Principles of Database Systems(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Final Review

Mengxuan Wu

1 Lecture 1

1.1 Definition of Database

A **relational database** is based on the relational model of data. It organizes data into one or more tables (or "relations") of columns and rows. Rows are also called records or tuples. Columns are also called attributes or fields. *(The word "relational" here indicates the data in one row are related.)*

1.2 Keys

The order of the columns is not important. What is important is that the data in one row are related, and correspond to the header of the column.

The database don't want to have duplicate rows, since it is a waste of space and will give us incorrect results when we query the database. If we want to have no duplicate rows, we need to identify what makes a row unique, which is called a **key**. There might be different keys available in a table, but we need to choose one of them to be the **primary key**.

1.3 Normalization

1.3.1 ER Diagram

In a database model, an **entity** is a something that has a life of its own. **Relation** connects two or more entities, it has no life of its own.

An **ER diagram** is a graphical representation of entities and their relationships to each other. The **cardinality** of a relationship is the number of entities that can be involved in a relationship.

- (1:n): one to many
- (0:n): zero to many
- (m,n): many to many

1.3.2 First Normal Form (1NF)

The first normal form (1NF) is that each column should contain only one value. (atomic values) A violation example:

Student ID	Name
1	John Smith

Name should be split into first name and last name.

1.3.3 Second Normal Form (2NF)

The second normal form (2NF) is that each column should be dependent on the entire primary key, not just one part of it. A violation example:

Student ID	Course ID	Course Name
1	1	Database Systems

The course name is dependent on course ID, not student ID.

1.3.4 Third Normal Form (3NF)

The third normal form (3NF) is that each column should be dependent on the primary key, not on another column. A violation example:

Student ID	Course ID	Course Name	Course Instructor
1	1	Database Systems	John Smith

The course instructor is dependent on course name. It has nothing to do with primary key.

2 Lecture 2

2.1 SQL

A good database language should be good at dealing with data and containers.

2.1.1 Data Definition Language (DDL)

The data definition language (DDL) deals with tables or other database objects.

- **CREATE** - create a new table, a view of a table, or other object in the database
- **ALTER** - alter an existing database object, such as a table
- **DROP** - delete objects from the database

2.1.2 Data Manipulation Language (DML)

The data manipulation language (DML) deals with data manipulation.

- **SELECT** - retrieve data from a database
- **INSERT** - insert data into a table
- **UPDATE** - updates existing data within a table
- **DELETE** - deletes all records from a table, the space for the records remain

SQL language is actually quite "relax" compared to other programming languages. If we are not rigorous enough, we might get a wrong result without warning. The only enforced key property is that all rows are unique. (But not for rows in query result)

2.2 Create Table

The standard syntax for creating a table is:

```
CREATE TABLE <table name> (
<column name> <data type> <constraints>,
<column name> <data type> <constraints>,
...
);
```

Identifiers are **case-insensitive** in SQL. **"tablename"** and **"TABLENAME"** are the same. In PostgreSQL, identifiers must begin with a letter or an underscore, and can contain letters, digits, and underscores. However, if you quote the identifier with double quotes, you can use any characters you want except double quotes, and will be treated as case-sensitive. When doing so, the word is always treated as an identifier, and thus you can use reserved words like "UPDATE" as identifiers. As a result, we often use underscore to separate words in identifiers, instead of camel case. *(UPDATE without quotes is not an identifier, it is a keyword.)*

2.2.1 Data Types

Text data types:

- **CHAR(n)** - fixed-length character string, blank padded
- **VARCHAR(n)** - variable-length character string, no padding
- **TEXT** - variable-length character string, no limit (sometimes as CLOB)

Numeric data types:

- **INTEGER** - signed four-byte integer
- **FLOAT** - floating-point number (will be converted to double precision or real)
- **REAL** - single precision floating-point number (4 bytes)
- **DOUBLE PRECISION** - double precision floating-point number (8 bytes)

- **NUMERIC(precision, scale)** - exact numeric of selected precision and scale (scale is the number of digits after the decimal point, precision is the total number of digits)
- **NUMERIC(precision)** - exact numeric of selected precision and 0 scale, this will force the number to be an integer

Date and time data types:

- **DATE** - calendar date (year, month, day)
- **TIME** - time of day (no time zone)
- **TIMESTAMP** - date and time (no time zone)
- **TIMESTAMP WITH TIME ZONE** - date and time, including time zone
- **INTERVAL** - time interval

Binary data types:

- **RAW(n)** - fixed-length binary string
- **VARBINARY(n)** - variable-length binary string
- **BLOB** - variable-length binary string, no limit
- **BYTEA** - variable-length binary string, no limit

