Foundations of SQL

Tara Cool

Boston University Metropolitan College

CS 669: Database Design & Implementation for Business

Professor Warren Mansur

Group 1 (Pamela Farr, Facilitator)

May 16, 2023

Section 1: Absolute Fundamentals

Step 1: Creating a Table

Create the Cars table so that it contains the following columns:

Cars Table

CarBrand: VARCHAR(64)	CarModel: VARCHAR(50)	AcquiredDate: DATE	Price: DECIMAL(10,2)
Ford	Econoline Full-Size Van	15-AUG-2021	29,995.00

```
CREATE TABLE Cars(
CarBrand      VARCHAR(65),
CarModel      VARCHAR(50),
AcquiredDate  DATE,
Price         DECIMAL(10,2)
);
```

100 %

Messages

```
Commands completed successfully.
```

Step 2: Inserting a Row

Insert a row where the Car Brand name is "Ford", the Car Model is "Econoline Full-Size Van", the acquisition date for the car is August 15,2021, and the price is $29,995.00.

```
INSERT INTO Cars (CarBrand, CarModel, AcquiredDate, Price)
VALUES ('Ford', 'Econoline Full-Size Van', '2021-8-15', 29995);
```

100 %

Messages

```
(1 row affected)

Completion time: 2023-05-12T15:07:43.8059390-04:00
```

Step 3: Selecting All Rows

Select all rows in the table to review the inserted data.

```
SELECT *
FROM Cars;
```

100 %

Results Messages

	CarBrand	CarModel	AcquiredDate	Price
1	Ford	Econoline Full-Size Van	2021-08-15	29995.00

Step 4: Updating All Rows

Update the price of the row in the "Cars" table to $28,000, then select all rows to confirm the update.

```
UPDATE Cars
SET Price = 28000;

SELECT *
FROM Cars;
```

100 %

Results Messages

	CarBrand	CarModel	AcquiredDate	Price
1	Ford	Econoline Full-Size Van	2021-08-15	28000.00

Step 5: Deleting All Rows

Remove all rows from the "Cars" table & verify that no rows remain.

```
DELETE FROM Cars;

SELECT *
FROM Cars;
```

100 %

Results Messages

CarBrand	CarModel	AcquiredDate	Price

Step 6: Dropping a Table

Drop the "Cars" table, then select all rows in the table to verify the table doesn't exist. We will also explain the use of the error messages for diagnosis.

```
DROP TABLE Cars;

SELECT *
FROM Cars;
```

100 %

Messages

```
Msg 208, Level 16, State 1, Line 38
Invalid object name 'Cars'.
```

The error message implies the 'Cars' table (i.e., "object") does not exist (i.e., it is "invalid"), when trying to reference the entity with SELECT statement. Thus, verifying the table was successfully dropped and does not exist in the database. Ultimately, the error message might be used in conjunction with the SELECT command to diagnose the error, as the message suggests an error occurred in a command used in Line 38. The location of the error (e.g., given as a line number) is especially helpful when we are executing more than one SQL command simultaneously, as we would know the error starts in the line where the SELECT command was used.

Section 2: More Precise Data Handling

Step 7: Table Setup

Create the "Apartments" table with appropriate columns, datatypes, and constraints.

```
CREATE TABLE Apartments (
ApartmentNum  DECIMAL     PRIMARY KEY,
ApartmentName VARCHAR(64) NOT NULL,
Description   VARCHAR(64) NULL,
CleanedDate   DATE        NOT NULL,
AvailableDate DATE        NOT NULL
);
```

