Journal of
Information
Systems
Education

Volume 33
Issue 3
Summer 2022

Teaching Tip
Using SQL to Create and Mine Large, Customizable Datasets

Reagan Siggard, Pamela A. Dupin-Bryant, Robert J. Mills, and David H. Olsen

Recommended Citation: Siggard, R., Dupin-Bryant, P. A., Mills, R. J., & Olsen, D. H. (2022). Teaching Tip: Using SQL to Create and Mine Large, Customizable Datasets. *Journal of Information Systems Education*, 33(3), 209-228.

Article Link: https://jise.org/Volume33/n3/JISE2022v33n3pp209-228.html

Initial Submission:	20 April 2021
Minor Revision:	24 August 2021
Accepted:	3 November 2021
Published:	15 September 2022

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 33(3), 209-228, Summer 2022

Teaching Tip

Using SQL to Create and Mine Large, Customizable Datasets

Reagan Siggard
Department of Data Analytics and Information Systems
Utah State University
Logan, UT 84322, USA
reagan.siggard@usu.edu

Pamela A. Dupin-Bryant
Department of Data Analytics and Information Systems
Utah State University
Tooele, UT 84074, USA
pam.dupin-bryant@usu.edu

Robert J. Mills
Department of Data Analytics and Information Systems
Utah State University
Logan, UT 84322, USA
bob.mills@usu.edu

David H. Olsen
Department of Information Systems and Analytics
Dixie State University
St. George, UT 84770, USA
david.olsen@dixie.edu

ABSTRACT

The SQL-Explore Learning Module detailed in this teaching tip provides an opportunity for students to apply database course knowledge beyond solving traditional pre-determined Structured Query Language (SQL) coding questions. In this unique constructivist activity using the apropos 5E Instructional Model, students explore tables to locate data anomalies, trends, and other key findings in a 100,000-invoice dataset. Detailed instructions and the source code needed to facilitate this innovative learning experience are included. Based on student feedback, 100% of study participants strongly agree or somewhat agree that exploring datasets through the SQL-Explore Activity enhances their knowledge of SQL.

Keywords: Structured query language (SQL), Constructivist learning, 5E instructional model, Data exploration, Data anomalies

1. INTRODUCTION

Business analytics is disrupting the marketplace. To prepare students for future careers, instructors must provide students with experiences in turning data into information. The market for business analytics and data management graduates has exploded in the past five years with no signs of slowing (Glassdoor, 2021; Oostendorp, 2019; Sasso, 2018). Graduates with a strong foundation in Structured Query Language (SQL) are in high demand for careers in business analytics and big data (Hale, 2020). Although students use several tools for data analyses, SQL remains an effective tool for detecting data anomalies (i.e., irregularities or inconsistencies in datasets). In the oft-cited "beer and diapers" example, the original connection was not found using "data mining or other types of advanced analysis. Heath and her team used SQL queries" (Swoyer, 2016, para. 7). By writing SQL queries, a correlation was found between beer and diaper sales. While subsequent studies showed mixed results on validity, the original beer/diaper example highlights the potential benefits of SQL as an exploratory tool. In this case, the example is used as a motivational technique to excite students to dive deeper into the data.

Effectively teaching data analysis includes both new and familiar challenges. Faculty and training practitioners face an evolving need for larger data sets to more effectively implement analytics concepts (Yap & Drye, 2018). A familiar challenge to design business analytics instruction relates to the lack of instructional design background possessed by those responsible for delivering the content. "Technology training is not immune to spray-and-pray training techniques where information is thrown out with the hope some of the content will stick" (Mills et al., 2015, p. 20). Further, academic textbooks on data analysis often focus on solving pre-determined coding questions rather than creating activities to encourage data exploration. There is an apparent need for activities rooted in systematic instructional design that encourage data exploration of large datasets and enhance student learning. This teaching tip provides an innovative solution to this educational need by helping educators (i) create large datasets, (ii) design an explorative data analysis activity, and (iii) update exploratory activities and anomalies.

The SQL-Explore Learning Module outlined in this paper details an opportunity for students to apply their upper-division or graduate database course knowledge to find data anomalies, trends, and critical findings within a dataset. To overcome the challenge of creating larger datasets, the Invoice Dataset SQL-Setup Code (Appendix A) automatically generates random invoices to the desired number of records in Microsoft SQL Server. For this example, the SQL code generates 100,000 invoices, though an instructor implementing this activity can select more (e.g., 1 million invoices) or fewer invoices based on educational preference. The instructor can also update the dataset (through the SQL-Setup Code) differently each iteration to provide their students with an updated, unique dataset. Table 1 highlights the benefits of implementing this SQL exploratory learning module in academic settings.

Educational Challenge	Teaching Module Benefit
Creating large datasets for students to explore.	• Invoice Dataset-SQL Setup Code provided in Appendix A automatically generates as many invoices as the instructor desires. ◦ 100,000 invoices in paper form would stack ~ 33 feet high. ◦ 1,000,000 invoices in paper form would stack ~ 328 feet high.
Limited SQL activities for data exploration.	• Exploratory learning module utilizes a constructivist educational approach that encourages students to explore data to locate data anomalies instead of being limited to solving pre-determined coding questions.
Updating data anomalies each semester.	• Invoice Dataset-SQL Setup Code provided in Appendix A can be easily updated each semester to introduce a wide variety of unique data anomalies.

Table 1. Benefits of Implementing SQL Exploratory Teaching Module

This teaching tip is organized as follows. Section 2 introduces the instructional design considerations used for developing the SQL-Explore Learning Module. Section 3 guides instructors on creating the invoice dataset and embedded anomalies used in the activity. Section 4 illustrates the SQL-Explore Activity used in the study. Section 5 presents student feedback based on SQL coding submissions along with questionnaire results and instructor feedback. Section 6 provides conclusions, recommendations, and lessons learned from this research project.

2. INSTRUCTIONAL DESIGN CONSIDERATIONS

2.1 Selecting an Instructional Design Strategy

Based on prior research, about 50 percent of instructional design practitioners working in industry use specific instructional design theories. In contrast, a higher percentage (86 percent - not mutually exclusive) rely on brainstorming with those involved in the project (Christensen & Osguthorpe, 2004). According to Mayer (2019), there are three phases in conceptualizing learning, instruction, and assessment (behaviorist, cognitivist, and constructivist). Among these, behaviorism learning outcomes are based on response strengthening (e.g., drill and practice), cognitivist on information acquisition (e.g., increasingly complex practice problems), and constructivist on knowledge construction and exploration (e.g., learner developing their inquiries) (Mayer, 2019).

The primary goal of this SQL-Explore Learning Module is for students to explore large invoice datasets to find data anomalies, descriptive statistics, and trends within a dataset to learn how to approach large data sets in their future careers. Given the overriding goal, a constructivist approach to design using the 5E Model of Instruction was selected (Jackson, 2021; Pappas, 2016). A constructivist approach treats the role of the learner as an explorer where the primary instructional strategy is to create puzzlement to afford exploratory opportunities (Christensen, 2008). Treating the learner as an explorer gives the learner a role similar to industry, where they will seek answers to questions through exploring rather than solving pre-determined questions.

2.2 Implementing the 5E Instructional Model

The 5E is a constructivist model widely used in Science, Technology, Engineering, and Math (STEM) education. The philosophies of educational theorists John Dewey, Johann Herbart, Jean Piaget, and Lev Vygotsky provide the foundation for the 5E Instructional Model. The model emphasizes a constructivist approach to learning in which people "construct" knowledge and meaning through their experiences (Bybee, 2009; Bybee et al., 2006).

The phases of the 5E Instructional Model include (E1) *engagement* – engage students in the learning task; (E2) *exploration* – help students explore related ideas and skills; (E3) *explanation* – provide students with a common use of terms, skills, and concepts relative to the learning experience; (E4) *elaboration* – challenge and extend students conceptual understanding and skills; (E5) *evaluation* – provide feedback on the adequacy of student explanations and abilities. Each of the distinct phases is supported by research and has been widely applied in diverse educational settings. Several studies highlight the implementation of the 5E Instructional Model as a successful guide for teaching data analytics and database concepts (Dupin-Bryant & Olsen, 2014; Olsen & Dupin-Bryant, 2016; Piyayodilokchai et al., 2013). Research indicates students who learned SQL via the 5E constructivist model significantly outperformed their peers in understanding SQL concepts and the ability to apply SQL (Piyayodilokchai et al., 2013). Table 2 highlights each phase in the 5E Instructional Model and design considerations implemented in this SQL-Explore Learning Module.

