Бази даних

Лекція 12

Тематика лекції

- ALTER
- Міграції

ALTER

ALTER - інструмент DDL для оновлення вже створених структур, таких як таблиці, колонки, індекси і т.д.

Приклад:

ALTER TABLE users ADD COLUMN age INTEGER;

Як працює додавання колонки

1. **Отримується ACCESS EXCLUSIVE лок на табличку, що модифікується.**
2. **Виконується оновлення метаданих таблиці - дані в таблиці не змінюються.**
3. **Лок “відкривається”.**

Всі записи та читання з таблиці блокуються протягом всього процесу.
Початок процесу очікує закінчення всіх поточних операцій.
Вже існуючі дані в таблиці не змінюються.
Операція швидка (зазвичай кілька мілісекунд).

Як працює додавання constraint до колонки

1. **Отримується ACCESS EXCLUSIVE лок на табличку, що модифікується.**
2. **Виконується читання всіх даних в табличці та виконується перевірка того, що дані відповідають новому обмеженню.**
3. **Виконується оновлення метаданих таблиці - дані в таблиці не змінюються.**
4. **Лок “відкривається”.**

Всі записи та читання з таблиці блокуються протягом всього процесу.
Початок процесу очікує закінчення всіх поточних операцій.
Вже існуючі дані в таблиці не змінюються.
Відбувається читання ВСІХ даних в таблиці.

Безпечні (швидкі) ALTER

- Додавання колонки
- Видалення колонки
- Переіменування колонки
- Додавання значення за замовчуванням для колонки
- CREATE INDEX CONCURRENTLY

Небезпечні (повільні) ALTER

- **Додавання CHECK / NOT NULL для існуючої колонки - провокує читання і перевірку всіх даних таблички.**
- **Зміна типу колонки - провокує переписування даних таблички.**
- **CREATE INDEX - блокує записи в табличку протягом створення індексу.**

Міграції

Міграції це версіоновані послідовні зміни структури бази даних.

Основні властивості:

- **Чітка послідовність - кожна наступна міграція будується на попередніх.**
- **Версіонування - кожна міграція має послідовну версію.**
- **Повторюваність - після виконання заданого набору міграцій, база даних завжди приходить до однієї структури.**
- **Відстежуваність - база даних зберігає інформацію про усі виконані міграції.**

Міграції

Складові частини:

- **Версіонована послідовність міграцій (sql скріпти чи інструменти фреймворків).**
- **Таблиця в базі даних, що містить інформацію про виконані міграції.**
- **Інструмент виконання міграцій (Flyway/Liquibase/Prisma/etc).**

Що можуть містити скріпти міграцій

- Створення / видалення / зміна таблиць
- Додавання / видалення / зміна колонок
- Додавання / видалення / оновлення індексів
- Інші зміни схеми бази даних
- Додавання / видалення / зміна даних в таблицях

Базовий алгоритм системи міграцій

1. **Порівнюється список файлів міграцій зі списком виконаних міграцій.**
2. **Для кожної з ще не виконаних міграцій по черзі:**
    a. **Обчислюється чек-сума міграції.**
    b. **Починається транзакція.**
    c. **Виконується скріпт міграції.**
    d. **В таблицю версій додається версія та чек-сума.**
    e. **Відбувається комміт транзакції.**

Структура таблиці версій

```
CREATE TABLE IF NOT EXISTS schema_migrations (
    version VARCHAR(50) PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    checksum VARCHAR(64),
    execution_time_ms INTEGER,
    success BOOLEAN DEFAULT TRUE
);
```

Приклад файлу міграції

Ім’я файлу:
V015_add_user_age_column.sql

Контент файлу:
ALTER TABLE users ADD COLUMN age INTEGER;

Тут версія міграції 015 та назва міграції “add user age column”.

Чек-сума

Чек-сума це хеш всього контенту файлу міграції.

Навіщо це потрібно: для перевірки того, що файл міграції не був змінений з часу її виконання.

Чому це важливо: всі файли міграцій повинні бути повністю незмінними, адже якщо міграція була виконана - очікується, що це фіксований стан бази даних, який можна повністю повторити в будь-який момент за допомогою виконання всіх попередніх та даної міграцій. Якщо ж файли міграцій змінюються - цього досягнути не вийде.

Успішність міграції

Якщо стається будь-яка помилка - міграція позначається в таблиці версій як неуспішна. Після цього виконання усіх наступних міграцій є неможливим.