100 %

Messages

```
Commands completed successfully.
```

Step 8: Table Population

Populate the table with multiple rows and then select all rows from the Apartments table to show that the inserts were successful. Note that the description for Apartment 316 at Paradise Palms is null.

```
INSERT INTO Apartments (ApartmentNum,
    ApartmentName,
    Description,
    CleanedDate,
    AvailableDate)
VALUES (498,
    'Deer Creek Crossing',
    'Great view of Riverwalk',
    CAST('19-APR-2022' AS DATE),
    CAST('25-APR-2022' AS DATE)
    );

INSERT INTO Apartments (ApartmentNum,
    ApartmentName,
    Description,
    CleanedDate,
    AvailableDate)
VALUES (128,
    'Town Place Apartments',
    'Convenient walk to Parking',
    CAST('20-MAY-2022' AS DATE),
    CAST('25-MAY-2022' AS DATE)
    );

INSERT INTO Apartments (ApartmentNum,
    ApartmentName,
    Description,
    CleanedDate,
    AvailableDate)
VALUES (316,
    'Paradise Palms',
    NULL,
    CAST('02-JUN-2021' AS DATE),
    CAST('08-JUN-2021' AS DATE)
    );

SELECT *
FROM Apartments;
```

100 %

Results Messages

	ApartmentNum	ApartmentName	Description	CleanedDate	AvailableDate
1	128	Town Place Apartments	Convenient walk to Parking	2022-05-20	2022-05-25
2	316	Paradise Palms	NULL	2021-06-02	2021-06-08
3	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25

Step 9: Invalid Insertion

The following values leave the Apartment Name with no value.

ApartmentNum = 252

ApartmentName = NULL

Description = Close to Downtown shops

CleanedDate = 17-JUL-2020

AvailableDate = 13-JUL-2020

Attempt an invalid insertion, explain null values, and the "NOT NULL" constraint.

Null values are missing values, such as values are not known/might be entered or realized at a later point in time and those values corresponding to optional fields/columns in the database table. This is in contrast to a zero or blank row value, (i.e., a space, which is interpreted as a character & given as ' ', rather than 'NULL').

The "NOT NULL" constraint is an integrity rule that ensures a given field/column value is not optional/conditional. In other words, every row must contain a value for fields with the "NOT NULL" constraint. For example, when the Apartments table was created, a "NOT NULL" constraint was included in the ApartmentName, CleanedDate, and AvailableDate columns.

```
INSERT INTO Apartments (ApartmentNum,
    ApartmentName,
    Description,
    CleanedDate,
    AvailableDate)
VALUES (252,
    NULL,
    'Close to downtown shops',
    CAST('17-JUL-2020' AS DATE),
    CAST('13-JUL-2020' AS DATE)
    );
```

100 %

Messages

```
Msg 515, Level 16, State 2, Line 96
Cannot insert the value NULL into column 'ApartmentName', table 'CS669.dbo.Apartments'; column does not allow nulls. INSERT fails.
The statement has been terminated.

Completion time: 2023-05-13T16:44:51.3634123-04:00
```

The error message suggests an error in a command starting on Line 96 – specifically stating that the "INSERT" command was unsuccessful. This is because NULL values cannot be inserted into the "ApartmentName" column in the "Apartments" table of the "dbo" default schema in the "CS669" database; as the column does not permit NULL values (i.e., the NOT NULL constraint is applied as a column constraint).

Step 10: Valid Insertion

Now insert the row with the Apartment Name intact, with the following values.

ApartmentNum = 252

ApartmentName = The Glenn

Description = Close to Downtown shops

CleanedDate = 17-JUL-2020

AvailableDate = 13-JUL-2020

```
INSERT INTO Apartments (ApartmentNum,
    ApartmentName,
    Description,
    CleanedDate,
    AvailableDate)
VALUES (252,
    'The Glenn',
    'Close to downtown shops',
    CAST('17-JUL-2020' AS DATE),
    CAST('13-JUL-2020' AS DATE)
    );
```

100 %

Messages

```
(1 row affected)
```

Step 11: Filtered Results

Retrieve specific data from the "Apartments" table and discuss the importance of limiting rows and columns.

```
SELECT ApartmentName, Description
FROM Apartments
WHERE ApartmentNum = 498;
```