(Phase 1) *Engagement* – Engage students in the learning task.
• The instructor should showcase the importance of the SQL-Explore Activity and the augmentation of the industry practice. Based upon the instructor's experience/background, the method of showcasing the importance will vary. By sharing interesting and relevant examples, students understand the significance of the task and become engaged in the learning process. • Instructors should introduce the 'diapers and beer' study to illustrate the potential benefits of using SQL to locate data anomalies (Step 1 – SQL-Explore Activity). • Before challenging the students to explore data, they should first complete several pre-created queries that help them prepare to analyze the data (Step 2 – SQL-Explore Activity). These queries focus on summary statistics and other data understanding techniques to help guide the student toward more exploratory questions.
(Phase 2) *Exploration* – Help students explore related ideas and skills.
• The main goal of this phase is for students to write exploratory queries to comprehend the dataset and locate data anomalies (Step 2 – SQL-Explore Activity). • This exploration experience provides students with a common base of activities to help them explore the dataset and generate new ideas for query development.
(Phase 3) *Explanation* – Provide students with relevant terms, skills, and concepts.
• Prior knowledge of SQL and accounting will prove helpful in implementing this module. Students will need to know the basic structure of an invoice and related accounts payable and receivable processes. The necessary SQL knowledge is described in section 4.1 and should be familiar to students in an upper-division database management course. • Students' prior knowledge of accounting and SQL will be essential throughout the activity. Knowledge can be reinforced using additional materials in textbooks, slide sets, and the course learning management system.
(Phase 4) *Elaboration* – Challenge and extend students' conceptual understanding and skills.
• After completing the exploratory queries, the instructor could add enhanced activities for students to encourage them to explore the data with different technologies, including Tableau, Python, Excel, PowerPivot, or PowerBI, to better "tell the story" of the invoice dataset. • The instructor should provide an opportunity for students to verbalize and summarize their findings from their experience exploring the invoice dataset (Step 3 and Step 4 – SQL-Explore Activity).
(Phase 5) *Evaluation* – Provide feedback on the adequacy of student explanation and abilities.
• The instructor should ask students to share their findings and conclusions (Step 4 – SQL-Explore Activity). • After students share their discoveries, the instructor can reveal what anomalies are purposefully built into the dataset to test their knowledge. To enhance learning SQL advanced coding techniques, the instructor may choose to share the complete Invoice Dataset-Setup Code (Appendix A) with students. • Students should take time to reflect and discuss what other resources would help complete this activity.

Table 2. 5E Instructional Model Summary and Constructivist Design Considerations

3. DESIGNING THE SQL - EXPLORE ACTIVITY

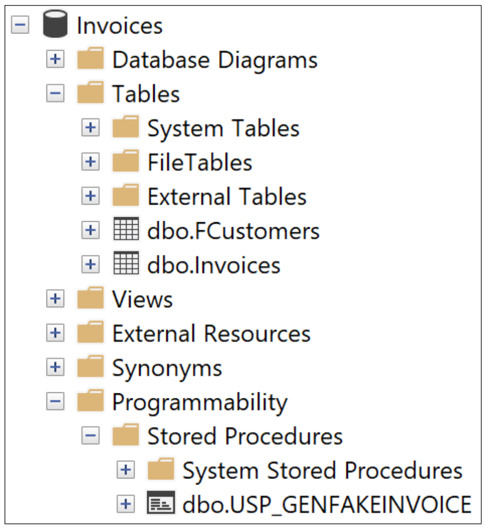
3.1 Preparing the SQL-Explore Activity

This section provides step-by-step instructions instructors can use to build a 100,000-invoice dataset. The instructor begins by setting up a generic, read-only account for all students to access (e.g., Username: invoice and Password: invoice) in Microsoft SQL Server. Before the SQL-Explore Activity, the instructor should run the Invoice Dataset-SQL Setup Code (Appendix A) to create the dataset.

This code runs as one fluid program yet contains five unique sections. The following will highlight the purpose of each code section. Comments are also provided in the code to identify the section's purpose. Section 1 of the code includes preparation activities: (i) kills the process using the invoice database if it already exists, (ii) drops the database if it already exists, and (iii) creates and sets up a new database.

3.2 Creating the Tables and Stored Procedure

Section 2 of the code (i) generates the initial invoices table, (ii) creates and populates the customer table, and (iii) creates a stored procedure (USP_GENFAKEINVOICE) to generate the 100,000 random invoices. In the code, USE [Invoices] and ALTER DATABASE [Invoices] direct the code to the database (Invoices) used in the activity. Figure 1 illustrates the file structure created for the SQL-Explore Activity, along with an example invoice given to students as a visual representation of invoice structure and corresponding field names.



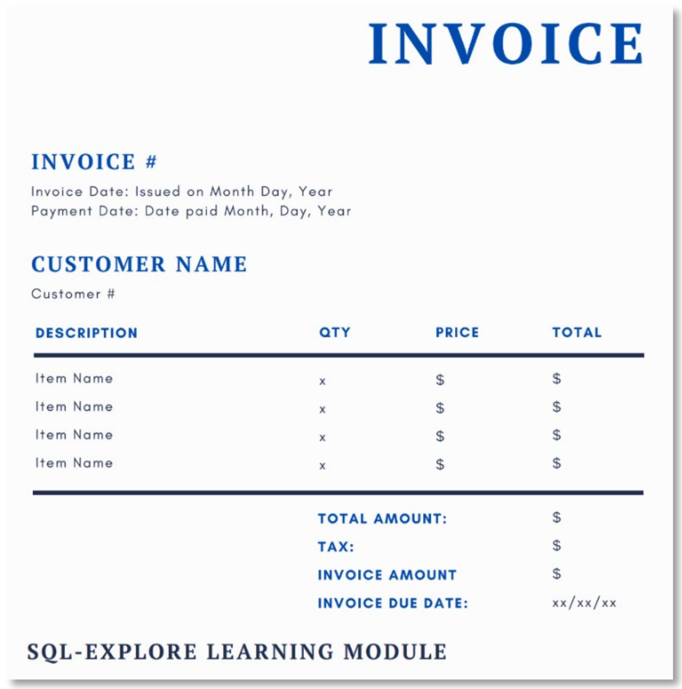


Figure 1. Tables, Stored Procedures, and Example Invoice Created for SQL-Explore Activity

3.3 Define Invoice Parameters and Execute the Stored Procedure

Once the invoice table, customer table, and stored procedure exist, the procedure USP_GENFAKEINVOICE is executed (Section 3 of the code) to create the 100,000 invoices. The instructor can use the default parameters or modify them based on educational preferences (EXEC USP_GENFAKEINVOICE 100000, '2000/01/01', '2019/12/31',60,2000,1000,100;). See specific stored procedure parameters in Table 3.

Stored Procedure Parameters	Purpose
1st Parameter (100000)	# of records in the database created by the stored procedure
2nd Parameter ('2000/01/01')	Start date for the invoice
3rd Parameter ('2019/12/31')	End date for the invoice
4th Parameter (60)	# of days between the invoice date and invoice due date
5th Parameter (200)	Upper bound of the invoice
6th Parameter (1000)	Lower bound of the invoice
7th Parameter (100)	# of customers built with the routine

Table 3. Stored Procedure Parameters

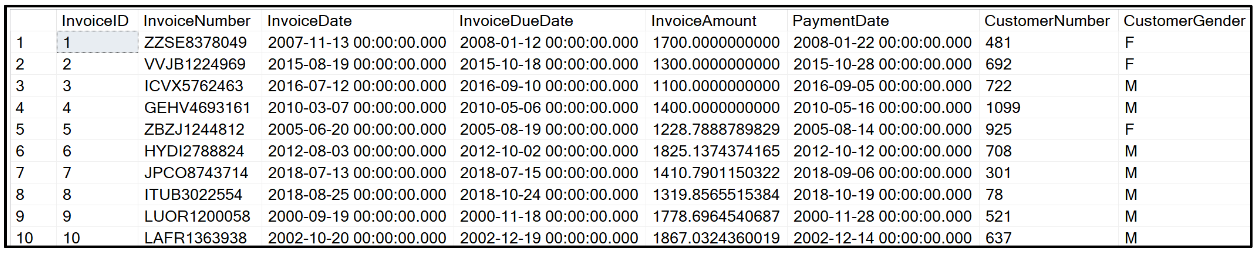
3.4 Add SQL Code to Introduce Variability into the Dataset

For this study, data anomalies are any data point that deviates from a randomized, normally distributed set of data. The 100,000 invoices are randomly generated based on the parameters specified in the USP_GENFAKEINVOICE stored procedure. This final step in preparing the invoices for the SQL-Explore Activity is to purposely introduce variability and potential anomalies into the data. Section 4 and Section 5 of the code illustrate how variability is added. Guidance for students can be specific or vague based on the instructor's objectives for the activity. Table 4 provides a list of embedded variability included in this activity, each of which the instructor can easily change when implementing the code. See specific SQL code to create these embedded anomalies in Appendix A – Section 5.

Once the stored procedure executes, the following output will generate for the invoice table (Figure 2) and the customer table (Figure 3).

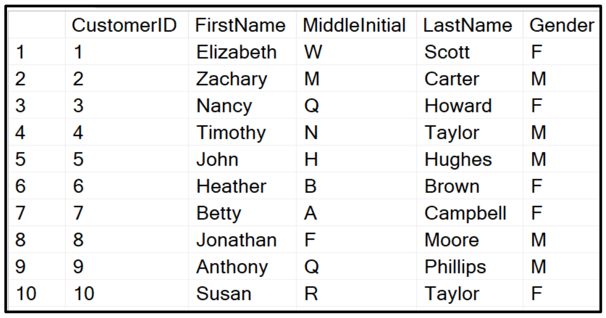
Embedded Variability in Invoice Dataset-SQL Setup Code
Four invoice amounts are an even hundred-dollar amount.
Multiple invoices for a single customer occur within one week.
Some invoice due dates and invoice (creation) dates are less than 60 days apart.
Some invoice dates are approximately 50 years older than the others.
One invoice is due two days after purchase.
Females out-purchase males.
Incorrect invoice numbers (included characters).
Some invoices included large purchases (i.e., $50,000).
One invoice amount is equal to a dollar.
One invoice included a payment before the purchase date.

