PAPER NAME	AUTHOR
DBMS theory assignment.docx	D GR

WORD COUNT	CHARACTER COUNT
8187 Words	45571 Characters
PAGE COUNT	**FILE SIZE**
27 Pages	51.2KB
SUBMISSION DATE	**REPORT DATE**
Oct 1, 2024 12:44 PM GMT+5:30	Oct 1, 2024 12:45 PM GMT+5:30

● 38% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 31% Internet database
- 15% Publications database
- Crossref database
- Crossref Posted Content database
- 32% Submitted Works database

Introduction to Structured Query Language (SQL)

SQL – is an extension of computer programming, which addresses the data in a relational database. It is a common interface standard with which nearly all systems for managing databases, such as MySQL, PostgreSQL, Oracle and SQL server, are built.

Key Concepts in SQL

A Database: A database is a structured collection of interrelated data stored in tables.

A Table: A two-dimensional array made up of rows and columns, where a row is a single record, and a column is a field.

A Row: A record in the table's data set.

A Column: A single field in the given table.

A Schema: The design of the database which consists of attributes such as tables, columns etc, and their inter relations.

Data Definition Language (DDL): Involves commands that are carried out for the purpose of creating, altering, or dropping database objects.

CREATE: This is used to create database objects; they can come in the forms of tables, indexes, views, etc.

ALTER: This is used to change already existing database objects.

DROP: The use of this is meant for removal of objects from a database.

Data Manipulation Language (DML): Includes commands executed to change the content of the tables of the database.

INSERT: Used to input new records to the database by sections in the tables.

UPDATE: Used to edit half or the whole selected records that already exist in the database.

DELETE: Used to delete either selected records or the whole record table in the database.

Data Query Language (DQL): It is concerned about directing and understanding the data in a table.

SELECT: Always offers an outlet where data is retrieved from one or several tables.

More SQL Concepts
Join: This combines data from more than one table based on related columns.
Aggregates: Evaluate summary statistics such as SUM, AVG, COUNT, MIN, and MAX.
Subqueries: Nested queries within another more significant query.
Views: virtual tables created from existing ones.
Indexes: Data structures used to support fast querying of items.

SQL controls as well as manages relational databases for manipulation.
SQL is the basis for working with and manipulating relational databases. It is a standardized, or codified, language for discussing databases and makes tasks undertaken with them easier.

Key Roles of SQL in Database Management:
Data Definition:

Writing database structures: SQL statements like CREATE TABLE, CREATE INDEX, and CREATE VIEW are used to define the structure of databases, tables, indexes, and views.
Modifying Database Objects When you want to alter existing database objects, such as adding or removing columns from a table, change the type of a column, and other operations, you will use the ALTER statement.
Dropping Database Objects: One can delete database objects, that is, a table, an index, or any view, using the DROP statement.

Manipulating Data:

Insert data: the INSERT statement inserts rows into a table.
Update data: the UPDATE statement updates rows already existing in a table.
Delete data: the DELETE statement deletes rows from a table.
Querying Data:

Retrieving data: the SQL SELECT statement occurs most often in SQL. You can, for instance, extract data from tables as long as you want them based on specified conditions.
Filter data: Using the WHERE clauses, you can filter any query result to the criteria you want.
Join data: The JOIN clause allows you to combine multiple data from tables of related columns.
Aggregate data: SQL provides you with many built-in aggregate functions you can use on your data, including SUM, AVG, COUNT, MIN, and MAX. These can calculate summary statistics about your data.
Data Security and Integrity:

Granting and revoking privileges: SQL statements such as GRANT and REVOKE manage access to database objects.
Ensuring Data Integrity: Using constraints like PRIMARY KEY, UNIQUE, FOREIGN KEY, and CHECK will keep data integrity intact and thus prevent wrong input.
Database Administration:

Working with a transaction: SQL provides several ways to restore data integrity so that it remains in consistent and recoverable condition
Backup and recovery: SQL statements can be employed for creating backups of databases, from which copies may be reproduced when failure occurs.
Performance optimization: SQL helps analyze query performance as well as optimizing database structures for performance.

The History of SQL Development and Standardization
SQL, or Structured Query Language, passed through many changes after it was released. Its evolution and standardization depended upon the growing demand for interaction with relational databases using a single, common language.

Early Development Introduction

1970s: IBM developers developed a query language called SEQUEL (Structured English Query Language) for their research on the System R project. SEQUEL was meant to be a more user-friendly interface to access relational databases.

1974: SEQUEL was shortened to a very minimal form and named SQL.
Standardization Efforts
1986: ANSI and ISO initiated work on a standard for SQL.
1987 ANSI accepted its first SQL standard, ANSI X3.135-1987.
1989 ISO adopted the ANSI standard as ISO/IEC 9075:1989.
Follow-up Revisions
1992 ANSI and ISO published a new SQL standard, ANSI X3.135-1992 and ISO/IEC 9075:1992, which presented outer joins, triggers, and stored procedures.
1999 The standard updated the support for object-relational databases in the newer ANSI X3.135-1999 and ISO/IEC 9075:1999 with support for user-defined data types and inheritance.
2003. The SQL:2003 standard introduces support for XML and window functions, as well as recursive common table expressions.
2008. The SQL:2008 standard includes JSON and features designed to help with data warehousing, improved performance
2011. SQL:2011 adds temporal data types, sequences, and OLAP enhancements
2016. SQL:2016 brings new features like row pattern matching, JSON path expressions, and time and date improvements
2019. This standard introduced graph databases and enhanced time-series data.
Continuing Development
SQL standard is in constant evolution to meet changes as required by the database management systems and applications. It normally lists introduction of new features and improvements aimed at improving either performance or flexibility, or creating interoperability.

Key Drivers Pushing SQL Development and Standardization:

Increasing Complexity in the Databases: The method of SQL is needed to be more potent with regard to handling and querying data, the more complex and data-intensive databases become.
Advancement in technology: There have been changes in SQL from new technologies such as object-relational databases, XML, and JSON.
Industry needs: SQL had ensured that different systems have an interoperability, with compatibility within databases.

Basic SQL Statement Structure
SQL statements are also structured similarly. However, the type of operation dictates the way they look. Here is a general outline of the basic elements of an SQL statement:

1. Keywords:
The SQL keywords are such reserved words used in SQL. They outline the actual operation to be performed.
Examples of keywords are: SELECT, FROM, WHERE, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, etc.
2. Tables:
Names of the tables are used to outline the database tables that are being referenced in the statement.
Tables are groupings of data arranged into rows and columns
3. Columns
Column names are the particular columns or fields within a given table that is being referenced or updated.
Columns are individual elements within a row of a table
4. Operators:
Operators are either symbols or keywords that are used to compare things, perform logical operations, arithmetic calculations, and many more.