100 %

Results Messages

	ApartmentName	Description
1	Deer Creek Crossing	Great view of Riverwalk

Limiting the number of rows and columns returned from a SELECT statement with the "WHERE" command is especially useful when dealing with a large volume of data – where a table might contain millions of rows of data. However, even with a small volume of data, the ability to limit the number of rows enables us to drill down the dataset and filter the output in a way only the necessary information is displayed – maximizing efficiency in terms of the time/effort required by the database the generate the values for every row/column and the time/effort required by the end-user to mine through the data and obtain the appropriate insights.

Step 12: Updating a Column

Update an apartment's description & select all rows to confirm the update.

```
UPDATE Apartments
SET description = 'A mile walk to the beach'
WHERE ApartmentNum = 316;

SELECT *
FROM Apartments;
```

100 %

Results Messages

	ApartmentNum	ApartmentName	Description	CleanedDate	AvailableDate
1	128	Town Place Apartments	Convenient walk to Parking	2022-05-20	2022-05-25
2	252	The Glenn	Close to downtown shops	2020-07-17	2020-07-13
3	316	Paradise Palms	A mile walk to the beach	2021-06-02	2021-06-08
4	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25

Step 13: Updating to Null

Update the Town Place Apartments so that it no longer has a description (i.e., its description is null) & confirm the change by selecting all rows.

```
UPDATE Apartments
SET description = NULL
WHERE ApartmentNum = 128;

SELECT *
FROM Apartments;
```

100 %

Results Messages

	ApartmentNum	ApartmentName	Description	CleanedDate	AvailableDate
1	128	Town Place Apartments	NULL	2022-05-20	2022-05-25
2	252	The Glenn	Close to downtown shops	2020-07-17	2020-07-13
3	316	Paradise Palms	A mile walk to the beach	2021-06-02	2021-06-08
4	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25

Step 14: Targeted Deletion

Delete all rows where the Cleaned date is greater than April 1, 2022. Then, select all rows to confirm the deletion.

```
DELETE FROM Apartments
WHERE CleanedDate > '2022-04-01';

SELECT *
FROM Apartments;
```

100 %

Results Messages

	ApartmentNum	ApartmentName	Description	CleanedDate	AvailableDate
1	252	The Glenn	Close to downtown shops	2020-07-17	2020-07-13
2	316	Paradise Palms	A mile walk to the beach	2021-06-02	2021-06-08

Section 3: Concepts Demonstration

Step 15: Data Anomalies

Create a table demonstrating insertion and deletion anomalies and discuss their implications for data integrity.

```
CREATE TABLE FashionDesigners (
DesignerName    VARCHAR(64),
Brand           VARCHAR(64),
YearFounded     SMALLINT,
Birthdate       DATE,
CountryOfBirth VARCHAR(64),
Education       VARCHAR(64),
NetWorth        DECIMAL(4,1)
);
```

100 %

Messages

Commands completed successfully.

a. Insertion Anomaly

```
INSERT INTO FashionDesigners (DesignerName,
Brand,
YearFounded,
Birthdate,
CountryOfBirth,
Education,
NetWorth)
VALUES ('Coco Chanel',
'Chanel',
1910,
CAST('19-AUG-1883' AS DATE),
'France',
'Self-educated',
15.0
);

INSERT INTO FashionDesigners (DesignerName,
Brand,
YearFounded,
Birthdate,
CountryOfBirth,
Education,
NetWorth)
VALUES ('Giorgio Armani',
'Armani',
1975,
CAST('11-JUL-1934' AS DATE),
'Italy',
'University of Milan',
900.0
);
```

```
INSERT INTO FashionDesigners (DesignerName,
Brand,
YearFounded,
Birthdate,
CountryOfBirth,
Education,
NetWorth)
VALUES ('Ralph Lauren',
'Ralph Lauren',
1967,
CAST('14-OCT-1939' AS DATE),
'United States',
'Baruch College',
800.0
);

INSERT INTO FashionDesigners (DesignerName,
Brand,
YearFounded,
Birthdate,
CountryOfBirth,
Education,
NetWorth)
VALUES ('Marc Jacobs',
'Marc Jacobs',
1984,
CAST('17-MAR-1963' AS DATE),
'United States',
'Parsons School of Design',
100.0
);
```

```
INSERT INTO FashionDesigners (DesignerName,
Brand,
YearFounded,
Birthdate,
CountryOfBirth,
Education,
NetWorth)
VALUES ('Tom Ford',
'Tom Ford',
2006,
CAST('27-AUG-1961' AS DATE),
'USA',
'Parsons School of Design',
400.0
);

SELECT *
FROM FashionDesigners;
```

