Databases

Seminar 8
Transactions

Theoretical background

A transaction is an object that groups a sequence of operations that must be performed as a whole.

Ensures the transition of the database from one integral state to another.

A transaction is an object that groups a sequence of operations that must be performed as a whole.

Theoretical background

Ensures the transition of the database from one integral state to another.

- run a query to the product table and check the availability of goods in stock;
- add an order to the invoice table;
- update the goods table by subtracting the ordered quantity of goods from the quantity of goods available;
- update the sales table by adding the cost of the order to the sales volume of the employee who accepted the order;
- update the table of offices by adding the cost of the order to the sales volume of the office in which this employee works.

Theoretical background

Transactions ensure the integrity of the database under the conditions:

- Parallel data processing
- Physical disk failures
- Emergency power failure
- And others

Theoretical background

Transactions have 4 characteristics that satisfy the ACID paradigm:

1. Atomic
2. Consistent
3. Isolated
4. Durable

Atomicity

- A transaction must be an atomic (indivisible) unit of work;
- Either all operations included in the transaction must be performed, or none of them;
- Therefore, if it is impossible to perform all operations, all the changes made must be canceled.:
  - Commit – making a transaction
  - Rollback – cancellation of the transaction

Consistency

- Upon completion of the transaction, all data should remain in a consistent state
- When executing a transaction, you must follow all the rules of a relational DBMS:
  - Checks for compliance with restrictions (domains, uniqueness indexes, foreign keys, checks, rules, etc.)
  - Updating indexes;
  - Executing Triggers
  - And others

Isolation

- Data changes performed within a transaction must be isolated from all changes performed in other transactions until the transaction is committed.;
- There are different levels of isolation – to achieve a compromise between the degree of parallelization of work with the database and the rigor of the principle of consistency:
  - The higher the isolation level, the higher the degree of data consistency;
  - The higher the isolation level, the lower the degree of parallelization and the lower the degree of data availability.

Standard classification of problems with isolation levels:

- P1: dirty read – "dirty" reading
- P2: non-repeatable read – non-repeatable read
- P3: phantom read – phantom reading

P1 (dirty read)

1. Transaction T1 makes changes to a number of tables.
2. T2 reads this row after making changes to T1, but before committing T1.
3. If T1 is canceled, then the data read by T2 will be incorrect.

```
-- T1

UPDATE Users

SET AGE = 21

WHERE Id = 1;


ROLLBACK;
```

```
-- T2
SELECT Age
FROM Users
WHERE Id = 1;
```

P2 (non-repeatable read)

1. Transaction T1 reads a series.
2. Transaction T2 then makes changes to this row or deletes it.
3. If T1 then counts the same row again, it will get a new result compared to the first reading.

```
-- T1

SELECT *

FROM Users

WHERE Id = 1;
```

```
-- T2
UPDATE Users
SET AGE = 21
WHERE Id = 1;

COMMIT;
```

P3 (phantom)

1. Transaction T1 reads a set of rows N satisfying some condition.
2. After that, T2 executes SQL queries that create new rows that satisfy this condition.
3. If T1 repeats the query with the same condition, it will get a different set of rows.

```
-- T1
SELECT *
FROM Users
WHERE Age BETWEEN 10 AND 30;
```

```
-- T2
INSERT INTO
Users (Name, Age)
VALUES ('Bob', 27);

COMMIT;
```

Durability

- If a transaction has been completed, its result should be saved in the system, despite the system failure.;
- If the transaction has not been completed, its result may be completely canceled following a system failure.

Transaction Control Language

BEGIN TRANSACTION transaction_mode [, ...]

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }

READ WRITE | READ ONLY

[NOT] DEFERRABLE

Transaction Control Language

BEGIN / START

COMMIT

ROLLBACK

Transaction Control Language

SAVEPOINT name

ROLLBACK TO SAVEPOINT name

RELEASE SAVEPOINT name

Transaction boundaries

```
BEGIN;

INSERT ...;

UPDATE ...;

DELETE ...;

COMMIT;
```

```
BEGIN;

INSERT ...;

UPDATE ...;

DELETE ...;

ROLLBACK;
```

ANSI SQL Transaction Isolation Levels

In an ideal world, parallel transactions are isolated (they get along perfectly with each other in the database), but in the real world this is impossible. The characteristic of the correspondence of the base to the

Isolation level	Dirty reading	Non-reproducible reading	Phantom Reading
0. Read Uncommitted	perhaps	perhaps	perhaps
1. Read Committed	-	perhaps	perhaps
2. Repeatable Read	-	-	perhaps
3. Serializable	-	-	-

ANSI SQL Transaction Isolation Levels

READ UNCOMMITTED: Helps to deal with lost updates when the same data block is changed by multiple transactions.

READ COMMITTED: helps to deal with reading "dirty" data changed later by a rolled-back transaction

REPEATABLE READ: helps to deal with changes in data when the same block of data is read repeatedly within the same transaction

SERIALIZABLE: similar to REPEATABLE READ, but applicable to INSERT, not UPDATE

PostgreSQL access control.

Role – set of DB users

- Can own objects in the database
- May have certain access to some database objects without being their owner
- Can grant access to some objects in the database to other roles
- Can grant membership in the role of another role

Creating a user

```
CREATE USER == CREATE ROLE

CREATE USER name [[WITH] option [...]]
where option can be:
SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| BYPASSRLS | NOBYPASSRLS
| CONNECTION LIMIT connlimit
| [ENCRYPTED] PASSWORD 'password' | PASSWORD NULL
| VALID UNTIL 'timestamp'
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN role_name [, ...]
| USER role_name [, ...]
| SYSID uid
```

```
CREATE ROLE jonathan LOGIN;

CREATE USER davide
WITH PASSWORD 'jw8s0F4';

CREATE ROLE Miriam
WITH LOGIN PASSWORD 'jw8s0F4'
VALID UNTIL '2005-01-01';
```

Data Control Language (DCL)

- Allows you to configure access to objects
- Supports 2 types of actions:
  - GRANT – granting access to an object
  - REVOKE – revoke access to the object

Rights that can be granted to an object:

- SELECT
- INSERT
- UPDATE
- DELETE
- TRUNCATE
- REFERENCESE
- TRIGGER
- CREATE
- CONNECT
- TEMPORARY
- EXECUTE
- USAGE

GRANT ON TABLE

```
GRANT {{SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
    REFERENCES | TRIGGER}
[, ...] | ALL [PRIVILEGES]}
ON {[ TABLE] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...]}
TO role_specification [, ...] [WITH GRANT OPTION]
```

REVOKE ON TABLE

```
REVOKE [GRANT OPTION FOR]
    {{SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
        REFERENCES | TRIGGER}
[, ...] | ALL [PRIVILEGES]}
ON {[TABLE] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...]}
FROM {[GROUP] role_name | PUBLIC} [, ...]
[CASCADE | RESTRICT]
```

GRANT ALL PRIVILEGES ON kinds TO manuel;
REVOKE ALL PRIVILEGES ON kinds FROM manuel;

GRANT SELECT ON kinds TO manuel WITH GRANT OPTION;

Note: If we want to take the grant option from manuel, then we need to use CASCADE to take the rights from everyone to whom he granted the rights. Otherwise, if you try to revoke the rights from manuel, the request will fall with an error.

Usage examples

CREATE USER davide WITH PASSWORD 'jw8s0F4';

GRANT CONNECT ON DATABASE db_prod TO davide;

GRANT USAGE ON SCHEMA my_schema TO davide;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA my_schema

TO davide;