Examples of operators include: =, <, >, AND, OR, NOT, +,-, *, /
5. Values:
Values refer to the actual data being inserted, updated, or retrieved.
Values may be literals ('John Doe', 123), variables, or expressions.
6. Clauses:
Clauses are optional parts of a SQL statement which give extra conditions or specifications.
Some common clauses are :
WHERE: Filters rows based on certain conditions.
FROM: Specifies one or more tables to query.
GROUP BY: Groups rows based on certain columns.
HAVING: Filters groups based on conditions.
ORDER BY: Orders the results by specific columns.
Example: A Simple SELECT Statement
SQL
SELECT * FROM Customers;

This statement retrieves all the columns and rows from the Customers table.

Example: A More Complex SELECT Statement
SQL
SELECT FirstName, LastName FROM Customers WHERE City = 'New York';

This statement is to retrieve only the FirstName and LastName columns from the Customers table, but only for customers who live in the city "New York".
DDL and DML Assignment.docx

1 / 9
Data definition language
A Data definition language means it gives reference to a language that is used to modify data and helps in the definition of the data structures. For instance, the DDL commands could be used to remove, add, and modify tables within a database. The DDLs used in data base applications are considered to be a subset of the Structured Query Language.
List of DDL Commands:
Here are all the Data definition language commands with their syntax
Command Description Syntax
CREATE
Create database or its objects (table, index, function, views, store procedure, and triggers)
CREATE TABLE tablename (column1 datatype, column2 datatype, ...);
DROP
Delete objects from the database drop table tablename;
ALTER
Alter the structure of the database alter TABLE tablename add column columnname datatype;
TRUNCATE
Remove all records from a table, including all spaces allocated for the records are removed truncate Ttable tablename;
COMMENT
Add comments to the data dictionary COMMENT 'comment-text' ON TABLE tablename;
RENAME
Rename an object existing in the database RENAME TABLE oldtable-name TO new_table-name;

DDL (Data Definition Language)
actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.
DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.
List of DDL Commands:
2 / 9
Here are all the main DDL (Data Definition Language) commands along with their syntax:
Command Description Syntax
CREATE
Create database or its objects (table, index, function, views, store procedure, and triggers)
CREATE TABLE table_name (column1 data_type, column2 data_type, ...);
DROP
Delete objects from the database DROP TABLE tablename;
ALTER
Alter the structure of the database ALTER TABLE tablename ADD COLUMN columnname datatype;
TRUNCATE
Remove all records from a table, including all spaces allocated for the records are removed
TRUNCATE TABLE table_name;
COMMENT
Add comments to the data dictionary COMMENT 'commenttext' ON TABLE tablename;
RENAME
Rename an object existing in the database RENAME TABLE oldtable_name TO newtablename;

DML (Data Manipulation Language)
The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.
It is the component of the SQL statement that controls access to data and to the database.
Basically, DCL statements are grouped with DML statements.
List of DML commands
Here are all the main DML (Data Manipulation Language) commands along with their syntax:
Command Description Syntax
INSERT
Insert data into a table INSERT INTO tablename (column1, column2, ...) VALUES (value1, value2, ...);
UPDATE
3 / 9
Update existing data within a table UPDATE tablename SET column1 = value1, column2 = value2 WHERE condition;
DELETE
Delete records from a database table DELETE FROM tablename WHERE condition;
LOCK
Table control concurrency LOCK TABLE tablename in lock-mode;
CALL a PL/SQL or JAVA subprogram CALL procedurename(arguments);
EXPLAIN PLAN Describe the access path to data EXPLAIN PLAN FOR SELECT * FROM tablename;

SQL Commands
o SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
o SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

There are four types of SQL commands: DDL, DML, DCL, TCL.

Data Definition Language (DDL)
o DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
o All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Following are the some commands that come under DDL:

a. CREATE It is used to create a new table in the database.
Syntax:
<
1. CREATE TABLE TABLE_NAME (COLUMN_NAMES DATATYPES [...]);
4 / 9
In above statement, TABLE_NAME is the name of the table, COLUMN_NAMES is the name of the columns and DATATYPES is used to define the type of data.
Example:
1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
b. DROP: It is used to delete both the structure and record stored in the table.
Syntax: To DROP a table permanently from memory
1. DROP TABLE tablename [cascade constraint];
The cascade constraint is an optional parameter which is used for tables which have foreign keys that reference the table being dropped. If cascade constraint is not specified and used attempt to drop a table that has records in a child table, then an error will occur. So by using cascade constraints, all child table foreign keys are dropped.
Example
1. DROP TABLE EMPLOYEE;
c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
Following are the list of modifications that can be done using ALTER command.
o With the use of ALTER commands we can add or drop one or more columns form existing tables.
o Increase or decrease the existing column width by changing the data type
o Make an existing mandatory column to optional.
o Enable or disable the integrity constraints in a table. We can also add, modify or delete the integrity constraints from a table.
o We can also specify a default value for existing column in a table.
Adding new columns in Table:
With the use of ALTER table command we can add new columns existing table.
Syntax: To add a new column in the table
1. ALTER TABLE tablename add columnname column-definition;
In the above syntax, where tablename corresponds to the name of the table, column-definition corresponds to the valid specifications for a column name and data type.

EXAMPLE:
1. ALTER TABLE STU_DETAILS ADD (ADHAR_NUM VARCHAR2 (15));
Syntax: To ADD a multiple column from a table.
ALTER TABLE tablename ADD column_name1, column_name2;
Example:
5 / 9
1. ALTER TABLE STU_DETAILS ADD ADHAR_NUM, NAME;
Adding constraints in a Table:
You can also add constraints to an existing table. For example: If you forget to add a primary key constraint to a table during creation, you can add it using the ALTER TABLE statement.