100 %

Results Messages

	DesignerName	Brand	YearFounded	Birthdate	CountryOfBirth	Education	NetWorth
1	Coco Chanel	Chanel	1910	1883-08-19	France	Self-educated	15.0
2	Giorgio Armani	Armani	1975	1934-07-11	Italy	University of Milan	900.0
3	Ralph Lauren	Ralph Lauren	1967	1939-10-14	United States	Baruch College	800.0
4	Marc Jacobs	Marc Jacobs	1984	1963-03-17	United States	Parsons School of Design	100.0
5	Tom Ford	Tom Ford	2006	1961-08-27	USA	Parsons School of Design	400.0

As seen above, the same data is inserted multiple times with different values – causing an insertion anomaly. In particular, the "CountryOfBirth" of three of the fashion designers (i.e., 'Ralph Lauren', 'Marc Jacobs' & 'Tom Ford') is the United States; but two of the rows contain the value 'United States', while the remaining row contains the value 'USA'.

Given the inconsistency of the data that results from using different values to represent the same data (e.g., 'United States' vs. 'US'); we can say the data lacks data integrity. In other words, the data is not accurate or verifiable, as the "CountryOfBirth" will not always yield results that are reasonably sound. For example, if we wanted to filter the output to show all rows where the fashion designer's "CountryOfBirth" is the 'United States', Tom Ford would not be listed – even though he was born in the United States.

b. Deletion Anomaly

```
DELETE FROM FashionDesigners
WHERE CountryOfBirth= 'USA';

SELECT *
FROM FashionDesigners;
```

100 %

Results Messages

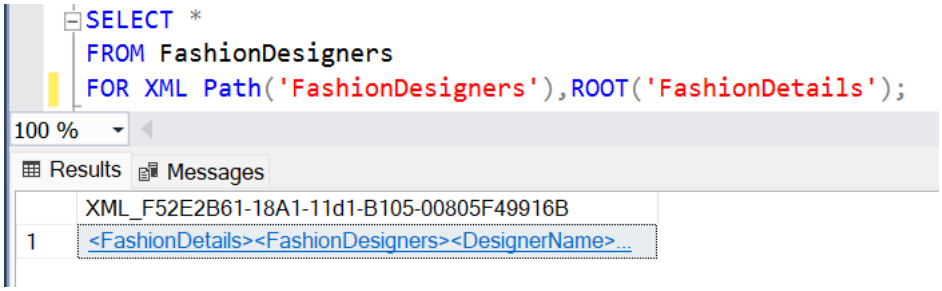
	DesignerName	Brand	YearFounded	Birthdate	CountryOfBirth	Education	NetWorth
1	Coco Chanel	Chanel	1910	1883-08-19	France	Self-educated	15.0
2	Giorgio Armani	Armani	1975	1934-07-11	Italy	University of Milan	900.0
3	Ralph Lauren	Ralph Lauren	1967	1939-10-14	United States	Baruch College	800.0
4	Marc Jacobs	Marc Jacobs	1984	1963-03-17	United States	Parsons School of Design	100.0

Deleting all rows where the value for the "CountryOfBirth" column is equal to 'USA' results in a deletion anomaly, as it not only deletes the "DesignerName" 'Tom Ford' from the database, but also (unintentionally) deletes data about the "Education" of the fashion designers (e.g., there were originally 2 designers in the database that attended Parsons School of Design) and the "CountryOfBirth" for the designers (e.g., there were originally 3 designers born in the United States).