Table 4. Examples of Embedded Anomalies



	InvoiceID	InvoiceNumber	InvoiceDate	InvoiceDueDate	InvoiceAmount	PaymentDate	CustomerNumber	CustomerGender
1	1	ZZSE8378049	2007-11-13 00:00:00.000	2008-01-12 00:00:00.000	1700.0000000000	2008-01-22 00:00:00.000	481	F
2	2	VVJB1224969	2015-08-19 00:00:00.000	2015-10-18 00:00:00.000	1300.0000000000	2015-10-28 00:00:00.000	692	F
3	3	ICVX5762463	2016-07-12 00:00:00.000	2016-09-10 00:00:00.000	1100.0000000000	2016-09-05 00:00:00.000	722	M
4	4	GEHV4693161	2010-03-07 00:00:00.000	2010-05-06 00:00:00.000	1400.0000000000	2010-05-16 00:00:00.000	1099	M
5	5	ZBZJ1244812	2005-06-20 00:00:00.000	2005-08-19 00:00:00.000	1228.7888789829	2005-08-14 00:00:00.000	925	F
6	6	HYDI2788824	2012-08-03 00:00:00.000	2012-10-02 00:00:00.000	1825.1374374165	2012-10-12 00:00:00.000	708	M
7	7	JPCO8743714	2018-07-13 00:00:00.000	2018-07-15 00:00:00.000	1410.7901150322	2018-09-06 00:00:00.000	301	M
8	8	ITUB3022554	2018-08-25 00:00:00.000	2018-10-24 00:00:00.000	1319.8565515384	2018-10-19 00:00:00.000	78	M
9	9	LUOR1200058	2000-09-19 00:00:00.000	2000-11-18 00:00:00.000	1778.6964540687	2000-11-28 00:00:00.000	521	M
10	10	LAFR1363938	2002-10-20 00:00:00.000	2002-12-19 00:00:00.000	1867.0324360019	2002-12-14 00:00:00.000	637	M

Figure 2. SQL Relation Output of the 100,000 Invoice Rows



	CustomerID	FirstName	MiddleInitial	LastName	Gender
1	1	Elizabeth	W	Scott	F
2	2	Zachary	M	Carter	M
3	3	Nancy	Q	Howard	F
4	4	Timothy	N	Taylor	M
5	5	John	H	Hughes	M
6	6	Heather	B	Brown	F
7	7	Betty	A	Campbell	F
8	8	Jonathan	F	Moore	M
9	9	Anthony	Q	Phillips	M
10	10	Susan	R	Taylor	F

Figure 3. SQL Relation Output of the 100 Customer Rows

4. SQL-EXPLORE ACTIVITY

4.1 Classroom Activity – Setup

This SQL-Explore Activity is for an upper-division undergraduate or graduate database management course. The activity is recommended for students who understand basic accounting principles (invoice structure and accounts payable) and SQL principles, including SQL clauses, aggregate functions, case expressions, and multi-table querying techniques. Students should also have previous exposure to Common Table Expressions (CTEs), derived tables, and Window functions. Instructors can present this activity in various delivery modes (online, broadcast, hybrid, or in-person classroom delivery). Students should work independently or in small, self-selected groups (2 to 4 students per group).

4.2 Classroom Activity – Instructions

At the beginning of the SQL-Explore Learning Module, the instructor introduces the tables as part of a company's invoice tracking data. The SQL-Explore Activity includes five components: (i) read and summarize the "beer and diapers" article, (ii) explore the data by deriving coding questions and solutions, (iii) summarize findings, (iv) participate in a follow-up questionnaire, and (v) submit work and present activity findings. Appendix B outlines the student activity (adapted from an online format) that is amendable to individual course needs.

Step 2 is the heart of the activity. This step helps orient the student to the dataset, provides sample coding questions/solutions, and asks the student to explore the dataset by writing queries to identify trends, abnormal data distribution, or some systematic difference. This step potentially includes establishing rules that would hypothetically warrant further investigation. See a list of potential rules in Appendix B – Step 2.

5. FEEDBACK AND EVALUATION

The SQL-Explore Activity was recently implemented as an extra credit assignment in an online, upper-division database management course to provide context for the module's effectiveness. Data collection was approved by the respective Institutional Review Board (IRB) and found to be purely educational. The data collected includes a nine-item questionnaire (Appendix C) used to measure perceived benefits and challenges to the SQL-Explore Activity along with assignment submissions. Assessment data collected from the assignment submissions include student-derived coding questions, solutions, and summaries based on data exploration. Thirty of forty-five students in the advanced database course began the extra credit SQL-Explore Activity (Step 1 – summarizing the "Beer and Diaper" article). Twenty-two of those thirty students completed the entire SQL-Explore Activity (Steps 1-5).

5.1 Student Feedback

Eight of the nine-item questionnaire (Appendix C) are based on a five-point Likert scale with one (1) as strongly disagree and five (5) as strongly agree. Find the results of the questionnaire in Table 5.

Questionnaire Question	Mean	SD
1. Exploring datasets (i.e., invoice activity) enhances my knowledge of SQL	4.8	.39
2. Exploring datasets (i.e., invoice activity) will help improve my value at a future job.	4.7	.57
3. Exploring datasets (i.e., invoice activity) is more difficult than solving pre-determined coding questions.	4.1	.87
4. Exploring datasets (i.e., invoice activity) is more important than solving pre-determined coding questions.	4.0	.84
5. After completing this activity, I want to spend more time learning how to explore datasets.	4.8	.43
6. After completing this activity, I want to spend more time learning other methods for exploring data.	4.7	.43
7. After completing this activity, I am more confident about my SQL coding abilities.	4.0	.95
8. After completing this activity, I am more excited to continue learning SQL.	4.6	.59

Table 5. Student Questionnaire Response Summary (n = 22)

5.2 Coding Examples and Evaluation

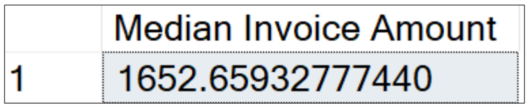
On the exploratory part of the activity, students submitted an average of 8.45 SQL exploratory queries. A database expert ranked the submissions as complex, moderate, or simple to determine the complexity of the student's SQL queries. Among the 186 query submissions, 30 percent ranked complex, 42 percent moderate, and 28 percent were simple. Appendix D highlights several student coding examples.

Students generally followed one of three approaches to complete the exploratory activity. The first approach included students responding to specific conditions given in the assignment prompt (Appendix B – Step 2 provides a list of pre-determined conditions). For instance, invoice amounts need to be greater than $1,000. A second approach focused on trends (e.g., worst customer in terms of payment history), and students proceeded to write queries to discover the origin of the trend. A third approach included students developing hypothetical company invoice standards and writing queries to identify invoices that violated the standards. Table 6 provides examples of student-developed queries, query outputs, and descriptions that demonstrate each of these three approaches.

Approach Category 1 – *Responding to Specific Conditions in Assignment Prompt*

Student Example 1 – Description: What is the median invoice amount?

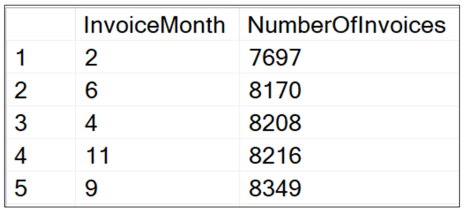
```
SELECT AVG(1.0 * InvoiceAmount) AS 'Median Invoice
Amount'
FROM (
    SELECT InvoiceAmount
    FROM Invoices
    ORDER BY InvoiceAmount
    OFFSET (@c - 1) / 2 ROWS
    FETCH NEXT 1 + (1 - @c % 2) ROWS ONLY
) AS x;
```



	Median Invoice Amount
1	1652.65932777440

Student Example 2 – Description: Number of Male/Female Customers: (Primary Customer = Female).

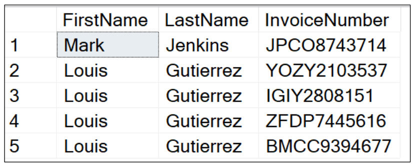
```
SELECT COUNT(CustomerGender) as 'Gender of Customers'
FROM Invoices
GROUP BY CustomerGender;
```



	InvoiceMonth	NumberOfInvoices
1	2	7697
2	6	8170
3	4	8208
4	11	8216
5	9	8349

Student Example 3 – Description: Invoices due 60 days from the invoice date (all else considered problematic).

```
SELECT FirstName, LastName, i.InvoiceNumber
FROM FCustomers AS FC JOIN Invoices as I
    ON fc.CustomerID = i.CustomerNumber
WHERE DATEDIFF(day, i.InvoiceDate,
i.InvoiceDueDate)<>60;
```

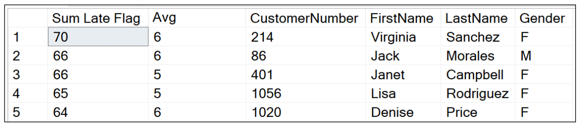


	FirstName	LastName	InvoiceNumber
1	Mark	Jenkins	JPCO8743714
2	Louis	Gutierrez	YOZY2103537
3	Louis	Gutierrez	IGIY2808151
4	Louis	Gutierrez	ZFDP7445616
5	Louis	Gutierrez	BMCC9394677

Approach Category 2 – *Trend Identification*

Student Example 1 – Description: Which customers have the worst payment history (a tendency of being late), and on average, how late are they on payments?