Syntax: To ADD a constraint from a table.
1. ALTER TABLE tablename ADD (columnname column-definition CONSTRAINT constraintname);
Example:
1. ALTER TABLE STU_DETAILS ADD (CONSTRAINT PK_STU_DETAILS PRIMARY KEY (STU_ID);
Following points should be kept in mind while adding new columns/relationships to existing tables.
o No need for parentheses if you add only one column or constraints.
o You can add a column at any time if NULL is not specified. You can add a new column with NOT NULL if the table is empty.
Modifying Column using alter:
With the use of ALTER table we can modify column and constraint in the existing table. These statements can increase or decrease the column widths and changing a column from mandatory to optional.
Syntax:
1. alter table tablename modify (column definitions....);
Example:
1. ALTER TABLE STU_DETAILS MODIFY (ADHAR_NUM VARCHAR2 (20));
SQL does not allow column widths to be reduced even if all column values are of valid length. So the values should be set to NULL to reduce the width of the columns. It is also not possible to reduce the width of the ADHAR_NUM column from 18 to 12 even if all values in the ADHAR_NUM column are less than 12 characters, unless all al values in the name column are null. You can modify the column form NULL to NOTNULL constraints if there is no record in that column in the table.
Example:
1. ALTER TABLE STU_DETAILS MODIFY (ADHAR_NUM VARCHAR2 (20) NOT NULL);
Drop column and constraints using ALTER
You cannot only modify columns but you can also drop them entirely if it is no longer required in a table. Using drop statement in alter command we can also remove the constraints form the table.
Syntax: To drop a column from a table.
1. ALTER TABLE tablename DROP COLUMN columnname;
Example:
6 / 9
1. ALTER TABLE STU_DETAILS DROP COLUMN ADHAR_NUM;
Syntax: To drop a multiple column from a table.
1. ALTER TABLE table_name DROP COLUMN column_name1, column_name2;
Example:
1. ALTER TABLE STU_DETAILS DROP COLUMN ADHAR_NUM, NAME;
Syntax: To drop a constraint from a table.
1. ALTER TABLE table_name DROP CONSTRAINT constraint_name;
Example:
1. ALTER TABLE STU_DETAILS DROP CONSTRAINT FK_STU_DETAILS;

Following points should be kept in mind while deleting columns/associations:
o You cannot drop columns in a table. If you want to drop a column from a table, the deletion is permanent so you cannot undo the column if you accidentally drop the wrong column.
o You cannot drop a column whose username is SYS.
o If you want to drop a primary key column unless you drop the foreign keys that belong to it then use cascade keyword for this.
Example:
1. ALTER TABLE STU_DETAILS DROP PRIMARY KEY CASCADE;
o You can also enable or disable the key constraint in a table. It can be done in various situations such as: when loading large amount of data into table, performing batch operations, migrating the organizations legacy data.
Example: To disable constraint
1. ALTER TABLE STU_DETAILS DISABLE CONSTRAINT FK_STU_DETAILS;
Example: To Enable constraint
1. ALTER TABLE STU_DETAILS ENABLE CONSTRAINT FK_STU_DETAILS;
o Instead of dropping a column in a table, we can also make the column unused and drop it later on. It makes the response time faster. After a column has been marked as unused, the column and all its contents are no longer available and cannot be recovered in the future. The unused columns will not be retrieved using Select statement
Example:
1. ALTER TABLE STU_DETAILS SET UNUSED COLUMN ADHAR_NUM;
RENAMING TABLE
SQL provides the facility to change the name of the table by using a ALTER TABLE statement.
Syntax:
7 / 9
1. ALTER TABLE <OLD_TABLENAME> Rename to <NEW_TABLENAME>;
Example:
1. ALTER TABLE STU_NAME Rename to STUDENT_NAME;
d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:
1. TRUNCATE TABLE tablename;
Example:
1. TRUNCATE TABLE EMPLOYEE;
e. Rename: It is used to rename the table.

Syntax:
1. Rename <OLD_TABLENAME> to <NEW_TABLENAME>;
In the above syntax, Rename is a command, <OLD_TABLENAME> is the name of the table and <NEW_TABLENAME> is the name that you have changed.

Example:
1. Rename STU_NAME to STUDENT_NAME;
2. Data Manipulation Language
o DML commands are used to modify the database. It is responsible for all form of changes in the database.
o The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Following are the some commands that come under DML:

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table. To insert a new row into a table you must be your on schema or INSERT privilege on the table.
Following are the list of points should be considered while inserting the data into tables.
o SQL uses all the columns by default if you do not specify the column name while inserting a row.
o The number of columns in the list of column name must match the number of values that appear in parenthesis after the word "values".
8 / 9
o The data type for a column and its corresponding value must match.
Syntax: To add row in a table
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, ... value-n);
Or
1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, ... value-n);
In the above syntax, TABLE_NAME is the name of the table in which the data will be inserted. The (col1, col2, col3, col N) are optional and name of the columns in which values will be inserted. The value1 corresponds to the value of be inserted in col1 and similarly value2 corresponds to the value of be inserted in col2 and so on.
For example:
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
Syntax: To add multiple rows in a table
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, ... value-n), (value1, value2, value3, ... value-n);
For example:
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Hayat", "DBMS"), ("Asli", "DBMS"), ("Gulbahar", "DBMS");
b. update: This command is used to update or modify the value of a column in the table.
Syntax: To update record in a table
1. update tablename set fieldname= "Something" with where condition ,and if where condition is not given then the attribute name mentioned in this quere will have same values in all it tuples.
The following the list of points should be remembered while executing the update statement.
o It references only one table.
o In the set clause at least one column must be assigned an expression for the update statement,
o In the where clause you could also give multiple conditions for update statement.
For example:
1. update students
2. set user-name = 'Hayat'
3. where student_Id = '3'
9 / 9
c. delete: It is used to remove one or more row from a table. To delete rows from the table, it must be in your schema or you must have delete privilege.
Syntax: To Delete a record from table
1. delete from tablename where condition;
In the above syntax, condition is used in the where clause to filter the records that are actually being removed. You can remove zero or more rows from a table. If you do not use where condition then DELETE statement will remove all the rows from the table. You can also use one or multiple conditions in WHERE clause.

For example:

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

d. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax: It is used for retrieve the records from table

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

Comprehensive Guide to SQL Transactions and Concurrency Control

1. Understanding Transactions and Their Properties (ACID) in SQL

In the world of database management, transactions play a crucial role in maintaining data integrity and consistency. Let's dive deep into what transactions are and why they're so important.

What is a Transaction?

A transaction, in the context of SQL databases, is a sequence of one or more SQL statements that are executed as a single unit of work. We can think of it as an atomic operation - it either completes entirely or not at all. This all-or-nothing approach is fundamental to maintaining data consistency in complex database operations.

The ACID Properties

When we talk about transactions, we often refer to the ACID properties. ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. Let's explore each of these in detail:

Atomicity

Atomicity ensures that all operations within a transaction are treated as a single unit. If any part of the transaction fails, the entire transaction is rolled back, and the database is left unchanged. It's like an all-or-nothing principle.

For example, consider a bank transfer:

```sql
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 'A';
UPDATE accounts SET balance = balance + 1000 WHERE account_id = 'B';
COMMIT;
```

If either update fails, the entire transaction is rolled back, ensuring that money isn't deducted without being credited elsewhere.

