.

**Instructions**

Use the “Titanic” SQLite DBMS attached to Assignment 4 along with this Word document … this database contains one Table called titanic.

Here’s the DDL that might be used to create it.

|  |
| --- |
| CREATE TABLE titanic (  ID INTEGER PRIMARY KEY,  full\_name VARCHAR(255),  age INTEGER,  gender VARCHAR(16),  cabin\_no VARCHAR(255),  survived INTEGER,  no\_parch INTEGER,  no\_sibsp INTEGER,  pass\_class DECIMAL(2,1),  port\_of\_embark VARCHAR(255),  ticket\_cost DECIMAL(6,2),  ticket\_number VARCHAR(255)  ); |

**Assignment**

Given the structure of the “titanic” Table and the way the following valid example queries make use of the data, what indexes would you consider creating, and why?

**QUERIES**

|  |
| --- |
| **-- show min, avg and max of survivors and non survivors**  SELECT  (case when survived = 1 then "Yes" else "No" end) as 'Survived?',  MAX(age) AS MAX\_AGE,  MIN(age) AS MIN\_AGE,  AVG(age) as AVG\_AGE  FROM  titanic  WHERE  age IS NOT NULL  GROUP BY  survived  ;  **-- Number of passengers by age**  SELECT ALL  CAST(age AS FLOAT) as PASSENGER\_AGE,  count(id) as PASS\_COUNT  from  titanic  WHERE age IS NOT NULL  GROUP BY age  ORDER BY  PASSENGER\_AGE DESC  ;  **-- show passenger counts by survived? and age**  SELECT ALL  CAST(age AS FLOAT) as PASSENGER\_AGE,  (case when survived = 1 then "Yes" else "No" end) as 'Survived?',  count(id) as PASS\_COUNT  from  titanic  WHERE age IS NOT NULL  GROUP BY age  ORDER BY  PASSENGER\_AGE, 'Survived?'  ;  **-- all female passengers sorted by cabin class**  SELECT  \*  FROM  titanic  WHERE  gender = 'female'  ORDER BY  pass\_class  ;  **-- Survival count by cabin class**  SELECT ALL  pass\_class as "Cabin Class",  (case when survived = 1 then "Yes" else "No" end) as 'Survived?',  count(id) as PASS\_COUNT  FROM  titanic  GROUP BY  pass\_class  ORDER BY  pass\_class  ;  **-- Pull several fields based on gender and survived?**  SELECT  full\_name,  age,  pass\_class,  ticket\_cost,  ticket\_number,  port\_of\_embark  FROM  titanic  WHERE  gender = 'male' and survived = 1  ORDER BY  full\_name  ;  SELECT  full\_name,  age,  pass\_class,  port\_of\_embark  FROM  titanic  WHERE  gender = 'female' and survived = 1  ORDER BY  full\_name  ; |

(80 points)

So what Indexes would you create against the “titanic” Tables assuming the queries above and similar queries are commonly executed.   
Fill in the following [Word] tables.

**(1)**

|  |  |
| --- | --- |
| Identify Index based  Table and Attribute(s) | **Table: titanic**  **Attributes(s): survived, age** |
| Rationale (Why?) | * **We group by** Survived**, and filter on** age **in the where statement.** * **Indexing survived and age will speed up the grouping filtering operations.** |
| Implement the Index DDL and apply to database – that is, execute the DDL | **CREATE INDEX idx\_survived\_age ON titanic(survived, age);** |
| Justification (or not):  Show results of “Explain Plan” | **explain query plan <query> Confused with this part. Don’t know if we need the sql query or explanation. But I provided both below…  ………………………………………..**  **JUSTIFY:** SELECT  (case when survived = 1 then "Yes" else "No" end) as 'Survived?',  MAX(age) AS MAX\_AGE,  MIN(age) AS MIN\_AGE,  AVG(age) as AVG\_AGE  FROM  titanic  WHERE  age IS NOT NULL  GROUP BY  survived;   Explan plain:   * **Before:** Full table scan (SCAN titanic) with temporary B-Trees for grouping. * **After:** The query uses a **covering index** (idx\_survived\_age), avoiding a full table scan. |