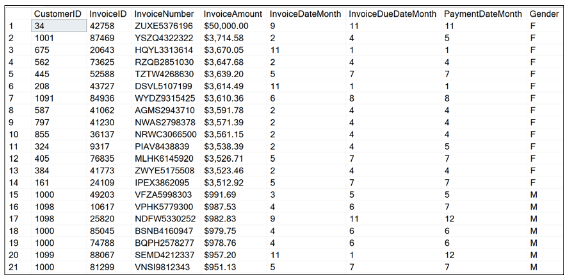
```
WITH LatePmtFlag
 AS
 (SELECT CustomerNumber, CustomerGender,
InvoiceAmount, InvoiceDueDate,
PaymentDate,
 CASE
 WHEN PaymentDate > InvoiceDueDate
 THEN 1
 ELSE 0
 END AS 'Late Flag',
 CASE
 WHEN PaymentDate > InvoiceDueDate
 THEN DATEDIFF(Day, InvoiceDueDate, PaymentDate)
 ELSE 0
 END AS 'Days Late'
 FROM Invoices)
 SELECT TOP 50 SUM(i.[Late Flag]) AS 'Sum Late Flag',
AVG(i.[Days Late]) AS 'Avg
Days Late', i.CustomerNumber, c.FirstName, c.LastName,
c.Gender
 FROM LatePmtFlag AS i LEFT JOIN FCustomers AS c
 ON i.CustomerNumber = c.CustomerID
 GROUP BY i.CustomerNumber, c.FirstName, c.LastName,
c.Gender
 ORDER BY [Sum Late Flag] DESC, [Avg Days Late] DESC;
```



	Sum Late Flag	Avg	CustomerNumber	FirstName	LastName	Gender
1	70	6	214	Virginia	Sanchez	F
2	66	6	86	Jack	Morales	M
3	66	5	401	Janet	Campbell	F
4	65	5	1056	Lisa	Rodriguez	F
5	64	6	1020	Denise	Price	F

Student Example 2 – Description: When evaluating the customer IDs associated with the invoices, it becomes apparent almost all of the problematic invoices for the men are caused by two individuals (1000 and 1098). This could indicate some problem with these two customers or just coincidence. Regardless, there is reason to look into the activity of these two customers.

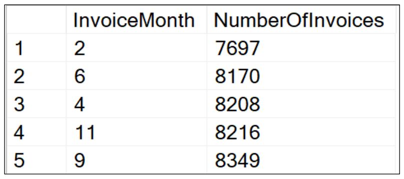
```
SELECT FC.CustomerID, I.InvoiceID, I.InvoiceNumber,
FORMAT(I.InvoiceAmount,'C') AS 'InvoiceAmount',
DATEPART(MONTH,I.InvoiceDate) AS 'InvoiceDateMonth',
DATEPART(MONTH,I.InvoiceDueDate) AS
'InvoiceDueDateMonth',
DATEPART(MONTH,I.PaymentDate) AS
'PaymentDateMonth', FC.Gender
FROM INVOICES AS i JOIN FCUSTOMERS AS fc
ON i.CustomerNumber = fc.CustomerID
WHERE I.InvoiceAmount > 3500 OR I.InvoiceAmount <1000
ORDER BY I.InvoiceAmount DESC;
```



	CustomerID	InvoiceID	InvoiceNumber	InvoiceAmount	InvoiceDateMonth	InvoiceDueDateMonth	PaymentDateMonth	Gender
1	34	42758	ZUXE5376196	$50,000.00	9	11	11	F
2	1001	87469	YSZQ4322322	$3,714.58	2	4	5	F
3	675	20643	HQYL3313614	$3,670.05	11	1	1	F
4	562	73625	RZQB2851030	$3,647.68	2	4	4	F
5	445	52588	TZTW4268630	$3,639.20	5	7	7	F
6	208	43727	DSVL5107199	$3,614.49	11	1	1	F
7	1091	84936	WYDZ9315425	$3,610.36	6	8	8	F
8	587	41062	AGMS2943710	$3,591.78	2	4	4	F
9	797	41230	NWAS2798378	$3,571.39	2	4	4	F
10	855	36137	NRWC3066500	$3,561.15	2	4	4	F
11	324	9317	PIAV8438839	$3,538.39	2	4	5	F
12	405	76835	MLHK6145920	$3,526.71	5	7	7	F
13	384	41773	ZWYE5175508	$3,523.46	2	4	4	F
14	161	24109	IPEX3862095	$3,512.92	5	7	7	F
15	1000	49203	VFZA5998303	$991.69	3	5	5	M
16	1098	10617	VPHK5779300	$987.53	4	6	7	M
17	1098	25820	NDFW5330252	$982.83	9	11	12	M
18	1000	85045	BSNB4160947	$979.75	4	6	6	M
19	1000	74788	BQPH2578277	$978.76	4	6	6	M
20	1099	88067	SEMD4212337	$957.20	11	1	12	M
21	1000	81299	VNSI9812343	$951.13	5	7	7	M

Student Example 3 – Description: The most interesting thing is that February is the slowest month for invoices. Far less are processed during that month than any other, but the majority of problematic invoices are during that month. As a general trend, women spend far more than men, and they are far more likely to spend over $3,500 in February. It can be postulated that women are our best customers in this case.

```
SELECT DATEPART(MONTH,InvoiceDate) AS
'InvoiceMonth', COUNT(InvoiceID) AS 'NumberOfInvoices'
FROM INVOICES
GROUP BY DATEPART(MONTH,InvoiceDate)
ORDER BY NumberOfInvoices ASC;
```

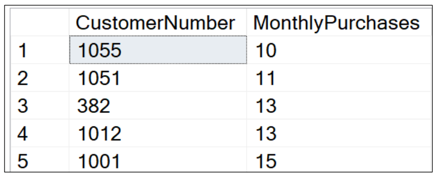


	InvoiceMonth	NumberOfInvoices
1	2	7697
2	6	8170
3	4	8208
4	11	8216
5	9	8349

Approach Category 3 – *Developing Hypothetical Company Standards*

Student Example 1 – Description: Customers that have under 30 monthly purchases with invoices less than $1,500 are considered standard customers. How many standard customers are in this table?

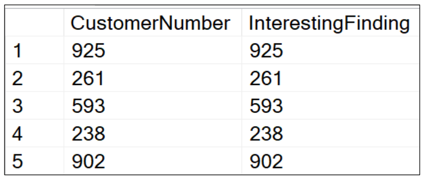
```
SELECT CustomerNumber, COUNT(CustomerNumber) AS
[MonthlyPurchases]
FROM Invoices
WHERE InvoiceAmount < 1500
GROUP BY CustomerNumber
HAVING COUNT(CustomerNumber) <= 30
ORDER BY [MonthlyPurchases];
```



	CustomerNumber	MonthlyPurchases
1	1055	10
2	1051	11
3	382	13
4	1012	13
5	1001	15

Student Example 2 – Description: The CustomerNumber is equivalent to the sum of the CustomerNumber divided by the count of the CustomerNumber.

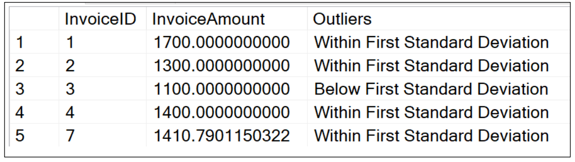
```
SELECT CustomerNumber,
SUM(CustomerNumber)/COUNT(CustomerNumber) AS
[InterestingFinding]
FROM Invoices
GROUP BY CustomerNumber;
```



	CustomerNumber	InterestingFinding
1	925	925
2	261	261
3	593	593
4	238	238
5	902	902

Student Example 3 – Description: There is only one outlier below the second standard deviation, and it was a purchase for one dollar. This is very different from the more than a dozen that are above the second standard deviation. This implies a lop-sided bell curve in the InvoiceAmounts table.

```
WITH cte_Invoice
AS
(SELECT AVG(InvoiceAmount) as Average,
STDEV(InvoiceAmount) as StdDeviation,
AVG(InvoiceAmount)-STDEV(InvoiceAmount) AS
First_Deviation_below,
AVG(InvoiceAmount)-STDEV(InvoiceAmount)-
STDEV(InvoiceAmount) AS Second_Deviation_below,
```