Consistency

Consistency guarantees that a transaction brings the database from one valid state to another. This means all data integrity constraints must be satisfied after the transaction completes.

For instance, if we have a constraint that an account balance can't be negative:

```
CREATE TABLE accounts (
    account_id VARCHAR(10) PRIMARY KEY,
    balance DECIMAL(10,2) CHECK (balance >= 0)
);
```

Any transaction that would result in a negative balance would be rolled back, maintaining consistency.

Isolation

Isolation ensures that concurrent execution of transactions leaves the database in the same state as if the transactions were executed sequentially. This property prevents interference between simultaneous transactions.

We'll dive deeper into isolation levels later, but here's a quick example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;
-- Perform operations
COMMIT;
```

Durability

Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure. This is typically achieved through transaction logs and database backups.

For example, most database systems write to a transaction log before confirming a commit:

```
COMMIT;
-- Database writes to transaction log
-- Only then does it return success to the client
```

Importance of ACID Properties

The ACID properties are crucial for several reasons:

1. **Data Integrity**: They ensure that our data remains accurate and consistent, even in the face of system failures or concurrent access.
2. **Reliability**: We can rely on transactions to either complete fully or not at all, which is essential for many business processes.
3. **Concurrency**: They allow multiple users to work with the same data simultaneously without interfering with each other.
4. **Recovery**: In case of system failures, ACID properties ensure that the database can be recovered to a consistent state.

Implementing Transactions in Different Database Systems

While the concept of transactions is universal, the implementation can vary slightly between different database management systems:

MySQL / MariaDB

```
START TRANSACTION;
-- SQL statements
COMMIT;
```

SQL Server

```
BEGIN TRANSACTION;
-- SQL statements
COMMIT TRANSACTION;
```

In our next section, we'll explore how to begin, commit, and roll back transactions in more detail, providing practical examples and best practices for managing transactions effectively.

2. Beginning, Committing, and Rolling Back Transactions using SQL

Now that we understand what transactions are and why they're important, let's dive into the practical aspects of working with transactions in SQL. We'll cover how to begin a transaction, commit changes, and roll back when necessary.

Beginning a Transaction

Most SQL databases allow us to explicitly start a transaction. The exact syntax may vary slightly between different database management systems, but the concept remains the same.

Standard SQL

```
START TRANSACTION;
```

or

```
BEGIN TRANSACTION;
```

Example in Practice

Let's say we're developing an e-commerce application. When a customer places an order, we need to update multiple tables: `orders`, `order_items`, and `inventory`. We'd start our transaction like this:

```
START TRANSACTION;
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1001, CURRENT_DATE, 150.00);

SET @order_id = LAST_INSERT_ID();

INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES (@order_id, 5001, 2, 75.00);
```

```
UPDATE inventory SET stock_quantity = stock_quantity - 2
WHERE product_id = 5001;
```

Committing a Transaction

Once we've executed all the statements in our transaction and we're sure everything is correct, we need to commit the transaction to make the changes permanent. If all our operations were successful, we'd commit the transaction:

```
COMMIT;
```

At this point, the new order has been created, the order items have been added, and the inventory has been updated. These changes are now permanent and visible to other transactions.

Rolling Back a Transaction

If something goes wrong during our transaction, or if we decide we don't want to keep the changes we've made, we can roll back the transaction. This undoes all the changes made since the transaction began.

Standard SQL

```
ROLLBACK;
```

Example of Rolling Back

Let's say we're processing our order, but we find out that we don't have enough inventory:

```
START TRANSACTION;
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1001, CURRENT_DATE, 150.00);
SET @order_id = LAST_INSERT_ID();
INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES (@order_id, 5001, 2, 75.00);
SELECT @available_quantity := stock_quantity
FROM inventory
WHERE product_id = 5001;
IF @available_quantity < 2 THEN
    ROLLBACK;
    SELECT 'Not enough inventory. Order cancelled.' AS message;
ELSE
    UPDATE inventory SET stock_quantity = stock_quantity - 2
    WHERE product_id = 5001;
    COMMIT;
    SELECT 'Order processed successfully.' AS message;
END IF;
```

In this example, if we don't have enough inventory, we roll back the entire transaction, cancelling the order and leaving the database unchanged.

Savepoints

Some database systems also support savepoints, which allow us to create points within a transaction that we can roll back to without rolling back the entire transaction.

```sql
START TRANSACTION;

-- Some SQL statements

SAVEPOINT my_savepoint;

-- More SQL statements

-- If something goes wrong:
ROLLBACK TO SAVEPOINT my_savepoint;

-- Continue with the transaction
```

Best Practices for Managing Transactions

1. **Keep Transactions Short**: Long-running transactions can lead to concurrency issues and impact system performance.

2. **Only Include Necessary Operations**: Include only the operations that must be executed atomically in a transaction.

3. **Handle Errors Properly**: Always include error handling to ensure transactions are rolled back if something goes wrong.

4. **Be Mindful of Locks**: Transactions often involve locks on database objects. Be aware of potential deadlock situations.

5. **Use Appropriate Isolation Levels**: Choose the right isolation level for your needs to balance consistency and performance.

6. **Consider Using Stored Procedures**: Encapsulating transactions in stored procedures can help ensure consistent handling of transactions across your application.

7. **Avoid Mixing DDL and DML in Transactions**: Some databases don't support rolling back DDL (Data Definition Language) statements.

3. Isolation Levels and Their Impact on Data Consistency using SQL

When we work with transactions in a multi-user database environment, we need to consider how concurrent transactions interact with each other. This is where isolation levels come into play. Isolation levels define the degree to which one transaction must be isolated from resource or data modifications made by other concurrent transactions.

Understanding Isolation Levels

SQL defines four standard isolation levels, each with different rules about what kinds of data reads and writes are allowed. Let's explore each of these levels and their implications for data consistency.

READ UNCOMMITTED

This is the lowest isolation level. In this level, transactions can read data that has been modified by other transactions but not yet committed.

Pros: - Highest concurrency - Lowest overhead

Cons: - Allows dirty reads, non-repeatable reads, and phantom reads - Can lead to inconsistent data

Example:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN TRANSACTION;

SELECT * FROM accounts WHERE account_id = 1001;

-- This might read data that another transaction has modified but not yet
committed

COMMIT;
```

READ COMMITTED

This level ensures that any data read was committed at the moment it was read. It prevents dirty reads but allows non-repeatable reads and phantom reads.