**(2)**

|  |  |
| --- | --- |
| Identify Index based  Table and Attribute(s) | **Table: titanic**  **Attribute(s): age** |
| Rationale (Why?) | * **The query groups by and orders by age. Indexing age will improve performance for both the grouping and sorting operations.** |
| Implement the Index DDL and apply to database – that is, execute the DDL | **CREATE INDEX idx\_age ON titanic(age);** |
| Justification (or not):  Show results of “Explain Plan” | **explain query plan <query>**  **………………………………………. Justify** SELECT ALL  CAST(age AS FLOAT) as PASSENGER\_AGE,  count(id) as PASS\_COUNT  FROM  titanic  WHERE age IS NOT NULL  GROUP BY age  ORDER BY PASSENGER\_AGE DESC;  **Explain Plain**   * **Before: Full table scan with temporary B-Trees for grouping and ordering.** * **After: The query uses an index search (idx\_age\_survived) for filtering and grouping but still uses a temporary B-Tree for sorting.** |

**(3)**

|  |  |
| --- | --- |
| Identify Index based  Table and Attribute(s) | **Table: titanic**  **Attribute(s): age, survived** |
| Rationale (Why?) | * **This query filters by gender and sorts by pass\_class. Indexing both fields will optimize filtering and sorting.** |
| Implement the Index DDL and apply to database – that is, execute the DDL | **CREATE INDEX idx\_gender\_pass\_class ON titanic(gender, pass\_class);** |
| Justification (or not):  Show results of “Explain Plan” | **explain query plan <query>**  **…………………………………………..  JUSTIFY** SELECT ALL  pass\_class as "Cabin Class",  (case when survived = 1 then "Yes" else "No" end) as 'Survived?',  count(id) as PASS\_COUNT  FROM  titanic  GROUP BY  pass\_class  ORDER BY  pass\_class;  **EXPLAIN PLAIN:**   * **Before: Same as Query 2.** * **After: Same improvements as Query 2, utilizing idx\_age\_survived.** |

**(20 Points)**

Review the table schema again. If we were designing a relational database to support the Titanic Voyage before its tragic end, how might you reconstruct this 1 table database into a multi-table database?   
More precisely, **name [at least] 3 entities** that you would include to make this a better data model.

**1**. Passengers

* Attributes:
  + passenger\_id (Primary Key)
  + full\_name
  + age
  + gender
  + port\_of\_embark (Foreign Key referencing Ports table)
* Rationale:  
  This table captures personal details of each passenger. It centralizes information specific to individuals while connecting to other tables for additional attributes.

2. Tickets

* Attributes:
  + ticket\_id (Primary Key)
  + ticket\_number
  + ticket\_cost
  + passenger\_id (Foreign Key referencing Passengers table)
* Rationale:  
  Ticket information can vary, and separating it allows for detailed tracking of ticket prices and associations with passengers. This supports normalization and removes redundancy.

3. Cabins

* Attributes:
  + cabin\_id (Primary Key)
  + cabin\_no
  + pass\_class
* Rationale:  
  Cabins have a class designation, and multiple passengers may share a cabin. Separating cabin information into its own table avoids duplication and facilitates queries based on class or cabin location.

4. Survivals

* Attributes:
  + passenger\_id (Primary Key, Foreign Key referencing Passengers table)
  + survived (Boolean: 1 for survived, 0 for not)
* Rationale:  
  Survival status is specific to a passenger but separate from other attributes. Moving this to its table simplifies survival-related queries.

5. Ports

* Attributes:
  + port\_id (Primary Key)
  + port\_name
* Rationale:  
  Instead of storing port names repeatedly, this table provides a lookup for embarkation ports, improving maintainability and reducing redundancy.

**(+5 extra credit)**

Furthermore, why would the above data model redesign not be worth the effort now in 2024 to benefit our data analysis of this tragedy?

|  |
| --- |
| Redesigning the data model isn’t worth the effort because the Titanic dataset is static, small, and focused on a single historical event. The current flat table structure works perfectly for the kinds of analyses typically done, such as survival rates or demographic breakdowns. Normalizing the data would add unnecessary complexity without offering significant benefits, especially since modern databases handle flat tables efficiently with indexing. Given the dataset’s simplicity and the lack of dynamic growth, the existing structure is more than sufficient for meaningful analysis. |