	InvoiceID	InvoiceAmount	Outliers
1	1	1700.0000000000	Within First Standard Deviation
2	2	1300.0000000000	Within First Standard Deviation
3	3	1100.0000000000	Below First Standard Deviation
4	4	1400.0000000000	Within First Standard Deviation
5	7	1410.7901150322	Within First Standard Deviation

```
AVG(InvoiceAmount)+STDEV(InvoiceAmount)              AS
First_Deviation_above,
AVG(InvoiceAmount)+STDEV(InvoiceAmount)+STDEV(Inv
oiceAmount) AS Second_Deviation_above
FROM Invoices)
SELECT InvoiceID, InvoiceAmount,
CASE
    When  InvoiceAmount  >  Second_Deviation_above  THEN
'Above Second Standard Deviation - High Outlier'
    WHEN  InvoiceAmount  >  First_Deviation_above  THEN
'Above First Standard Deviation'
    When  InvoiceAmount  <  Second_Deviation_below  THEN
'Below Second Standard Deviation - Low Outlier'
    WHEN  InvoiceAmount  <  First_Deviation_below  THEN
'Below First Standard Deviation'
    Else 'Within First Standard Deviation'
    END AS Outliers
FROM cte_Invoice, Invoices
WHERE DATEDIFF(DAY, InvoiceDate, InvoiceDueDatE) <>
60
OR
InvoiceAmount < 1000
OR
InvoiceAmount > 3500
OR
InvoiceAmount%1 = InvoiceAmount
OR
InvoiceAmount = ROUND(InvoiceAmount, 0);
```

Table 6. Student Coding Examples

5.3 Instructor Feedback

Based on the initial implementation of the SQL-Explore activity, future instructional design considerations to improve activity are possible. For instance, outstanding coding examples from prior students may serve as a form of teacher expectancy and create a Pygmalion effect where high-quality examples result in high-quality query submissions. Further, a follow-up activity providing feedback is recommended. Ideally, after receiving quality feedback, students would participate in an additional exploratory activity in what Merrill (2002) refers to as encountering a progression of problems, something recommended for effective design.

6. CONCLUSIONS

The SQL-Explore Learning Module detailed in this teaching tip provides a unique opportunity for students to interact with large numbers of invoices to identify summary statistics, trends, and anomalies. Exploring datasets in SQL is a special type of learning and requires a different mindset than more traditional assignments where students solve pre-determined coding problems. Combining conventional and exploratory activities into the curriculum helps students develop a well-rounded understanding of SQL coding practices. Instructors, students, and employers understand that the future is rooted in analyzing data and acting correctly on the findings.

Based on student feedback from the SQL-Explore Activity, results suggest instructional design considerations in database courses should move beyond traditional SQL teaching methods of solving pre-determined coding questions. Instructors should seek to incorporate constructivist learning activities where students apply their knowledge in an explorative fashion. This exploratory approach provides a more robust understanding of SQL and prepares students for future employment opportunities and applications. Students in this study believe activities that require exploration of datasets enhance their knowledge of SQL and improve their value in a future job. In addition, students indicated they wanted to spend more time learning how to explore datasets after completing the SQL-Explore Activity.

The overwhelmingly positive feedback related to the SQL-Explore Activity was based, in part, on improvements made during initial pilot studies. The activity presented in this teaching tip represents a third-generation approach based on two different pilot attempts. In the first iteration, students began exploring immediately without prior readings or coding examples. This approach did not provide enough context for students and largely neglected the engagement and explanation phases of the 5E Instructional Model. In the second iteration, the instructor provided 20 pre-determined SQL queries, which overwhelmed the students and resulted in fewer exploratory queries. As such, the current SQL-Explore Activity included appropriate context (i.e., initial reading activity), provided a limited number of pre-determined coding questions, and identified possible areas to investigate. These improvements greatly enhanced the exploratory SQL coding experience.

One of the highlights of this teaching tip is the innovative, custom-built Invoice Dataset SQL-Setup Code provided to help instructors create countless customizable invoices and introduce unique variability into datasets. Customization of invoices and inserted anomalies allows the instructor to determine the focus of the exploratory activity. For instance, the activity could focus on using analytics instead of locating potential trends and anomalies. If technically feasible, the instructor may generate larger datasets for students to explore (e.g., a million invoices).

In this SQL-Explore Activity, a single database is created for all students with generic credentials. While students do not individually run the Invoice Dataset-SQL Setup Code, the instructor may choose to share the entire code with students or focus on the code that introduces variability into the dataset after the activity is complete. If server space is adequate, the instructor may ask each student to run the setup code to create individual databases to explore. In sharing the code that creates the database and introduces variability into the dataset, students may recognize how easily arbitrary code and data anomalies are embedded in datasets. Instructors should emphasize how easily datasets can be manipulated and how individuals must always be vigilant when working with large datasets.

This SQL-Explore Learning Module is for students taking an advanced database/SQL course. As such, students were familiar with more advanced coding strategies. Future research is warranted to identify possible exploratory activities targeted for introductory classes. Further, given the complexity of student code specified in the study results, exploring how coding solutions were derived would also merit further study. For instance, were coding solutions primarily adapted from online examples, based on samples found in the course textbook (Ben-Gan's T-SQL Fundamentals), or derived in another way?

As analytics and information systems instructors look for innovative learning activities to help prepare students for future employment opportunities, this SQL-Explore Learning Module provides a valuable experience for students to practice exploratory coding strategies, including analytics, statistics, locating anomalies, and finding potential fraud. This teaching tip offers the necessary elements to introduce data exploration into any database course effortlessly. The SQL-Explore Activity helps strengthen student understanding of SQL beyond solving pre-determined coding questions and helps students better understand any stories the data are telling.

7. REFERENCES

Bybee, R. W. (2009). The BSCS 5E Instructional Model and 21st Century Skills. http://sites.nationalacademies.org/cs/groups/dbassesite/documents/webpage/dbasse_073327.pdf

Bybee, R., Taylor, J. A., Gardner, A., Van Scotter, P., Carlson, J., Westbrook, A., & Landes, N. (2006). *The BSCS 5E Instructional Model: Origins and Effectiveness.* Colorado Springs, CO: BSCS.

Christensen, T. K., & Osguthorpe, R. (2004). How Do Instructional-Design Practitioners Make Instructional-Strategy Decision? *Performance Improvement Quarterly*, 17(3), 45-65.

Christensen, T. K. (2008). The Role of Theory in Instructional Design: Some Views of an ID Practitioner. *Performance Improvement*, 47(4), 25-32.

Dupin-Bryant, P. A., & Olsen, D. H. (2014). Business Intelligence, Analytics and Data Visualization: A Heat Map Project Tutorial. *International Journal of Management & Information Systems*, 18(3), 185-200.

Glassdoor. (2021). 50 Best Jobs in America for 2021. https://www.glassdoor.com/List/Best-Jobs-in-America-LST_KQ0,20.htm

Hale, J. (2020). Most In-Demand Tech Skills for Data Analysts. https://towardsdatascience.com/most-in-demand-tech-skills-for-data-analysts-26d4ea4450f8

Jackson, D. K. (2021). Steps Toward Selecting Instructional Strategies That Promote Academic Achievement. https://academic.csuohio.edu/jackson_d/Dossier/Scholarship/Publications/Engaged_Learning/EnagedLearningJournalArticleDraft.pdf

Mayer, R. E. (2019). Thirty Years of Research on Online Learning. *Applied Cognitive Psychology*, 33(2), 152-159.

Merrill, M. D. (2002). First Principles of Instruction. *Educational Technology Research and Development*, 50(3), 1042-1629.

Mills, R. J., Dupin-Bryant, P. A., & Olsen, D. H. (2015). Designing SQL Database Modules Using Correlation Coefficients and Linear Regression: A Content-Centered Approach. *Performance Improvement*, 54(6), 20-31.

Olsen, D. H. & Dupin-Bryant, P. A. (2016). Integrating Data Cleansing With Popular Culture: A Novel SQL Character Data Tutorial. *Review of Business Information Systems*, 20(1), 13-28.

Oostendorp, N. (2019). Radical Change Is Coming to Data Science Jobs. https://www.forbes.com/sites/forbestechcouncil/2019/03/01/radical-change-is-coming-to-data-science-jobs/?sh=6acc8223dfcc

Pappas, C. (2016). 8 Tips to Choose the Best Instructional Design Model for Your Next eLearning Course. https://elearningindustry.com/tips-choose-best-instructional-design-model-elearning-course

Piyayodilokchai, H., Panjaburee, P., Laosinchai, P., Ketpichainarong, W., & Ruenwongsa, P. (2013). A 5E Learning Cycle Approach–Based, Multimedia-Supplemented Instructional Unit for Structured Query Language. *Educational Technology & Society*, 16(4), 146-159.

Sasso, M. (2018). Economics: This is America's Hottest Job. https://www.bloomberg.com/news/articles/2018-05-18/-sexiest-job-ignites-talent-wars-as-demand-for-data-geeks-soars

Swoyer, S. (2016). Beer and Diapers: The Impossible Correlation, Transforming Data With Intelligence. https://tdwi.org/articles/2016/11/15/beer-and-diapers-impossible-correlation.aspx

Yap, A. Y., & Drye, S. L. (2018). The Challenges of Teaching Business Analytics: Finding Real Big Data for Business Students. *Information Systems Education Journal*, 16(1), 41-50.