Pros: - Prevents dirty reads - Good balance of concurrency and consistency for many applications

Cons: - Allows non-repeatable reads and phantom reads

Example:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE account_id = 1001;

-- If we read balance again, it might be different if another transaction
-- has modified and committed the data

COMMIT;
```

REPEATABLE READ

This level ensures that if a transaction reads a row, it will consistently see the same data for that row until the transaction completes. It prevents dirty reads and non-repeatable reads but still allows phantom reads.

Pros: - Prevents dirty reads and non-repeatable reads - Provides strong consistency for read operations

Cons: - May reduce concurrency - Still allows phantom reads

Example:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;

SELECT * FROM accounts WHERE balance > 1000;

-- If we run this query again, we'll see the same rows, even if another
transaction
-- has changed some balances. However, we might see new rows (phantom
reads).

COMMIT;
```

SERIALIZABLE

This is the highest isolation level. It completely isolates one transaction from others, making transactions appear as if they were executed serially (one after the other) rather than concurrently.

Pros: - Prevents all concurrency side effects (dirty reads, non-repeatable reads, and phantom reads) - Provides the highest level of consistency

Cons: - Lowest concurrency - Can lead to performance issues in high-concurrency environments

Example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;

SELECT * FROM accounts WHERE balance > 1000;

-- This query is guaranteed to return the same result if run multiple
times
-- within the transaction, and no other transactions can modify data that
-- would affect this result set.

COMMIT;
```

Concurrency Phenomena

To understand the impact of isolation levels, we need to be familiar with the concurrency phenomena they prevent:

1. **Dirty Read**: Reading uncommitted changes from another transaction.
2. **Non-repeatable Read**: Getting different results when reading the same data twice in a transaction.
3. **Phantom Read**: When a transaction retrieves a set of rows twice and gets a different set of rows.

Here's a summary of which phenomena are prevented by each isolation level:

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
READ UNCOMMITTED	No	No	No
READ COMMITTED	Yes	No	No
REPEATABLE READ	Yes	Yes	No
SERIALIZABLE	Yes	Yes	Yes

Choosing the Right Isolation Level

Selecting the appropriate isolation level involves balancing data consistency needs with performance requirements:

1. **READ UNCOMMITTED**: Use when you need the highest level of concurrency and can tolerate inconsistent reads.
2. **READ COMMITTED**: A good default choice for many applications. It prevents dirty reads while allowing good concurrency.
3. **REPEATABLE READ**: Use when you need consistent reads within a transaction and can tolerate phantom reads.
4. **SERIALIZABLE**: Use when you need the highest level of consistency and isolation, and concurrency is less of a concern.

Implementing Isolation Levels

The way we set isolation levels can vary between database systems. Here are examples for some popular databases:

MySQL

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

SQL Server

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Best Practices for Working with Isolation Levels

1. **Understand Your Requirements**: Analyze your application's needs in terms of data consistency and concurrency.
2. **Start with READ COMMITTED**: Unless you have specific reasons to use a different level, READ COMMITTED is often a good default.
3. **Use Higher Isolation Levels Judiciously**: Higher levels provide more consistency but at the cost of concurrency. Use them only when necessary.
4. **Be Aware of Lock Escalation**: Higher isolation levels often involve more locking, which can lead to deadlocks if not managed carefully.

5. **Test Thoroughly**: Always test your application under concurrent load to ensure it behaves correctly with your chosen isolation level.

6. **Consider Optimistic Concurrency Control**: For high-concurrency scenarios, consider using optimistic concurrency control instead of relying solely on isolation levels.

By understanding and properly implementing isolation levels, we can ensure that our database transactions maintain the right balance between data consistency and system performance. In the next section, we'll explore how to handle concurrent transactions and ensure data integrity in more complex scenarios.

4. Handling Concurrent Transactions and Ensuring Data Integrity using SQL

In a multi-user database environment, handling concurrent transactions while maintaining data integrity is crucial. We need to implement strategies to prevent data inconsistencies and ensure that our database remains in a valid state even when multiple users are interacting with it simultaneously.

Understanding Concurrency Control

Concurrency control is the process of managing simultaneous operations on the database without having them interfere with one another. The main goals of concurrency control are:

1. To ensure data integrity
2. To provide a consistent view of the data
3. To maximize concurrent access to the data ## Conclusion

Ensuring data integrity in transactional systems requires a multi-faceted approach:

1. Utilize database constraints to enforce basic rules at the database level.
2. Implement triggers and stored procedures to encapsulate and consistently apply business logic.
3. Use appropriate isolation levels to balance consistency needs with performance requirements.
4. Implement optimistic or pessimistic concurrency control mechanisms based on your specific use case.
5. Handle distributed transactions carefully, using patterns like Two-Phase Commit or Sagas when necessary.
6. Consider eventual consistency models for systems that prioritize availability and partition tolerance.

By combining these techniques and carefully designing our database and application logic, we can create robust, concurrent systems that maintain data integrity even under high load and complex operational scenarios. Let's explore various techniques and best practices for achieving these goals.

Locking Mechanisms

Locking is one of the primary mechanisms used for concurrency control. There are two main types of locks:

1. **Shared Locks (S-locks)**: Used for read operations. Multiple transactions can hold shared locks on the same data simultaneously.

2. **Exclusive Locks (X-locks)**: Used for write operations. Only one transaction can hold an exclusive lock on a piece of data at a time.

Here's an example of how we might use locking in a transaction:

```sql
BEGIN TRANSACTION;
SELECT * FROM accounts WITH (XLOCK) WHERE account_id = 1001;
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1001;
COMMIT;
```

In this example, we're using an exclusive lock to ensure that no other transaction can modify the account data while we're updating the balance. This prevents issues like lost updates, where two transactions might try to modify the same data simultaneously.

Deadlock Detection and Prevention

When using locks, we need to be aware of the potential for deadlocks. A deadlock occurs when two or more transactions are waiting for each other to release locks, resulting in a circular dependency. Most database systems have built-in deadlock detection mechanisms. For example, in SQL Server, we can set a deadlock priority:

```sql
SET DEADLOCK_PRIORITY HIGH;
BEGIN TRANSACTION;

COMMIT;
```

To prevent deadlocks, we can follow these best practices:

1. **Access objects in a consistent order**: If multiple transactions need to lock the same set of objects, always lock them in the same order.

2. **Keep transactions short**: The longer a transaction holds locks, the more likely it is to conflict with other transactions.

3. **Use appropriate isolation levels**: Higher isolation levels generally involve more locking, increasing the risk of deadlocks.

Optimistic vs. Pessimistic Concurrency Control

We have two main approaches to concurrency control:

Pessimistic Concurrency Control

This approach assumes that conflicts between transactions are likely and uses locks to prevent them. We've been discussing this approach in the context of locking mechanisms.

Optimistic Concurrency Control

This approach assumes that conflicts are rare and allows transactions to proceed without locking. When a transaction is ready to commit, it checks if any conflicts have occurred and aborts if necessary.

Here's an example of optimistic concurrency control:

```sql
BEGIN TRANSACTION;

-- Read the current value
SELECT @original_balance = balance FROM accounts WHERE account_id = 1001;

-- Perform some calculations...
SET @new_balance = @original_balance - 100;

-- Try to update, but only if the balance hasn't changed
UPDATE accounts
SET balance = @new_balance
WHERE account_id = 1001 AND balance = @original_balance;

-- Check if the update was successful
IF @@ROWCOUNT = 0
    BEGIN
        -- The balance changed, so we need to handle the conflict
        ROLLBACK;
        THROW 51000, 'Concurrency conflict occurred.', 1;
    END
ELSE
    COMMIT;
```

In this example, we only update the balance if it hasn't changed since we first read it. If it has changed, we roll back the transaction and handle the conflict.

Ensuring Data Integrity

Beyond concurrency control, we need to implement other measures to ensure data integrity:

Constraints

SQL provides several types of constraints that help maintain data integrity:

1. **Primary Key Constraints**: Ensure each row in a table is uniquely identifiable.

```sql
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

2. **Foreign Key Constraints**: Maintain referential integrity between tables.

```sql
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

3. **Check Constraints**: Enforce domain integrity by limiting the values that can be placed in a column.

```
ALTER TABLE accounts
ADD CONSTRAINT chk_positive_balance CHECK (balance >= 0);
```

4. **Unique Constraints**: Ensure that all values in a column or set of columns are unique.

```
ALTER TABLE customers
ADD CONSTRAINT unique_email UNIQUE (email);
```

Triggers

Triggers are special stored procedures that automatically run when certain events occur in the database. We can use them to enforce complex business rules and maintain data integrity.

Here's an example of a trigger that ensures the total order amount matches the sum of its line items:

```
CREATE TRIGGER check_order_total
ON order_items
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    UPDATE orders
    SET total_amount = (
        SELECT SUM(quantity * price)
        FROM order_items
        WHERE order_items.order_id = orders.order_id
    )
    WHERE order_id IN (
        SELECT DISTINCT order_id
        FROM inserted
        UNION
        SELECT DISTINCT order_id
        FROM deleted
    );
END;
```

This trigger automatically updates the total amount in the `orders` table whenever order items are inserted, updated, or deleted.

Handling Concurrent Access in Application Code

While much of concurrency control is handled at the database level, we also need to consider how our application code interacts with the database:

1. **Use Transactions Wisely**: Wrap related operations in transactions to ensure atomicity.

2. **Implement Retry Logic**: When a transaction fails due to a concurrency conflict, implement logic to retry the operation.

3. **Use Connection Pooling**: This helps manage database connections efficiently in high-concurrency scenarios.

4. **Consider Using Stored Procedures**: Stored procedures can encapsulate complex logic and reduce network traffic, which can be beneficial in concurrent environments.

Here's an example of how we might implement retry logic in our application code (using Python with SQLAlchemy):

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import OperationalError
engine = create_engine('postgresql://user:password@localhost/dbname')
Session = sessionmaker(bind=engine)
def update_account_balance(account_id, amount):
    max_retries = 3
    retries = 0
    while retries < max_retries:
        try:
            session = Session()
            with session.begin():
                account =
session.query(Account).filter_by(id=account_id).with_for_update().one()
                account.balance += amount
                session.commit()
            return True
        except OperationalError:
            retries += 1
            if retries == max_retries:
                raise
            session.rollback()
    return False
```

In this example, we're using SQLAlchemy's `with_for_update()` method to lock the row we're updating, and we're implementing retry logic to handle potential concurrency conflicts.

Monitoring and Tuning

To ensure our concurrency control strategies are effective, we need to monitor our database performance and tune as necessary:

1. **Use Database Profiling Tools**: Most database systems provide tools to monitor query performance and lock contention.

2. **Analyze Transaction Logs**: Regular analysis of transaction logs can help identify patterns of conflicts or inefficiencies.

3. **Implement Performance Metrics**: Track key metrics like transaction throughput and response times in your application.

4. **Regular Maintenance**: Ensure that statistics are up-to-date and indexes are optimized to support your query patterns.

5. Advanced Topics and Best Practices in SQL Transaction Management and Concurrency Control

As we wrap up our comprehensive guide to SQL transactions and concurrency control, let's explore some advanced topics and best practices that can help us take our database management skills to the next level.

Distributed Transactions

In modern, distributed systems, we often need to coordinate transactions across multiple databases or even different types of data stores. This is where distributed transactions come into play.

22 *Two-Phase Commit Protocol (2PC)*

The Two-Phase Commit protocol is a distributed algorithm that ensures all nodes in a distributed system agree to commit a transaction before the transaction is actually committed.

1. 49 **Prepare Phase**: The coordinator asks all participants if they're ready to commit.
2. **Commit Phase**: If all participants agree, the coordinator tells everyone to commit. If any participant can't commit, everyone is told to abort.

While 2PC ensures consistency, it can be slow and susceptible to coordinator failures. Modern systems often use alternative approaches like sagas or eventual consistency.

89 *Saga Pattern*

The Saga pattern is an alternative to distributed transactions that's often used in microservices architectures. 18 A saga is a sequence of local transactions where each transaction updates data within a single service. If a step fails, the saga executes compensating transactions to undo the changes made by the preceding steps. Here's a conceptual example of how we might implement a saga for a hotel booking system:

```sql
-- Begin Saga
BEGIN TRANSACTION;

-- Step 1: Reserve Hotel Room
INSERT INTO room_reservations (room_id, guest_id, date) VALUES (101, 1,
'2023-09-01');

-- Step 2: Charge Credit Card
INSERT INTO payments (guest_id, amount, status) VALUES (1, 200.00,
'pending');

-- Step 3: Send Confirmation Email
INSERT INTO email_queue (guest_id, template, status) VALUES (1,
'booking_confirmation', 'pending');

-- If all steps succeed, commit the transaction
```

```
COMMIT;

-- If any step fails, we need to execute compensating transactions
-- ROLLBACK;
-- DELETE FROM room_reservations WHERE room_id = 101 AND date = '2023-09-
01';
-- UPDATE payments SET status = 'cancelled' WHERE guest_id = 1 AND amount
= 200.00;
-- DELETE FROM email_queue WHERE guest_id = 1 AND template =
'booking_confirmation';
```

