# Online Book Retailer Database Report

## 1. Data Generation Methodology

- The Online Book Retailer sqlite database was created using Python. It utilized the sqlite3 library for managing the database and the Faker library (with UK locale 'en_GB') for generating realistic, random data.
- **Information Sources:** No external data was downloaded. Faker was used to create all of the strings, including names, emails, countries, and product names. Python's random module was used to create all numerical values (prices, ratings, quantities, and timestamps), making sure that they all fell within reasonable, achievable ranges.
- **Scale:** The database contains three linked tables:
    - **Customers:** 149 rows
    - **Products:** 50 rows
    - **Orders: 1201 rows** (Exceeds the 1000-row minimum requirement.)
- **Realism and Integrity:**
    - **Duplicate Data**: An attempt was made to add a duplicate customer_email. Robust schema design was demonstrated by SQLite's INSERT OR IGNORE clause and UNIQUE constraint, which made sure that only the first instance was retained. In order to mimic actual data entry errors, one duplicate order record was purposefully added to the Orders table.
    - **Missing Data**: To reflect the real-world situation where not every item receives a review, the product rating column was purposefully set to NULL for roughly \$50\%$ of the products.

---

## 2. Database Schema and Data Types

The database employs a three-table normalized schema to manage customer, product, and transaction information.

| Table | Column Name | Data Type | Constraint(s) | Data Type Category |
|---|---|---|---|---|
| Customers | customer_id | INTEGER | PRIMARY KEY | N/A |
| | customer_name | TEXT | NOT NULL | N/A |
| | customer_email | TEXT | UNIQUE, NOT NULL | N/A |
| | customer_country | TEXT | NOT NULL | Nominal (e.g., USA, UK) |
| | customer_signup_date | TEXT | NOT NULL | N/A |
| Products | product_id | INTEGER | PRIMARY KEY | N/A |
| | product_name | TEXT | NOT NULL | N/A |
| | product_sku | TEXT | NOT NULL | N/A |
| | product_category | TEXT | NOT NULL | Nominal (e.g., Fiction, Science) |
| | product_price | REAL | NOT NULL | Ratio (Meaningful zero) |
| | product_rating | INTEGER | NULLABLE | Ordinal (1-5 star order) |

| | | | | |
|---|---|---|---|---|
| | (Composite Key) | | UNIQUE(product name, product sku) | N/A |
| Orders | order_id | INTEGER | PRIMARY KEY | N/A |
| | customer_id | INTEGER | FOREIGN KEY (Customers) | N/A |
| | product_id | INTEGER | FOREIGN KEY (Products) | N/A |
| | order_timestamp | INTEGER | NOT NULL | Interval (UNIX time) |
| | order_quantity | INTEGER | NOT NULL, CHECK(>0) | Ratio (Meaningful zero) |
| | order_total_amount | REAL | NOT NULL | Ratio (Meaningful zero) |
| | order_status | TEXT | NOT NULL | Ordinal (e.g., Pending, Shipped, Delivered) |

## 3. Justification for Schema and Constraints

### *Rationale for Multiple Tables and Schema Links*

1. By separating entity data from transaction data, the three-table structure complies with database normalization principles, particularly 3NF. This method enhances data integrity and reduces redundancy:
2. **Customers** and **Products** store static attribute data.
3. Dynamic transactional data is stored in Orders, which associates a particular transaction record with a particular client and product.
4. **Foreign Keys (Schema Links)**: The Orders table links back to the primary keys in the corresponding parent tables using two foreign keys: the customer and product ids. Referential integrity is thus guaranteed.

    - **`ON DELETE CASCADE`** on the `customer id` foreign key means that if a customer record is deleted, all their associated orders are automatically removed.
    - **`ON DELETE RESTRICT`** on the `product id` foreign key prevents the accidental deletion of a product that has existing order history, safeguarding transactional metrics.

- **Composite Key:** Every name and SKU combination is guaranteed to be unique thanks to the Products table's UNIQUE(product_name, product_sku) constraint. In inventory management, this is essential to avoid mistakes where the same product is listed under different keys.

### *Constraints and Data Integrity*

- **UNIQUE and NOT NULL**: Enforced on customer_email to guarantee that every user is unique, which is crucial for communication and customer support.
- **`CHECK(order_quantity > 0)`:** By preventing the insertion of illogical order data, this constraint guarantees that only legitimate purchase quantities are noted..

---

## 4. Ethical and Data Privacy Discussion

In generating and designing this database, ethical considerations for data handling were prioritized:

- **Data minimization**: Very sensitive Personally Identifiable Information (PII), like complete home addresses or payment information, is not stored in the database. It focuses on the bare minimum of information (name and email) needed to process an order and identify a customer.
- **Anonymization and segregation:** Transaction history analysis can be carried out with an additional degree of privacy by separating the PII (Customers table) from the behavioral data (Orders table), especially if customer IDs are pseudonymized for analytical reporting.
- **Right to Erasure (GDPR):** A possible "right to be forgotten" implementation is directly supported by the use of ON DELETE CASCADE on the customer_id in the Orders table. When a customer record is deleted, the transaction history that goes with it is also deleted, guaranteeing complete data removal when requested.

## Code Appendix:

## The link for the colab is below :

**https://colab.research.google.com/drive/1LEPiLj2hxiNWhyXosfw6LLrEr1CPkq5p#scrollTo=OmegoidIY_s8**