2.2.2 Constraints

Constraints are used to specify rules for data in the table. They are usually used to limit the type of data that can go into a table. Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table (only primary key or unique column can be referenced)
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly
- **AUTO INCREMENT** - Automatically increments the value of the column by 1 each time a new row is inserted
- **SERIAL** - Automatically increments the value of the column by 1 each time a new row is inserted
- **CONSTRAINT** - Used to define a constraint on a column

2.2.3 Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements. Single-line comments start with –. However, comment written in this way will not be stored in the database. To store comments in the database, we need to use the COMMENT command.

```
COMMENT ON TABLE <table name> IS 'comment';
COMMENT ON COLUMN <table name>.<column name> IS 'comment';
```

2.3 Insert

The standard syntax for inserting a row into a table is:

```
INSERT INTO <table name> (<column name>, <column name>, ...)
VALUES (<value>, <value>, ...);
```

If the column names are omitted, database will try to insert the values into the columns in the order they were defined in the table.

3 Lecture 3

3.1 Select

The standard syntax for selecting data from a table is:

```
SELECT <column name>, <column name>, ...
FROM <table name>;
```

3.1.1 WHERE Clause

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified condition.

```
SELECT <column name>, <column name>, ...
FROM <table name>
WHERE <condition>;
```

When using AND and OR in the WHERE clause, we need to use parentheses to make sure the database knows what to evaluate first. The default precedence of AND and OR is AND first, then OR.

Sometimes using the IN operator is more convenient than using multiple OR conditions. BETWEEN is also a convenient way to check if a value is within a range. (inclusive)

LIKE is a special operator used to match text patterns. It has two wildcards:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

The DBMS is case-sensitive by default, but we can use ILIKE to make it case-insensitive. Also, it is not a good idea to add UPPER or LOWER to the column name, since the performance will be bad.

3.2 Date

When comparing a data column with a string, the database will try to convert the string to a date. It is very wrong to compare a date with a datetime, since the database will try to convert the date to a datetime and the time part will be 00:00:00. When doing arithmetic operations on dates, you should use the INTERVAL type.

3.3 NULL

NULL is a special value that represents missing or unknown information. When comparing a column with NULL, you should use IS NULL or IS NOT NULL.

NULL is troublesome when doing arithmetic operations. The result of any arithmetic operation with NULL is NULL. When doing logic operations, NULL sometimes behaves like FALSE, sometimes behaves like TRUE. It may be short circuited like FALSE AND NULL.

When NULL appears in the IN or NOT IN clause, we need to be careful. A NULL in the IN clause can be ignored, the condition will work as expected. Because IN will be converted to condition1 OR condition2, and a FALSE chained by OR can be omitted. But a NULL in the NOT IN clause will lead to bad result. The condition is always false, since the NOT IN will be converted to condition1 AND condition2, and a FALSE chained by AND will always return FALSE.

3.4 Functions

Here are some useful functions:

- || - concatenate two strings
- **cast** - convert a value to another type (cast('1' as integer))
- **date_part** - extract a part of a date (date_part('year', date))
- **round** - round a number to a specified number of decimal places (round(number, decimal places))
- **trunc** - truncate a number to a specified number of decimal places (trunc(number, decimal places))
- **floor** - round a number down to the nearest integer
- **ceil** - round a number up to the nearest integer
- **upper** - convert a string to uppercase
- **lower** - convert a string to lowercase
- **substr** - extract a substring from a string (substring(string, start, length))
- **trim** - remove leading and trailing spaces from a string (trim(string))
- **replace** - replace all occurrences of a substring in a string with another substring (replace(string, substring, replacement))

- **coalesce** - return the first non-null value in a list (coallesce(value1, value2, ...))

For performance reasons, we can use functions on the selected columns or on the right side of the WHERE clause, but not on the left side of the WHERE clause.

3.5 CASE

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement).

Here is the first syntax of the CASE statement:

```
CASE <column name>
WHEN condition1 THEN result1
WHEN condition2 THEN result2
...
ELSE result
END AS <alias>
```

NULL cannot be tested in this way, since NULL is not equal to anything, not even itself. ("WHEN NULL" will always be false)

Here is the second syntax of the CASE statement:

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
...
ELSE result
END AS <alias>
```

We can compare NULL with IS NULL or IS NOT NULL in this way, by writing the condition as "WHEN column IS NULL".

4 Lecture 4

4.1 Distinct

The DISTINCT keyword is used to return only distinct (different) values. If multiple columns are specified, the DISTINCT keyword will evaluate the duplicate based on the combination of values of these columns.

4.2 Aggregate Functions

We can use aggregate functions to perform calculations on a set of values and return a single value. This would require a GROUP BY clause. Every column in the SELECT clause must either be in the GROUP BY clause or be an aggregate function.

A HAVING clause is used to filter values after they have been grouped. It may not need to be the same as the aggregate function in the SELECT clause. For example, we can use HAVING COUNT(*) > 1 to find duplicate rows.

Sometimes there are two to write the same query, one with HAVING and one with WHERE. The WHERE clause is applied before the GROUP BY clause, while the HAVING clause is applied after the GROUP BY clause. We encourage using WHERE first,

since the aggregate function is expensive, and we want it to be applied to as few rows as possible.

Aggregate functions ignore NULL values, even if we use DISTINCT. However, if we write COUNT(*), this will count all rows, including those with NULL values.

4.3 Inner Join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. If the two columns have the same name, we need to specify them with the table name.

JOIN can work on subqueries. Chained JOINs are evaluated from left to right.

5 Lecture 5

5.1 Outer Join

A left outer join will return all rows in the left table, and the matching rows in the right table. If there is no match, the right side will contain NULL.

A full outer join will return all rows in both tables. If there is no match, the side without a match will contain NULL.

When dealing with outer joins, we need to be careful with the WHERE clause. If we put a condition on the right table, it will be applied after the join, and the rows with NULL will be filtered out. This makes the outer join behave like an inner join.

5.2 Set Operators

The UNION operator is used to combine the result-set of two or more SELECT statements. This eliminates duplicate rows between the various SELECT statements. If we want to keep the duplicate rows, we can use UNION ALL.

The INTERSECT operator is used to return the records that two SELECT statements have in common. The INTERSECT operator is equivalent to the INNER JOIN clause.

The EXCEPT operator is used to return all the records in the first SELECT statement that are not in the second SELECT statement. The EXCEPT operator is equivalent to the LEFT OUTER JOIN clause where the second table's values are NULL.

5.3 Subqueries

A **correlated subquery** is a subquery that uses values from the outer query. This means for each row processed by the outer query, the inner query is executed as well.

6 Lecture 6

6.1 Order By

The ORDER BY keyword is used to sort the result-set in ascending or descending order. The default is ascending order.

How the data is sorted is determined by collation. This can be specified when creating a table like:

```
CREATE TABLE <table name> (
<column name> <data type> COLLATE <collation>,
<column name> <data type> COLLATE <collation>,
...
);
```

If sorting by alphabet doesn't work, we can specify an order using a CASE statement, like:

```
ORDER BY CASE WHEN <column name> = 'value1' THEN 1
WHEN <column name> = 'value2' THEN 2
...
ELSE 3
END
```

Then we sort by the number we assigned to each value.

6.2 Limit and Offset

The LIMIT keyword is used to limit the number of rows returned in a query result. The OFFSET keyword is used to offset the starting row number.

6.3 Window Functions

A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. However, unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row. Instead, the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

There are two types of window functions:

- **Ranking functions** - assign a rank to each row based on the value of the column
- **Aggregate functions** - calculate an aggregate value based on a group of rows

The ranking functions behave differently and should be used with care when there are duplicate values. Take 1, 2, 2, 3, 4 as an example. In **row_number**, the rank will be assigned in the order of the rows, so the result will be 1, 2, 3, 4, 5. In **rank**, the rank will be assigned in the order of the values, so the result will be 1, 2, 2, 4, 5. In **dense_rank**, the rank will be assigned in the order of the values, but the rank will not have gaps, so the result will be 1, 2, 2, 3, 4.

7 Lecture 7

7.1 Fuzzy Search

Fuzzy search is a technique for finding strings that match a pattern approximately (rather than exactly).

What usually do: isolate each word, and rank by the number of words matched.

7.2 Transaction

Transaction is a sequence of operations performed as a single logical unit of work. It starts with a BEGIN statement and ends with a COMMIT or ROLLBACK statement.

Some DBMS has autocommit enabled by default, such as JDBC.

Difference also appears about DDL. In PostgreSQL, DDL can be used in a transaction and can be rolled back.

7.3 Insert With Sequence

A sequence is a database object that generates numbers in sequence. The standard syntax for creating and using a sequence in PostgreSQL is:

```
CREATE SEQUENCE <sequence name>;
SELECT NEXTVAL('<sequence name>'); -- return the next value generated by the sequence
SELECT CURRVAL('<sequence name>'); -- return the last value generated by the sequence
SELECT LASTVAL(); -- return the last value generated by any sequence
```

Be careful that the sequence name should be quoted with single quotes in the SELECT statement.

If you specify a column as serial when creating a table, PostgreSQL will automatically create a sequence for you.

7.4 Insert with COPY

The COPY command copies data between a file and a table. The file can be on the server or on the client. It is also called a bulk load operation.

8 Lecture 8

8.1 Update and Delete

The standard syntax for updating a row in a table is:

```
UPDATE <table name>
SET <column name> = <value>, <column name> = <value>, ...
WHERE <condition>;
```

The standard syntax for deleting a row in a table is:

```
DELETE FROM <table name>
WHERE <condition>;
```

A delete operation is actually a logical delete. It can be slow, but it can be rolled back. If we want to delete a row permanently, we can use TRUNCATE.

9 Lecture 9

9.1 Procedure

A procedure is a subprogram that performs a specific action. It can be good for performance, since it can reduce network traffic. It is also good for security, since it can prevent SQL injection or direct access to tables.

9.2 Trigger

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server.

It can be used to modify input on the fly (with before trigger), or check complex constraints (with before trigger), or manage data redundancy (with after trigger).

9.3 Auditing

Trigger is the last line of defense. Do not use trigger to fix bad design, since the trigger is inefficient and complicated.

10 Lecture 10

10.1 Index

An index will be created automatically for the primary key or unique constraint. However, it is encouraged to declare unique constraint instead of using the unique index.

It is not a good idea to have too many indexes. First, it will take up a lot of space. Second, it will slow down the insert, update and delete operations.

For performance reasons, there are criteria: often used and selective. Often used means the column is used in the WHERE clause. Selective means the column has a lot of different values.

There are certain types of queries that cannot use indexes. For example, queries with LIKE that begins with a wildcard. Or a query with a function on the column.

On way to improve performance is to add an extra column to the table, and use it to index.(although this violates the normalization rule) Another is to create index on the function of the column.(the function must be deterministic)

11 Lecture 11

11.1 View

A view is a virtual table based on the result-set of an SQL statement. It contains rows and columns, just like a real table. The columns are fixed once the view is created, and cannot be changed.(Even if you create a view with SELECT *)

A view can be used to hide complex queries. This might not be a good idea in some scenarios, such as you want only part of the data, but view will compute all data and then filter it. Also, it is hard to improve query performance since you have no idea what the view is doing.

The positive side is that view can be used to hide sensitive data.

You may create view on other views, but this is dreadful for performance.

The standard syntax for creating a view is:

```
CREATE VIEW <view name> AS
SELECT <column name>, <column name>, ...
FROM <table name>
WHERE <condition>;
```

11.2 Deeper Into View

Update a view is possible, but it is not a good idea. If some data comes form multiple tables, the DBMS will throw an error.

11.3 Security

The DBMS provides a security mechanism to control access to data. We can grant or revoke privileges on tables and views to users and roles.

Be careful when updating a view, this will lose all previous privileges on the view. You can control update of a view with check option, or by using triggers.

12 Lecture 12

12.1 Catalog

There is one catalog per database. It stores metadata, such as table names, column names, data types, etc. These tables are called system tables, you cannot modify them directly.

13 Lecture 13

13.1 Query Optimization

When we write a query, DBMS will go through the following steps:

- **Parsing** - check syntax, check permissions, check if the table exists, etc.
- **Rewriting** - rewrite the query to an equivalent query
- **Planning** - find the best way to execute the query
- **Execution** - execute the query

One way to accelerate this is to store system tables in memory.(for parsing check and rewriting) And most DBMS will keep the parsed query in memory, so the next time you run the same query, it will be faster.

When query planning, DBMS evaluates:

- Logical transform
- indexes
- Hardware performance
- System load
- Setting

13.2 Scaling

There are two ways to scale a database: scaling up and scaling out. Scaling up means to replace the hardware with a more powerful one. Scaling out means to add more hardware.

However, with relative capacity formula, scaling out will have diminishing returns.

The major issue with scaling out is that it is hard to keep the data consistent. We use two-phase commit to solve this problem. That is, we ask every server whether they are ready to commit, and then commit if everyone is ready, if not we abort. However, due to latency, this is may have a bad performance.

14 Lecture 14

14.1 NoSQL

NoSQL is a non-relational database management system. It means more than SQL, not no SQL. It is often used for big data and real-time web applications. It handles unstructured data and can work with no predefined schema.

Advantages of NoSQL:

- Handles big data
- No predefined schema
- Handles unstructured data
- Cheaper to manage
- Easy to scale

Advantages of SQL:

- Better fir relational data
- Normalization
- Well known language
- Data integrity
- ACID compliance