Optimistic Concurrency Control with Versioning

We discussed optimistic concurrency control earlier, but let's explore a more advanced implementation using versioning.

```
CREATE TABLE accounts (
    account_id INT PRIMARY KEY,
    balance DECIMAL(10,2),
    version INT
);

-- To update an account:
BEGIN TRANSACTION;

DECLARE @current_version INT;
DECLARE @new_balance DECIMAL(10,2);

-- Read the current balance and version
SELECT @current_version = version, @new_balance = balance
FROM accounts
WHERE account_id = 1001;

-- Calculate new balance
SET @new_balance = @new_balance - 100.00;

-- Try to update, incrementing the version
UPDATE accounts
SET balance = @new_balance, version = @current_version + 1
WHERE account_id = 1001 AND version = @current_version;

-- Check if update was successful
IF @@ROWCOUNT = 0
BEGIN
    -- Concurrency conflict occurred
    ROLLBACK;
    THROW 51000, 'Concurrency conflict detected', 1;
END
ELSE
BEGIN
    COMMIT;
END
```

This approach uses a version number that's incremented with each update. If the version hasn't changed when we try to update, we know no other transaction has modified the data since we read it.

Handling Long-Running Transactions

Long-running transactions can be problematic in high-concurrency environments as they hold locks for extended periods. Here are some strategies to handle them:

1. **Break into Smaller Transactions**: If possible, break long-running operations into smaller, more manageable transactions.

2. **Use Snapshot Isolation**: This allows long-running read transactions without blocking writes.

3. **Implement Checkpoints**: For very long operations, implement checkpoints where you can commit partial work.

4. **Consider Async Processing**: Move long-running operations to background jobs when possible.

Implementing Row-Level Security

Row-Level Security (RLS) allows us to control access to rows in a database table based on the characteristics of the user executing a query. Here's an example of implementing RLS in SQL Server:

```
CREATE SCHEMA Security;
GO

CREATE FUNCTION Security.fn_securitypredicate(@OrgID AS int)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
    WHERE @OrgID = CAST(SESSION_CONTEXT(N'OrgID') AS int)
GO

CREATE SECURITY POLICY OrgFilter
ADD FILTER PREDICATE Security.fn_securitypredicate(OrgID)
ON dbo.Orders
WITH (STATE = ON);
```

With this policy in place, users will only see rows in the `Orders` table where the `OrgID` matches their session context.

Handling Concurrency in Multi-Tenant Systems

In multi-tenant systems, where a single instance of an application serves multiple customers (tenants), concurrency control becomes even more critical. Here are some strategies:

1. **Tenant Isolation**: Use separate schemas or databases for each tenant.

2. **Partitioning**: Partition data by tenant to improve performance and isolation.

3. **Tenant Context**: Always include tenant information in queries, possibly using Row-Level Security as shown above.

4. **Rate Limiting**: Implement rate limiting to prevent any single tenant from monopolizing resources.

Best Practices for Transaction and Concurrency Management

To wrap up, let’s summarize some key best practices:

1. **Use Transactions Judiciously**: Wrap related operations in transactions, but keep transactions as short as possible.

2. **Choose Appropriate Isolation Levels**: Understand the trade-offs between different isolation levels and choose based on your specific requirements.

3. **Implement Proper Error Handling**: Always include error handling and rollback mechanisms in your transaction logic.

4. **Avoid User Input During Transactions**: Collect all necessary user input before beginning a transaction to keep it short.

5. **Use Optimistic Concurrency When Appropriate**: For low-contention scenarios, optimistic concurrency can provide better performance.

6. **Implement Connection Pooling**: This can significantly improve performance in high-concurrency environments.

7. **Regular Monitoring and Tuning**: Continuously monitor your database performance and tune as necessary.

8. **Use Database-Specific Features**: Take advantage of features specific to your database system for optimizing concurrency (e.g., SQL Server’s READPAST hint).

9. **Consider NoSQL for Certain Scenarios**: For some high-concurrency, scale-out scenarios, NoSQL databases might be more appropriate.

10. **Educate Your Team**: Ensure all developers understand concurrency issues and how to handle them properly.

Conclusion

Ensuring data integrity in transactional systems requires a multi-faceted approach: 1. Utilize database constraints to enforce basic rules at the database level. 2. Implement triggers and stored procedures to encapsulate and consistently apply business logic. 3. Use appropriate isolation levels to balance consistency needs with performance requirements. 4. Implement optimistic or pessimistic concurrency control mechanisms based on your specific use case. 5. Handle distributed transactions carefully, using patterns like Two-Phase Commit or Sagas when necessary. 6. Consider eventual consistency models for systems that prioritize availability and partition tolerance. By combining these techniques and carefully designing our database and application logic, we can create robust, concurrent systems that maintain data integrity even under high load and complex operational scenarios. Managing transactions and concurrency in SQL databases is a complex but crucial aspect of building robust, high-performance applications. By understanding the fundamental concepts of ACID properties, isolation levels, and concurrency control techniques, and by applying advanced strategies like distributed transactions, optimistic concurrency with versioning, and row-level security, we can create database systems that maintain data integrity even under high levels of concurrent access. Remember, there’s no one-size-fits-all solution in concurrency control. The best approach depends on your specific use case, performance requirements, and the characteristics of your data and workload. Continuous learning, monitoring, and optimization are key to mastering this challenging but rewarding aspect of database management.

● **38% Overall Similarity**

Top sources found in the following databases:

- 31% Internet database
- 15% Publications database
- Crossref database
- Crossref Posted Content database
- 32% Submitted Works database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	coursehero.com Internet	6%
2	geeksforgeeks.org Internet	4%
3	ecomputernotes.com Internet	3%
4	vdocuments.site Internet	2%
5	Manuel S. Enverga University on 2022-12-19 Submitted works	1%
6	dev.to Internet	<1%
7	Centre for Distance and Online Education Galgotias University on 2024... Submitted works	<1%
8	bosecuttack.in Internet	<1%

9	University of North Texas on 2024-02-25 Submitted works	<1%
10	slideshare.net Internet	<1%
11	byjus.com Internet	<1%
12	University of Technology, Sydney on 2024-04-15 Submitted works	<1%
13	University of Technology, Sydney on 2024-04-15 Submitted works	<1%
14	Jagdish Chandra Patni. "A Comprehensive Study of SQL - Practice and ... Publication	<1%
15	en.wikipedia.org Internet	<1%
16	adaface.com Internet	<1%
17	Apprenticeship Learning Solutions Limited on 2024-07-23 Submitted works	<1%
18	devops.com Internet	<1%
19	University of North Texas on 2024-02-26 Submitted works	<1%
20	bluebirdinternational.com Internet	<1%

