

MySQL має декілька двіжків. Наприклад: InnoDB (**версіоннік**. Oracle, PostgreSQL - версіонніки), ISAM (**блокіровочнік**). В одних випадках краще одні, в інших - інші.

Версіонніки - дуступ до даних відбувається шляхом створення нових версій цих даних для окремих транзакцій.

Блокіровочніки - транзакція блокує данні, інша транзакція очікую поки перша не зафіксує зміну (комміт).

Вимога до СУБД. **ACID**:

**Atomicity** (атомарність) - атомарність транзакцій. Або відбуваються всі дії транзакції, або не відбуваєтся взагалі нічого. (роллбек)

**Consistency** (узгодженість) - транзакції не порушують взаємозвязок даних.

**Isolation** (ізоляція) - транзакції які працюють одночасно не впливають на одна на одну. Підтримка багатотисячного доступу до таблиць в БД відбувається за допомогою MVCC (MultyVersion Concurrency Control). Тобто контроль доступу до даних через створення багатьох версій цих даних.

**Durability** (надійність) - якщо транзакція успішно завершилася, то ніякі зовнішні чинники не повинні призвести до втрати змін які зробила транзакція.

Операції з даними умовно розділяють на:

**Select** - зчитуються валідні записи з таблиці. Запис вважається валідним, якщо вона створена транзакцією, яка була зафіксована, тобто зробила комміт по ній.

**Insert** - новий запис добавляється в вільне місце таблиці.

**Delete** - запис в таблиці помічається як невалідний. При чому сам запис не видаляється.

**Update** - комбінація delete/insert. Спочатку старий запис ми помічаємо невалідним, потім добавляються нові данні які коммітяться. При чому ростого флага валідності недостатньо, оскільки при баготопоточності у нас можуть бути колізії записів. Потрібен складніший механізм. Для деяких типів операцій використовуються блокування і операції до них доступні тільки в послідовному режимі.

**Реляційний** - від слова relation. (відношення, залежність, звязок)

2 процеси мають власний адресний простір в пам’яті і не мають спільної області пам’яті. ОС слідкує за тим аби один процес не виходив за рамки виділеної для неї пам’яті. Потоки ж можуть використовувати спільні об’єкти, тобто пам’ять бо процес може містити декілька потоків. Щоб дані зкопіювати з одного процеса в інший в Віндовс, наприклад, використовують буфер обміну. Ctrl C, Ctrl V.

Джава передає і зчитує дані в\з MySQL по TCP протоколу.

MySQL connector – **Connector**/**J** драйвер для MySQL. Можна скачати на офф сайті.

DAO лежить на гриниці між Bussines Layer і DB Layer.

Не рекомендується при виконанні метода, наприклад, вставки юзера, повертати код виконанняоперації (1 – вставлено, 2 – не вставлено…). Краще створити свої чекед виключення типу UserEmailAlreadyRegistered, NotUniqueUser etc. А сам метод робити void.

В своїх проекта для виключень рекомендують наслідуватися або від Exception або від RuntimeException аби отримати чекед або анчекед виключення відповідно. Поганою практикою ввжається наплоджувати багато власних ексепшенів які напряму наслідуються від Е або РЕ для свого модуля. Рекомендують створити один ексепшен для модуля який буде наслідуватися від Е або РЕ, а інші наші ексепшени будуть наслідуватися від цього ексепшена модуля. Таким чином не буде ризику того, що при кетчах по предку ми будемо перехоплювати і інші ексепшени типу IOE, FNFE, NPE etc.

В JDBC є параметри роботи з транзакціями. Там стоять по дефолту якісь налаштування. Один з параметрів – **рівень** **ізольованості** **транзакцій**: **TRANSACTION**\_**NONE** (не підтримується в MySQL), **TRANSACTION**\_**READ\_UNCOMMITED**, **TRANSACTION**\_**READ**\_**COMMITED**, **TRANSACTION**\_**REPEATABLE**\_**READ**, **TRANSACTION**\_**SERIALIZABLE**.

MySQL має функцію автокомміта, яка по замовчуванню ввімкнена. Це означає, що ми фактично не можемо робити ролбек бо після кожного SQL запиту буде автоматично відбуватися автокомміт. Якщо параметр автокомміта встановити в false, тоді ми можемо робити декілька SQL запитів, а потім робити комміт. Відповідно, в такому випадку, ми зможемо зробити ролбек, якщо нас якась з наших дій нас не влаштовуватиме.