In this example, a better idea might have been to update the value 'USA' in CountryOfBirth to 'United States'; ensuring data integrity in terms of accuracy (i.e., having information on all the designers that are born in the United States and all designers that received their education at Parsons School of Design) and validity (i.e., displays consistent data rather than multiple versions of the same data, like the value "United States" and "USA" for the country of birth).

Step 16: File & Database Table Comparison

File and Database Table Comparison – Create an XML file with similar data as the table in Step 15 and compare data access efficiency, security, and structural independence between a relational table and an XML file.


SELECT *
FROM FashionDesigners
FOR XML Path('FashionDesigners'),ROOT('FashionDetails');
100 %
Results
Messages
XML_F52E2B61-18A1-11d1-B105-00805F49916B
1
<FashionDetails><FashionDesigners><DesignerName>...

```xml
<FashionDetails>
  <FashionDesigners>
    <DesignerName>Coco Chanel</DesignerName>
    <Brand>Chanel</Brand>
    <YearFounded>1910</YearFounded>
    <Birthdate>1883-08-19</Birthdate>
    <CountryOfBirth>France</CountryOfBirth>
    <Education>Self-educated</Education>
    <NetWorth>15.0</NetWorth>
  </FashionDesigners>
  <FashionDesigners>
    <DesignerName>Giorgio Armani</DesignerName>
    <Brand>Armani</Brand>
    <YearFounded>1975</YearFounded>
    <Birthdate>1934-07-11</Birthdate>
    <CountryOfBirth>Italy</CountryOfBirth>
    <Education>University of Milan</Education>
    <NetWorth>900.0</NetWorth>
  </FashionDesigners>
  <FashionDesigners>
    <DesignerName>Ralph Lauren</DesignerName>
    <Brand>Ralph Lauren</Brand>
    <YearFounded>1967</YearFounded>
    <Birthdate>1939-10-14</Birthdate>
    <CountryOfBirth>United States</CountryOfBirth>
    <Education>Baruch College</Education>
    <NetWorth>800.0</NetWorth>
  </FashionDesigners>
  <FashionDesigners>
    <DesignerName>Marc Jacobs</DesignerName>
    <Brand>Marc Jacobs</Brand>
    <YearFounded>1984</YearFounded>
    <Birthdate>1963-03-17</Birthdate>
    <CountryOfBirth>United States</CountryOfBirth>
    <Education>Parsons School of Design</Education>
    <NetWorth>100.0</NetWorth>
  </FashionDesigners>
  <FashionDesigners>
    <DesignerName>Tom Ford</DesignerName>
    <Brand>Tom Ford</Brand>
    <YearFounded>2006</YearFounded>
    <Birthdate>1961-08-27</Birthdate>
    <CountryOfBirth>USA</CountryOfBirth>
    <Education>Parsons School of Design</Education>
    <NetWorth>400.0</NetWorth>
  </FashionDesigners>
</FashionDetails>
```

The database table represented in Step 15a-b was recreated and/or converted to Extensible Markup Language (XML) file format; where the same seven columns and five rows of data are stored in accordance with XML file format requirements. Given that we are dealing with information about fashion designers, the root element of the XML file was named "FashionDetails." Likewise, each of the five rows of data contain a "FashionDesigners" element, since each row represents a fashion designer; while each of the seven columns are represented with the associated column names from the database table: the "DesignerName", "Brand", "YearFounded", "Birthdate", "CountryOfBirth", "Education", & "NetWorth" elements.

a. Relational Table vs. XML File

Given the rise of the digital age and the increasing volume of data being generated, the ability to store, access, and manage data, both efficiently and securely, is pivotal – if not, indispensable. In terms of accessing the data, applications may use differing methods regarding the way the data is stored (i.e., the format/structure); and such disparities are highlighted in the efficiency, security, and structural independence – or lack thereof – of accessing the data in a relational database table versus in an XML file format.