AUTHOR BIOGRAPHIES

Reagan Siggard is an instructor in the data analytics & information systems in the Jon M. Huntsman School of Business at Utah State University. Her instruction areas include python programming, data analytics, and database management. Reagan enjoys modifying course materials based on industry demand and deploying courses through online, broadcast, and in-person methods. Reagan Siggard is currently working on her doctoral program applications in hopes of pursuing a career in instructional design and learning analytics.

Pam Dupin-Bryant is a professor of data analytics and information systems in the Jon M. Huntsman School of Business at Utah State University. She earned her Ph.D. at the University of Wyoming and her master's and bachelor's degrees at USU. Throughout her career, Dr. Dupin-Bryant has employed a wide variety of delivery methods and educational strategies to facilitate learning. Her primary teaching activities include business applications programming, web design/development, and data/information for business. Her research and scholarly writings focus primarily on information systems pedagogy and online/distance education. Pam Dupin-Bryant has received many awards for her teaching innovations, research, and service.

Robert J. Mills is a professor of data analytics & information systems in the Jon M. Huntsman School of Business at Utah State University. His research interests include computer-based learning environments, knowledge transfer, and MIS education. Bob Mills has consulted on technology-based training projects for a variety of organizations, including Silicon Graphics International (SGI), EnergySolutions Arena / Utah Jazz, International Center for Captive Insurance Education (ICCIE), and IBM. In addition, Mills designs and develops MIS and database textbook supplements (Pearson Publishing).

David Olsen received his Ph.D. in management information systems from the University of Arizona in 1993 and taught at the University of Akron accounting department in accounting information systems for five years. Dr. Olsen joined the MIS department at Utah State University in 1998 and served as the MIS department head from Fall 2012 to Summer 2019. He is currently on the faculty at Dixie State University and teaches primarily in data analytics using SQL, Python, Tableau, and machine learning. His research has been published in journals such as *Communications of the ACM*, *Issues in Accounting Education*, and the *Journal of Database Management*.

