Advanced Features

NGUYEN Hong Phuong
phuongnh@soict.hut.edu.vn

1

Contents

2

1. Views

- Create a view over the query, which gives a name to the query that you can refer to like an ordinary table:

```
CREATE VIEW myview AS
SELECT city, temp_lo, temp_hi, prcp, date, location
FROM weather, cities
WHERE city = name;

SELECT * FROM myview;
```

- Views allow you to encapsulate the details of the structure of your tables, which might change as your application evolves, behind consistent interfaces.

3

2. Foreign Keys

- Recall the weather and cities tables from previous chapter. Consider the following problem: You want to make sure that no one can insert rows in the weather table that do not have a matching entry in the cities table. This is called maintaining the *referential integrity* of your data.

```
CREATE TABLE cities (
city varchar(80) primary key,
location point
);
```

```
CREATE TABLE weather (
city varchar(80) references cities(city),
temp_lo int,
temp_hi int,
prcp real,
date date
);
```

4

3. Transactions

- A transaction comprises a unit of work performed within a DBMS (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions.
- It bundles multiple steps into a single, all-or-nothing operation
- The intermediate states between the steps are not visible to other concurrent transactions
- If some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.

5

An example

- Consider a bank database that contains balances for various customer accounts, as well as total deposit balances for branches.
- Suppose that we want to record a payment of $100.00 from Alice's account to Bob's account

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE
    name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE
    name = 'Bob');
```

6

An example (cont.)

- Assure that either all these updates happen, or none of them happen.
  - Bob received $100.00 that was not debited from Alice?
  - Alice was debited without Bob being credited?
- If something goes wrong partway through the operation, none of the steps executed so far will take effect.
- Grouping the updates into a transaction gives us this guarantee

7

Properties of a transaction - ACID

- Atomicity
  - A series of DB operations either all occur or nothing occurs
  - Prevent updates to the DB occurring only partially, which can cause greater problems than rejecting the whole series outright.
  - In other words, atomicy means indivisibility and irreducibility
- Consistency
  - Data is consistent after the transaction

8

Properties of a transaction - ACID

- Isolation
  - Define how/when the changes made by one operation become invisible to other concurrent operations
- Durability
  - Guarantees that transactions that have committed will survive permanently.
  - For example, if a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.

9

- In PostgreSQL, a transaction is set up by surrounding the SQL commands of the transaction with BEGIN and COMMIT commands

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
WHERE name = 'Alice';
.............
COMMIT;
```

10

- We do not want to commit (perhaps we just noticed that Alice's balance went negative), we can issue the command ROLLBACK instead of COMMIT, and all our updates so far will be canceled.
- PostgreSQL actually treats every SQL statement as being executed within a transaction.
- If you do not issue a BEGIN command, then each individual statement has an implicit BEGIN and (if successful) COMMIT wrapped around it.

11

- A group of statements surrounded by BEGIN and COMMIT is sometimes called a transaction block.

12

Save points

- Using savepoints to control the statements in a transaction.
- Savepoints allow you to selectively discard parts of the transaction, while committing the rest.
- After defining a savepoint with SAVEPOINT, you can roll back to the savepoint with ROLLBACK TO.
- All the transaction's database changes between defining the savepoint and rolling back to it are discarded, but changes earlier than the savepoint are kept.

13

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
WHERE name = 'Wally';
COMMIT;
```

14

4. Inheritance

- Inheritance is a concept from object-oriented databases. It opens up interesting new possibilities of database design.

```
CREATE TABLE cities(
    name text,
    population real,
    altitude int);
CREATE TABLE capitals(
    state char(2)
) INHERITS (cities);
```

15

- In this case, a row of capitals inherits all columns (name, population, and altitude) from its parent, cities.

```
Insert into cities values('Las
    Vergas',4.2,2174), ('Mariposa',2.1,1953);
Insert into capitals values('Madison',5.6,845,
    'CA');

SELECT *
FROM cities
WHERE altitude > 500;
```

16

```
SELECT *
FROM cities

SELECT name, altitude
FROM ONLY cities
WHERE altitude > 500;
```

- In PostgreSQL, a table can inherit from zero or more other tables.

17