Briefly, XML is a case-sensitive language that emphasizes the narrative of the data being accessed, rather than focusing on the way the data is stored and presented (Coronel & Morris, 2022). Its data elements must be formatted adequately in a hierarchical structure in terms of using tags and being "properly nested" (Coronel & Morris, 2022) – making the language or format a lot more meticulous and convoluted than that of the SQL programming language, used for relational tables/with relational database management systems (RDBMSs). For example, a quick glance at the data table would allow us to easily compare the net worths of each of the fashion designers (e.g., each record is listed horizontally, directly above/below one another); while mining through each line of data would be a lot more tedious and complex in the XML file format. In other words, if there were millions of rows of data, it would be more efficient to access a single record in the relational table format than in the XML file format.

In the context of database security and managing/controlling access to the data, a relational table might be preferred to the XML file format. For instance, suppose an application or a database administrator needed to restrict access to one record to allow only one single

person to access that record (i.e., meaning no other person has authorized access to this row of data, but may access the other records). When the data is stored in a relational database table, a simple SQL query might be helpful – in which the 'GRANT' and 'REVOKE' commands are leveraged to authorize user "privileges" or reverse such authorization (Coronel & Morris, 2023). In this case, the actual task of "granting" or "revoking" access to a particular record would be a lot more efficient in terms of the time and reduce the potential for data redundancy or islands of information (Coronel & Morris, 2023). This is in contrast to a more complex access control method in XML files; where access control lists (ACLs) (i.e., lists of objects/users who have access to each object) and capabilities (i.e., lists the users/the objects they have access to) are leveraged to control user access (Lee & Ting, 2014). Such circumstances set the stage for potential issues, such as data redundancies and data inconsistencies (Coronel & Morris, 2023). Moreover, ACLs and capabilities are not suitable for entities with convoluted structures, such as "element-level access control in XML, or row-level and cell-level access in relational databases" (Lee & Ting, 2014). Thus, it is much more difficult for database administrators to authorize/revoke individual user access to a specific row/record of data; and easier to secure this row in a relational table.

On the other hand, when frequent updates to the data are required or expected, relational tables are a better alternative to XML files (IBM Corporation, 2023). In particular, when altering the structure of the table by adding/removing columns, relational tables are more suitable in terms of their structural independence. That is, any change in the structure of a relational table – such as the addition/removal of a field (i.e., column in a table) – does not impact the application's ability to access the data (Coronel & Morris, 2023). However, for XML files, the data may only be altered by replacing the entire document (IBM Corporation, 2023); suggesting that the access to a file is dependent on its structure (i.e., structural dependence) (Coronel & Morris, 2023). In a similar vein, XML also lacks methods for data validation and the detection of 'NULL' values (i.e., missing values); unlike the 'CHECK' and 'NOT NULL' constraints used to define relational database tables/fields in SQL (Coronel & Morris, 2023). Ultimately, when the data must be altered by adding or removing columns, the structural independence of the relational table makes it more suitable for data storage/manipulation.

References

Coronel, C. & Morris, S. (2022, June 16). Database connectivity & web technologies. *Database systems: Design, implementation, and management* (14th ed., pp. 675-710). Cengage Learning.

IBM Corporation. (2023, January 20). *Comparison of the XML model and the relational model*. Documentation. https://www.ibm.com/docs/en/db2/11.5?topic=overview-comparison-xml-relational-models

Lee, D. & Ting, Y. (2014, May 15). XML access control. *Encyclopedia of Database Systems.* Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_790