21	Academies Australasia Polytechnic on 2024-06-17 Submitted works	<1%
22	Aspen University on 2020-01-15 Submitted works	<1%
23	Clark University on 2023-03-18 Submitted works	<1%
24	Colorado Technical University on 2023-11-26 Submitted works	<1%
25	Maharishi University of Management on 2024-01-02 Submitted works	<1%
26	Wentworth Institute on 2024-05-31 Submitted works	<1%
27	huggingface.co Internet	<1%
28	arorapooja1982.blogspot.com Internet	<1%
29	K ENGLAND. "Transactions and Locking", Microsoft SQL Server 2005 P... Crossref	<1%
30	Oxford & Cherwell Valley College on 2023-01-06 Submitted works	<1%
31	docs.appdynamics.com Internet	<1%
32	hackernoon.com Internet	<1%

33	University of East London on 2023-12-18 Submitted works	<1%
34	vssut.ac.in Internet	<1%
35	"Pro ASP.NET 3.5 in C# 2008", Springer Science and Business Media L... Crossref	<1%
36	Melbourne Institute of Business and Technology on 2019-04-12 Submitted works	<1%
37	University of North Texas on 2024-02-26 Submitted works	<1%
38	Academies Australasia Polytechnic on 2024-02-11 Submitted works	<1%
39	Berlin School of Business and Innovation on 2023-01-20 Submitted works	<1%
40	Melbourne Institute of Technology on 2024-09-04 Submitted works	<1%
41	Pro Oracle SQL, 2013. Crossref	<1%
42	University of Northampton on 2024-06-02 Submitted works	<1%
43	Wentworth Institute on 2024-02-05 Submitted works	<1%
44	archive.org Internet	<1%

45	yumpu.com Internet	<1%
46	php-generator.org Internet	<1%
47	sql-programmers.com Internet	<1%
48	CSU, Chico on 2005-04-15 Submitted works	<1%
49	Colorado Technical University on 2024-07-31 Submitted works	<1%
50	UNICAF on 2024-06-02 Submitted works	<1%
51	Universidad TecMilenio on 2024-09-12 Submitted works	<1%
52	habr.com Internet	<1%
53	altitude365.com Internet	<1%
54	AlHussein Technical University on 2024-01-31 Submitted works	<1%
55	Ashfaque Ahmed, Bhanu Prasad. "Foundations of Software Engineerin... Publication	<1%
56	Asia Pacific Instutute of Information Technology on 2024-02-05 Submitted works	<1%

57	Berlin School of Business and Innovation on 2023-01-20 Submitted works	<1%
58	University of Arizona Global Campus (UAGC) on 2023-04-16 Submitted works	<1%
59	University of Oklahoma on 2022-03-09 Submitted works	<1%
60	Wentworth Institute on 2024-06-02 Submitted works	<1%
61	marketsplash.com Internet	<1%
62	De La Salle University on 2024-03-14 Submitted works	<1%
63	AlHussein Technical University on 2024-06-12 Submitted works	<1%
64	De La Salle University on 2024-03-14 Submitted works	<1%
65	kb.objectrocket.com Internet	<1%
66	pt.scribd.com Internet	<1%
67	tutorialdeep.com Internet	<1%
68	Berlin School of Business and Innovation on 2024-01-29 Submitted works	<1%

69	National College of Ireland on 2015-11-23 Submitted works	<1%
70	Oregon State University on 2023-02-28 Submitted works	<1%
71	dokumen.site Internet	<1%
72	dotnettutorials.net Internet	<1%
73	Athens University of Economics and Business on 2024-07-06 Submitted works	<1%
74	De La Salle University on 2024-03-13 Submitted works	<1%
75	Institute of Management Technology on 2015-02-04 Submitted works	<1%
76	University of North Texas on 2021-07-12 Submitted works	<1%
77	community.developers.refinitiv.com Internet	<1%
78	studytrails.com Internet	<1%
79	Ajou University Graduate School on 2024-05-22 Submitted works	<1%
80	Beginning Oracle Programming, 2002. Crossref	<1%

81	Coventry University on 2023-03-31 Submitted works	<1%
82	Karen Morton. "Transaction Processing", Pro Oracle SQL, 2010 Crossref	<1%
83	Kean University on 2024-04-05 Submitted works	<1%
84	University of North Texas on 2024-02-25 Submitted works	<1%
85	University of North Texas on 2024-02-26 Submitted works	<1%
86	University of Wales, Lampeter on 2024-08-20 Submitted works	<1%
87	cesarkallas.net Internet	<1%
88	sircrrengg.ac.in Internet	<1%
89	Chellammal Surianarayanan, Gopinath Ganapathy, Pethuru Raj. "Essent... Publication	<1%
90	Madan Mohan Malaviya University of Technology on 2017-05-03 Submitted works	<1%
91	Preston Zhang. "Practical Guide to Oracle SQL, T-SQL and MySQL", CR... Publication	<1%
92	Purdue University on 2022-12-06 Submitted works	<1%

93	TED Universitesi on 2022-01-18 Submitted works	<1%
94	Teamlease Skill University on 2024-09-26 Submitted works	<1%
95	University of Lay Adventists of Kigali on 2023-09-01 Submitted works	<1%
96	University of New England on 2024-05-28 Submitted works	<1%
97	University of Stirling on 2023-04-25 Submitted works	<1%
98	dokumen.pub Internet	<1%
99	ebin.pub Internet	<1%
100	interviewzilla.com Internet	<1%
101	makingdatameaningful.com Internet	<1%
102	publications.ics.forth.gr Internet	<1%
103	vdoc.pub Internet	<1%
104	recital.com Internet	<1%

105	techtarget.com Internet	<1%
106	Universidad Anahuac México Sur on 2024-04-06 Submitted works	<1%
107	University Politehnica of Bucharest on 2024-07-04 Submitted works	<1%
108	Asia Pacific University College of Technology and Innovation (UCTI) on... Submitted works	<1%
109	CSU, San Jose State University on 2006-05-22 Submitted works	<1%
110	Saeed K. Rahimi, Frank S. Haug. "Distributed Database Management S... Crossref	<1%
111	South Australian Institute of Business and Technology on 2024-09-11 Submitted works	<1%
112	Southampton Solent University on 2012-05-15 Submitted works	<1%
113	Temple University on 2020-09-13 Submitted works	<1%
114	University of Leicester on 2023-01-13 Submitted works	<1%
115	Visvesvaraya Technological University, Belagavi on 2023-01-05 Submitted works	<1%
116	Wentworth Institute on 2024-05-30 Submitted works	<1%