APPENDICES

Appendix A. Invoice Dataset – SQL Setup Code

```
/*
SECTION 1
Kill the process using the Invoice database if it exists. Drop the database if it exists.
Create and set up the database.
*/

USE master;
GO
DECLARE @DatabaseName nvarchar(50)
SET @DatabaseName = N'Invoices'
DECLARE @SQL VARCHAR(MAX)
SELECT @SQL = COALESCE(@SQL,'') + 'Kill ' + Convert(varchar, SPId) + ';'
FROM MASTER..SysProcesses
WHERE DBId = DB_ID(@DatabaseName) AND SPId <> @@SPId

--SELECT @SQL--
EXEC(@SQL)
GO
IF DB_ID (N'Invoices') IS NOT NULL
DROP DATABASE Invoices;
GO
CREATE DATABASE Invoices;
GO

--Verify the database files and sizes--
SELECT name, size, size*1.0/128 AS [Size in MBs]
FROM sys.master_files
WHERE name = N'Invoices';
GO

/*
SECTION 2
Create the stored procedure that builds and populates the customers and invoices table.
*/

ALTER DATABASE [Invoices]
SET COMPATIBILITY_LEVEL = 130
GO
USE INVOICES
GO
CREATE PROCEDURE USP_GENFAKEINVOICE
@N INT,
@StartDate DATETIME,
@EndDate DATETIME,
@DueDiff INT,
@InvAmtUpperLimit Decimal(23, 10),
@InvAmtLowerLimit Decimal(23, 10),
@CN INT
AS

--Create Invoices table--
IF OBJECT_ID('dbo.FInvoices') IS NOT NULL
DROP TABLE dbo.FInvoices;
CREATE TABLE dbo.FInvoices
(
[InvoiceID] INT IDENTITY(1,1) NOT NULL,
[Invoice Code] CHAR(11) NOT NULL,
[Invoice Date] DATETIME NOT NULL,
[Due Date] DATETIME NOT NULL,
[Amount] DECIMAL(23,10) NOT NULL,
[Pay Date] DATETIME NOT NULL,
[CustomerID] INT NOT NULL
);

--Create Customers table--
IF OBJECT_ID('dbo.FCustomers') IS NOT NULL
DROP TABLE dbo.FCustomers;
CREATE TABLE dbo.FCustomers
(
[CustomerID] INT IDENTITY(1,1) NOT NULL,
[FirstName] CHAR(12) NOT NULL,
[MiddleInitial] CHAR(2) NOT NULL,
[LastName] CHAR(12) NOT NULL,
[Gender] CHAR(1) NOT NULL
);

--Generate fake customers--
DECLARE @CCurrentCount INT
SET @CCurrentCount = 0
WHILE @CCurrentCount < @CN
BEGIN
    DECLARE @Gender INT
    SET @Gender = (FLOOR(RAND()*(1-0+1))+1)
    INSERT INTO dbo.FCustomers
    SELECT
        CASE @Gender
            WHEN 1 THEN (SELECT TOP 1 value
                FROM
STRING_SPLIT('James,John,Robert,Michael,William,David,Richard,Joseph,Thomas,Charles,Christopher,Daniel,Matthew,A
nthony,Donald,Mark,Paul,Steven,Andrew,Kenneth,George,Joshua,Kevin,Brian,Edward,Ronald,Timothy,Jason,Jeffrey,Ryan,J
acob,Gary,Nicholas,Eric,Stephen,Jonathan,Larry,Justin,Scott,Brandon,Frank,Benjamin,Gregory,Raymond,Samuel,Patrick,Ale
xander,Jack,Dennis,Jerry,Tyler,Aaron,Henry,Jose,Douglas,Peter,Adam,Nathan,Zachary,Walter,Kyle,Harold,Carl,Jeremy,Ger
ald,Keith,Roger,Arthur,Terry,Lawrence,Sean,Christian,Ethan,Austin,Joe,Albert,Jesse,Willie,Billy,Bryan,Bruce,Noah,Jordan,
Dylan,Ralph,Roy,Alan,Wayne,Eugene,Juan,Gabriel,Louis,Russell,Randy,Vincent,Philip,Logan,Bobby,Harry,Johnny', ',')
                ORDER BY NEWID())
            ELSE (SELECT TOP 1 value
                FROM
STRING_SPLIT('Mary,Patricia,Jennifer,Linda,Elizabeth,Barbara,Susan,Jessica,Sarah,Margaret,Karen,Nancy,Lisa,Betty,Doro
thy,Sandra,Ashley,Kimberly,Donna,Emily,Carol,Michelle,Amanda,Melissa,Deborah,Stephanie,Rebecca,Laura,Helen,Sharon,
Cynthia,Kathleen,Amy,Shirley,Angela,Anna,Ruth,Brenda,Pamela,Nicole,Katherine,Samantha,Christine,Catherine,Virginia,D
ebra,Rachel,Janet,Emma,Carolyn,Maria,Heather,Diane,Julie,Joyce,Evelyn,Joan,Victoria,Kelly,Christina,Lauren,Frances,Mart
ha,Judith,Cheryl,Megan,Andrea,Olivia,Ann,Jean,Alice,Jacqueline,Hannah,Doris,Kathryn,Gloria,Teresa,Sara,Janice,Marie,Juli
a,Grace,Judy,Theresa,Madison,Beverly,Denise,Marilyn,Amber,Danielle,Rose,Brittany,Diana,Abigail,Natalie,Jane,Lori,Alexis
,Tiffany,Kayla', ',')
                ORDER BY NEWID())
            END AS [FirstName],
        CHAR(FLOOR(RAND()*(90-65+1))+65) AS [MiddleInitial],
        (SELECT TOP 1 value
                FROM
STRING_SPLIT('Smith,Johnson,Williams,Brown,Jones,Miller,Davis,Garcia,Rodriguez,Wilson,Martinez,Anderson,Taylor,Th
omas,Hernandez,Moore,Martin,Jackson,Thompson,White,Lopez,Lee,Gonzalez,Harris,Clark,Lewis,Robinson,Walker,Perez,H
all,Young,Allen,Sanchez,Wright,King,Scott,Green,Baker,Adams,Nelson,Hill,Ramirez,Campbell,Mitchell,Roberts,Carter,Phill
ips,Evans,Turner,Torres,Parker,Collins,Edwards,Stewart,Flores,Morris,Nguyen,Murphy,Rivera,Cook,Rogers,Morgan,Peterso
n,Cooper,Reed,Bailey,Bell,Gomez,Kelly,Howard,Ward,Cox,Diaz,Richardson,Wood,Watson,Brooks,Bennett,Gray,James,Rey
es,Cruz,Hughes,Price,Myers,Long,Foster,Sanders,Ross,Morales,Powell,Sullivan,Russell,Ortiz,Jenkins,Gutierrez,Perry,Butler,
Barnes,Fisher', ',')
                ORDER BY NEWID()),
        CASE @Gender
            WHEN 1 THEN 'M'
        ELSE 'F'
        END AS [Gender]
SET @CCurrentCount = @CCurrentCount + 1
END
```

```
--Generate fake invoices--
DECLARE @CurrentCount INT
SET @CurrentCount = 0
WHILE @CurrentCount < @N
BEGIN
     DECLARE @RandDate DATETIME
     SET @RandDate = CAST(FLOOR(RAND()*(DATEDIFF(DD, 1, @EndDate)-DATEDIFF(DD, 1,
@StartDate)+1))+DATEDIFF(DD, 1, @StartDate) AS DATETIME)
     INSERT INTO dbo.FInvoices
     SELECT CONCAT(CHAR(FLOOR(RAND()*(90-65+1))+65),
          CHAR(FLOOR(RAND()*(90-65+1))+65),
          CHAR(FLOOR(RAND()*(90-65+1))+65),
          CHAR(FLOOR(RAND()*(90-65+1))+65),
          FORMAT(FLOOR(RAND()*(9999999-1000000+1))+1000000, '0000000'))
          AS [Invoice Code],
          CONVERT(DATE, @RandDate) AS [Invoice Date],
          CONVERT(DATE, DATEADD(DD,@DueDiff,@RandDate)) AS [Due Date],
          RAND()*(@InvAmtUpperLimit-@InvAmtLowerLimit+1)+@InvAmtLowerLimit AS [Amount],
          CASE (FLOOR(RAND()*(1-0+1))+1)
               WHEN 1 THEN CONVERT(DATE, DATEADD(DD,-5,DATEADD(DD,@DueDiff,@RandDate)))
               ELSE CONVERT(DATE, DATEADD(DD,10,DATEADD(DD,@DueDiff,@RandDate)))
          END AS [Pay Date],
          (SELECT TOP 1 CustomerID
               FROM dbo.FCustomers
               ORDER BY NEWID()) AS [CustomerID]
SET @CurrentCount = @CurrentCount + 1
END
GO

/*
SECTION 3
Execute the stored procedure that creates and populates the invoices and customers tables.
*/

EXEC USP_GENFAKEINVOICE 100000, '2000/01/01', '2019/12/31',60,2000,1000,1100;
GO

/*
SECTION 4
Clean up the column names to be consistent and informative. Populate the CustomerGender column in the Invoices table.
*/

EXEC sp_rename 'FInvoices', 'Invoices'
GO
sp_rename 'Invoices.InvoiceID', 'InvoiceID', 'COLUMN';
GO
sp_rename 'Invoices.Invoice Code', 'InvoiceNumber', 'COLUMN';
GO
sp_rename 'Invoices.Invoice Date', 'InvoiceDate', 'COLUMN';
GO
sp_rename 'Invoices.Due Date', 'InvoiceDueDate', 'COLUMN';
GO
sp_rename 'Invoices.Amount', 'InvoiceAmount', 'COLUMN';
GO
sp_rename 'Invoices.Pay Date', 'PaymentDate', 'COLUMN';
GO
sp_rename 'Invoices.CustomerID', 'CustomerNumber', 'COLUMN';
GO
GO
ALTER TABLE Invoices
  ADD CustomerGender CHAR(2);
GO
UPDATE Invoices
```

```
    SET CustomerGender = fc.Gender
    FROM Invoices AS I JOIN FCustomers AS fc ON
    (i.CustomerNumber = fc.CustomerID)

/*
SECTION 5
Sample anomalies embedded in the data tables. Note: this will differ based upon instructor preference and the random
generation within the set-up code.
*/

--Create an invoice due two days after purchase--
UPDATE Invoices
SET InvoiceDueDate = '20180715'
WHERE InvoiceID = 7;
--Increase percentage of female purchases--
UPDATE Invoices
SET CustomerGENDER = 'F'
WHERE InvoiceID BETWEEN 40000 and 50000;
--Create incorrect invoicenumber--
UPDATE Invoices
SET InvoiceNumber = 'Hi1'
WHERE InvoiceID = 42757;
--Amend an invoice amount to be $50,000--
UPDATE Invoices
SET InvoiceAmount = 50000
WHERE InvoiceID = 42758;
--Amend an invoice amount to be $1-
UPDATE Invoices
SET InvoiceAmount = 1
WHERE InvoiceID = 99758;
--Create payment date before purchase--
UPDATE Invoices
SET PaymentDate = '20080101'
WHERE InvoiceID = 60147;
--Create four invoice amounts ending in an even hundred-dollar amount--
  WITH InvoiceAnomaly AS
(
SELECT TOP 4 InvoiceID, ROUND(InvoiceAmount,-2) AS InvoiceAmount,InvoiceNumber,
          ROW_NUMBER() OVER (ORDER BY InvoiceNumber DESC) AS RN
    FROM Invoices
)
UPDATE Invoices
SET InvoiceAmount = IA.InvoiceAmount
    FROM InvoiceAnomaly AS IA
    WHERE Invoices.InvoiceNumber = IA.InvoiceNumber;

--Push invoice dates back ~54 years for 4 random invoices--
WITH InvoiceAnomaly AS
(
SELECT TOP 4 InvoiceID + 68 AS InvoiceID, InvoiceDate - 20000 AS InvoiceDate,
            InvoiceDueDate - 20000 AS InvoiceDueDate,
            PaymentDate - 20000 AS PaymentDate,
ROW_NUMBER() OVER (ORDER BY InvoiceAmount DESC) AS RN
    FROM Invoices
)
UPDATE Invoices
SET InvoiceDate = IA.InvoiceDate,
    InvoiceDueDate = IA.InvoiceDueDate,
    PaymentDate = IA.PaymentDate
    FROM InvoiceAnomaly AS IA
      WHERE Invoices.InvoiceID = IA.InvoiceID;
GO

--Create multiple invoices for a single customer occur within a week--
```

```
USE INVOICES
GO
DBCC FREESYSTEMCACHE ('Temporary Tables & Table Variables')
GO
DECLARE @randomDate DATETIME
DECLARE @fromDate DATETIME='2019-06-16'
DECLARE @count INT
DECLARE @localVariable1 DATETIME;
DECLARE @Invoices TABLE (InvoiceID INT,
                         CustomerNumber INT,
                         InvoiceNumber CHAR(11),
                         InvoiceDate DATETIME,
                         RowNumbers INT);
SELECT @randomDate= (DATEADD(day, ROUND(DATEDIFF(day, @fromDate, @fromDate)
* RAND(CHECKSUM(NEWID())), 5),DATEADD(second, abs(CHECKSUM(NEWID())) % 886400,
@fromDate)));
WITH Oneweek AS
(
SELECT TOP 1 InvoiceID, CustomerNumber, InvoiceNumber, InvoiceDate,
    ROW_NUMBER() OVER (ORDER BY InvoiceNumber DESC) AS RN
    FROM Invoices
)
INSERT INTO @Invoices
SELECT  InvoiceID, CustomerNumber, InvoiceNumber, InvoiceDate, RN
    FROM Oneweek;
WITH CustomerAnomaly AS
(
SELECT TOP 4 InvoiceNumber, CustomerNumber, InvoiceDate
    FROM Invoices
    WHERE CustomerNumber IN
    (SELECT CustomerNumber
        FROM @Invoices)
)
UPDATE Invoices
SET InvoiceDate = @randomdate + (CHECKSUM(NEWID()) % 6)
    FROM CustomerAnomaly AS a
    WHERE Invoices.InvoiceNumber = a.InvoiceNumber
GO

--Change Invoice due date to 1-6 days from the Invoice Date--
UPDATE Invoices
SET InvoiceDueDate = InvoiceDate + ABS(CHECKSUM(NEWID()) % 6) + 1
WHERE InvoiceID % 20000 = 0
```

Appendix B. SQL-Explore Activity

Introduction: As you are aware, most SQL instruction includes demonstration of SQL principles followed by practice opportunities where you solve a specific question. In this activity, you will spend much of your time exploring 100,000 invoices. Think about it, 100,000 invoices in paper form stacked on your desk would be about 10 meters high (i.e., ~33 feet high). However, in SQL, you are able to scan through the data in just seconds to learn more about the invoices.

This activity includes four steps:

- **Step 1** has you read a short article about 'beer and diapers.' [Time to complete ~20 minutes]
- **Step 2** is the main exploration part of the activity and includes guiding instructions for completing this activity. [Time to complete ~90 minutes]
- **Step 3** has you succinctly summarize your findings. [Time to complete ~20 minutes]
- **Step 4** is a brief 9 question survey. [Time to complete ~10 minutes]
- **Step 5** has you submit your work and present activity findings. [Time to complete ~10 minutes]

Step 1. Read about "Beer and Diapers"

- Read the following article titled "Beer and Diapers: The Impossible Correlation" https://tdwi.org/articles/2016/11/15/beer-and-diapers-impossible-correlation.aspx
- Provide a brief overview of the article in 3-4 sentences.

Step 2. Explore the Invoices and Derive New Coding Questions and Solutions

Read through entire assignment before beginning -- Be Creative!!!

- Login to the Invoice Dataset on our course SQL Server. Username: invoice Password: invoice
- This dataset contains 100,000 invoices.
- This database contains two tables: Invoices and FCustomers

Try SELECT * FROM Invoices to make sure it is working! Invoices is the main table you will be exploring.