Для усунення проблеми потрібне ручне втручання:

- **Зрозуміти що пішло не так (прочитати логи виконання міграції)**
- **Виправити проблему, що виникла**
- **Видалити з таблиці міграцій інформацію про неуспішну міграцію**
- **Повторити міграцію**

Безпечність міграцій

Безпечність міграцій

Основні проблеми, які виникають при міграціях:

- **Блокування таблиці на довгий час - жоден процес не зможе читати та/або писати дані в таблицю, що перебуває в процесі модифікації.**
- **Міграція схеми та використання нової версії програмного забезпечення, що використовує нову схему ніколи не відбувається одночасно.**
- **Файли міграцій незмінні - якщо допущена помилка в скрипті міграції, її не вийде виправити просто змінивши файл міграції.**

Безпечні (швидкі) ALTER

- Додавання колонки (може залежати від версії PostgreSQL)
- Видалення колонки
- Переіменування колонки
- Додавання значення за замовчуванням для колонки
- CREATE INDEX CONCURRENTLY

Небезпечні (повільні) ALTER

- **Додавання CHECK / NOT NULL для існуючої колонки - провокує читання і перевірку всіх даних таблички.**
- **Зміна типу колонки - провокує переписування даних таблички.**
- **CREATE INDEX - блокує записи в табличку протягом створення індексу.**

Багатокроковий реліз

Ситуація: таблиця “orders” нормалізується та розбивається на кілька таблиць. Існує сервіс, який активно записує та читає дані в таблиці “orders”.

Багатокроковий реліз

Ситуація: таблиця “orders” нормалізується та розбивається на кілька таблиць. Існує сервіс, який активно записує та читає дані в таблиці “orders”.

Проблеми:

- **Якщо нова версія програмного забезпечення залежить на ще не створені таблички - ми отримаємо помилку в runtime.**
- **Якщо нова версія програмного забезпечення прочитає дані з нової таблички, але вони ще не мігровані - ми отримаємо неконсистентні дані.**
- **Якщо стара версія програмного забезпечення спробує прочитати дані з уже зміненої таблиці - ми отримаємо помилку в runtime.**

Багатокроковий реліз

Рішення: процес релізу, що складається з кількох кроків.

1. **Відбувається перша міграція - додаються нові таблиці, проте існуюча не змінюється.**
2. **Відбувається реліз версії ПЗ, що записує дані в старому форматі ТА в нові таблиці.**
3. **Відбувається друга міграція - дані переносяться з існуючої таблиці в нові.**
4. **Відбувається реліз версії ПЗ, що читає дані ЛИШЕ в новому форматі та з нових таблиць, запис в старому форматі ТА в новому.**
5. **Реліз ПЗ, з записом ЛИШЕ в новому форматі.**
6. **Відбувається третя міграція - видаляються непотрібні колонки зі старої таблиці.**

Що робити якщо допущена помилка

Типи міграцій:

- **Up - власне міграція, що потрібна.**
- **Down - “парна” міграція, що скасовує зміни поточної - опціональна.**

Існує 2 стратегії виправлення помилок міграцій:

- **Rollback - додавання окремої “down” міграції, що скасовує поточну.**
- **Roll forward - створення нової “up” міграції, що виправляє помилку, повна відмова від down міграцій.**

Наразі “roll forward” вважається “правильним” підходом та рекомендується використовувати саме його.

DDL та транзакції

PostgreSQL підтримує транзакційне виконання DDL запитів.

Таким чином, вся міграція, що може виконувати кілька змін схеми бази даних та зміни даних, виконується у одній транзакції як одна неподільна операція. Це надає всі переваги транзакційності для DDL операцій та міграцій в цілому.

Проте не всі СУБД підтримують транзакційність для DDL операцій.

Міграції даних

Окрім змін схеми баз даних, міграції використовується і для внесення змін до самих даних.

Прикладами можуть бути великі батчеві оновлення, вставка важливих даних таких як конфігурація і тд.

Правила тут такі ж самі як і з DDL - з обережністю відноситись до часу виконання запитів, що змінюють дані.

Практичні поради

- **Міграції НЕЗМІННІ.**
- **Варто робити міграції ідемпотентними - багаторазове їх виконання має бути можливим та результат багаторазового виконання має бути ідентичним одноразовому виконанню.**
- **Перед виконанням міграції її обов’язково потрібно тестувати і тестувати варто на очікуваних об’ємах даних в продакшені.**

Практичні поради

- **Варто притримуватись підходів багатокорокового релізу задля уникнення конфліктів ПЗ та схеми бази даних.**
- **Міграція має містити лише логічно неподільні зміни - краще мати кілька менших міграцій ніж одну велику.**
- **Зміну DDL та відповідну зміну даних варто тримати в межах однієї міграції.**

Запитання