Explore the Invoices and derive SQL questions and coding solutions that summarize data, and attempt to locate anomalies, trends, or unique findings in the Invoice Table. The Customer Table is available if you want to better describe your findings by connecting names or other information with a customer ID. Below are some potential questions, comments, and suggestions to explore (please keep in mind you will likely focus on a few of the possible starting points):

What do we know about the Invoice Table?

1. Any invoices ending in a whole dollar amount are considered problematic.
2. All invoices are due 60 days from the invoice date. Anything else is considered problematic.
3. Invoices are supposed to be above $1,000 and below $3,500. Anything else is considered problematic.
4. The CIO is interested in learning more about the Invoice Tables. He had some questions which include:
   a. Is our primary customer male or female?
   b. Who are our best customers? Ones that spend a lot? Ones that purchase a lot?
   c. What months produce the most invoices?
5. Derive basic statistics about the invoices – means, median, mode, standard deviation. Use Google to help locate simple formulas if needed. Find something that stands out whether by average, by standard deviation or some systematic difference.
   - Come up with rules that need to be investigated (be creative). For instance: Customers that made 10 purchases over 2,500 within a month may be considered troublesome and warrant further investigation.

```
SELECT CustomerNumber, COUNT(CustomerNumber) AS [PurchasedMoreThan10]
FROM Invoices
WHERE InvoiceAmount > 2500
GROUP BY CustomerNumber
HAVING COUNT(CustomerNumber) > 10
ORDER BY PurchasedMoreThan10 DESC;
```

6. Use DATE FUNCTIONS (i.e., DATEDIFF), aggregate functions (MIN, MAX, AVG..), GROUP BY, HAVING, or whatever you believe would help you better understand the 'story of the' invoices.

Step 3. Summarize Findings

- Succinctly summarize your findings from step 2. Organize findings in a professional format. Include your exploratory questions, coding solutions, screen captures, and a summary paragraph detailing your findings.

Step 4. Questionnaire (APPENDIX C)

After you have completed the module on 100,000 invoice exploration, please complete the questionnaire.

Step 5. Submit Your Work

- Submit your brief overview of the 'beer and diapers' article (~3-4 sentences).
- Submit your exploratory coding questions, coding solutions, and screen captures. If you focus on simpler exploration, you will have more submissions than if you decide to focus on more complex formulas (from Step 3).
- Submit your summary paragraph. The summary paragraph should detail your findings of the invoices. Make sure to include any summary statistics, key findings, trends, or anomalies you discover (from Step 3).
- Submit the student questionnaire (from Step 4).

Appendix C. Questionnaire

Instructions: Please rate each question from 1 to 5 as follows:

1 = Strongly Disagree ● 2 = Somewhat Disagree ● 3 = Neutral ● 4 = Somewhat Agree ● 5 = Strongly Agree

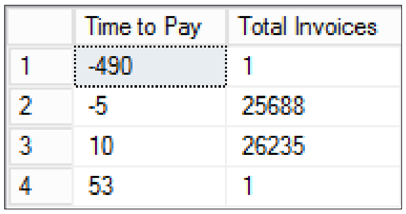
1. Exploring datasets (i.e., invoice activity) enhances my knowledge of SQL.
2. Exploring datasets (i.e., invoice activity) will help improve my value at a future job.
3. Exploring datasets (i.e., invoice activity) is more difficult than solving pre-determined coding questions.
4. Exploring datasets (i.e., invoice activity) is more important than solving pre-determined coding questions.
5. After completing this activity, I want to spend more time learning how to explore datasets.
6. After completing this activity, I want to spend more time learning other methods for exploring data.
7. After completing this activity, I am more confident about my SQL coding abilities.
8. After completing this activity, I am more excited to continue learning SQL.
9. Open-Ended Question: What would you recommend to make this explorative activity more effective?

Appendix D. Student Coding Examples

Student Coding Examples

Student Example 1 – Description: Compare InvoiceAmount with the PaymentDate and determine if lower purchase amounts cause more customers to pay their invoices on time.

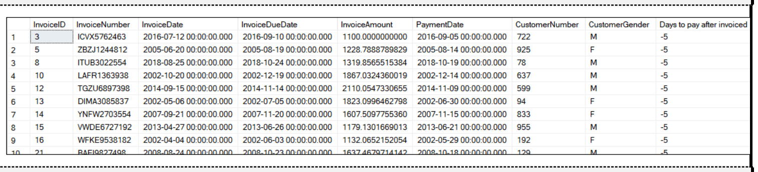
```sql
SELECT DATEDIFF(day, InvoiceDueDate,
PaymentDate) as 'Time to Pay',
COUNT(InvoiceID) as 'Total Invoices'
FROM Invoices
WHERE InvoiceAmount <= (SELECT
    AVG(InvoiceAmount)
        FROM Invoices)
GROUP BY DATEDIFF(day, InvoiceDueDate,
PaymentDate)
ORDER BY DATEDIFF(day, InvoiceDueDate,
PaymentDate);
```



	Time to Pay	Total Invoices
1	-490	1
2	-5	25688
3	10	26235
4	53	1

Student Example 2 – Description: Discrepancy in payment: The payment is due before the invoice date.

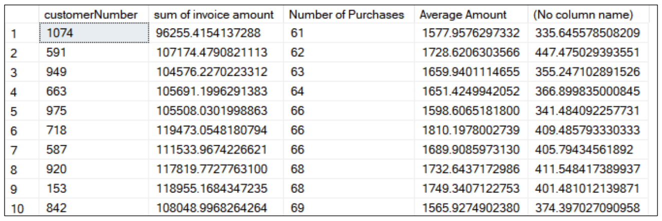
```sql
SELECT *, DATEDIFF(Day, InvoiceDueDate,
PaymentDate) AS 'Days to pay after invoiced'
FROM Invoices
WHERE SIGN(DATEDIFF(Day,
InvoiceDueDate, PaymentDate)) < 1;
```



	InvoiceID	InvoiceNumber	InvoiceDate	InvoiceDueDate	InvoiceAmount	PaymentDate	CustomerNumber	CustomerGender	Days to pay after invoiced
1	3	ICVX5762463	2016-07-12 00:00:00.000	2016-09-10 00:00:00.000	1100.0000000000	2016-09-05 00:00:00.000	722	M	-5
2	5	ZBZJ1244812	2005-06-20 00:00:00.000	2005-08-19 00:00:00.000	1228.7888789829	2005-08-14 00:00:00.000	925	F	-5
3	8	ITUB3022554	2018-08-25 00:00:00.000	2018-10-24 00:00:00.000	1319.8565515384	2018-10-19 00:00:00.000	78	M	-5
4	10	LAFR1363938	2002-10-20 00:00:00.000	2002-12-19 00:00:00.000	1867.0324360019	2002-12-14 00:00:00.000	637	M	-5
5	12	TGZU6897398	2014-09-15 00:00:00.000	2014-11-14 00:00:00.000	2110.0547330655	2014-11-09 00:00:00.000	599	M	-5
6	13	DIMA3085837	2002-05-06 00:00:00.000	2002-07-05 00:00:00.000	1823.0996462798	2002-06-30 00:00:00.000	94	F	-5
7	14	YNFW2703554	2007-09-21 00:00:00.000	2007-11-20 00:00:00.000	1607.5097755360	2007-11-15 00:00:00.000	833	F	-5
8	15	VWDE6727192	2013-04-27 00:00:00.000	2013-06-26 00:00:00.000	1179.1301669013	2013-06-21 00:00:00.000	955	M	-5
9	16	WFKE9538182	2002-04-04 00:00:00.000	2002-06-03 00:00:00.000	1132.0652152054	2002-05-29 00:00:00.000	192	F	-5
10	21	BAFI9827498	2008-08-24 00:00:00.000	2008-10-23 00:00:00.000	1637.4679714142	2008-10-18 00:00:00.000	129	M	-5

Student Example 3 – Description: Customers that constantly seem to get our lowest pricing.

```sql
SELECT CustomerNumber,
SUM(InvoiceAmount) AS 'sum of invoice
amount', COUNT(*) AS 'Number of Purchases',
AVG(InvoiceAmount) as 'Average Amount',
STDEV(InvoiceAmount)
FROM Invoices
GROUP BY CustomerNumber
HAVING AVG(InvoiceAmount) <
AVG(InvoiceAmount) - STDEV(InvoiceAmount)
ORDER BY 'Number of Purchases';
```



	customerNumber	sum of invoice amount	Number of Purchases	Average Amount	(No column name)
1	1074	96255.4154137288	61	1577.9576297332	335.645578508209
2	591	107174.4790821113	62	1728.6206303566	447.475029393551
3	949	104576.2270223312	63	1659.9401114655	355.247102891526
4	663	105691.1996291383	64	1651.4249942052	366.899835000845
5	975	105508.0301998863	66	1598.6065181800	341.484092257731
6	718	119473.0548180794	66	1810.1978002739	409.485793330333
7	587	111533.9674226621	66	1689.9085973130	405.79434561892
8	920	117819.7727763100	68	1732.6437172986	411.548417389937
9	153	118955.1684347235	68	1749.3407122753	401.481012139871
10	842	108048.9968264264	69	1565.9274902380	374.397027090958

Information Systems & Computing Academic Professionals

Education Special Interest Group

ISSN 2574-3872