**Connection** (абстракція одного фізичного з’єднання з БД) - приклад абстрактної фабрики. Паттерн абстрактна фабрика:

Connection генерує багато чого, але нас конкретно цікавлять 3 речі: Statement, Prepared Statement, Callable Statement.

|  |  |  |
| --- | --- | --- |
| Connection | Statement (надісланий в БД SQL) | Result Set (реляційне відношення яке вернулося на запит. Тобто таблиця) |
| Prepared Statement (надісланий в БД SQL) | Result Set (реляційне відношення яке вернулося на запит. Тобто таблиця.) |
| Callbale Statement | |

**Statement** - спосіб посилання SQL.

**PreparedStatmet** - спосіб посилання SQL. Працює з прекомпільованим SQL.

Коли ВМ посилає SQL в БД, то в конкретній БД відбуваються фази обробки запиту. Наприклад, в MySQL: 1) Парсинг і аналіз строки яка прийшла. Перевірка на правильність синтаксису тощо. 2) Вибір найефективнішого плану, по якому будуть шукатися елементи по таблиці. 3) execute (виконує запит). Також MySQL має механізм, який може пропускати перші 2 фази і виклнувати одразу третю, якщо у нас однотипні запити, щоб не нагружати ЦП. У нас в JDBC даним механізм є Prepared Statement.

**Callable** Statement - наш спосіб викликати збережену (хранимую) процедуру з БД. Не всі БД можуть мати хранимі процедури. І вони написані на якійсь свої внутрішній мові. Скоріш за все, якщо БД не має хранимих процедур, то ми отримаємо SQLException.

Джаві зручно працювати з великою кількістю короткоживучих об’єктів (new в Джаві виконується дуже швидко. Порядка 10 тактів процесора). А для БД, навпаки, зручніше створювати connection і працювати по ньому постійно, оскільки постійне відкриття і закриття нових конекшнів б’є по продуктивності БД. Також варто відмітити, що connection в БД не багатопоточний і ми не можемо писати в один конекшн одночасно. Одним з вирішенняи проблеми такого роду є опенсорс проект C3PO (сітріпіо). Вводиться поняття Connection pool. Пулів може бути декілька.

З точки зору ВМ і з точки зору MySQL у нас різні механізми роботи з Connection. Ця різність розв’язується за допомогою пула з’єднань. Пул з’єднань виділяє фіксовану кількість з’нань до БД. Цей конекшн пул зі сторони ВМ виглядає як JDBC драйвер. Коли ми з ВМ просимо конекшн, пул з’єднань перевіряє чи є у нього вільний конекшн з фіксованого пула і повертає нам так звану обгортку конекшена, яка зі сторони нічим не відрізняється від конешена (interface Connection). Ця обгортка є реалізацією паттерана Proxy. Якщо вільного конекшена нема - створюється новий. Якщо ми з ВМ даємо конекшену close(), то закриється тільки обгортка, а сам конекшн з пула не закриється. Connection не потокозахищений. Тому 2 і більше потоки з ним працювати не можуть - розвалиться.

**Pattern Proxy** - схожий на паттерн Декоратор, але не добавляє функціональності. Це коли ми маємо об’єкт і клієнта який хоче отримати наш об’єкт, а ми замість нашого об’єкта даємо інший об’єкт такого ж типу, щоб клієнт не мав прямого доступу до нашого першого об’єкта. Наприклад, наш об’єкта знаходиться на іншій машині і щоб клієнт не ходив на іншу машину і не дізнавався непотрібну для нього лишню інформацію - ми створюємо об’єкт такого ж типу як віддалений, який має такі ж методи і даємо клієнту працювати безпосередньо з ним. Тоді виникає відчуття, що клієнт не ходив на віддалену машину, а результат отримався безпосередньо в другому створеному об’єкті, який дьоргає віддалений об’єкт. Також наш об’єкт перевіряє чи має право клієнт на доступ до віддаленого об’єкту.

В крупних проектах не використовують шаблон DAO, а якийсь ORM. Наприклад, Hibernate. І він сам в рантаймі генерує SQL.

Зазвичай в папці проекта лежить SQL файл, який показувє як створювати еталонну таблицю для даного проекту. Інколи можк лежати 2 файла. Один для створення баз і таблиць, інший для наповнення їх. Просто запустивши їх ми маємо готову базу, наприклад, для тестів.

При створенні таблиці MySQL автоматично робить автокомміт.

**SQL** DML - підмножина мови SQL. SELECT, UPDATE, INSERT, DELETE. Тобто оператори маніпулювання з данними таблиці.

**SQL** DDL - підмножина мови SQL. Створення таблиць, видалення таблиць, зміна структури таблиць, добавлення індексів тощо. Мають автокомміт. Не можуть використовувати транзакцію в силу цього.

**Batch** - набір операторів TransactSQL, які передаються і виконуються як єдине ціле.

По факту, статичне поле - це глобальна змінна.

ГК шукає елементи на які зсилаються з стеку і перевіряє їх транзитивні замикання. Він не слідкує за мертвими об’єктами в пам’яті. Тобто визначає об’єкти які живі, а всю іншу пам’ять вважає вільною і перезаписує в неї нові об’єкти. Попередньо перед цим ГК робить дефрагментацію об’єктів в пам’яті тому Memory Manager знає, що "зліва" пам’ять зайнята, а "зправа" вільна.

Hibernate має механізм лінивого завантаження (lazy loading), який дозволяє викачувати з БД об’єкт не повністю з його всіма транзитивними замиканнями, а тільки сам об’єкт, а інші елементи підгружати безпосередню тоді, коли ми до них звертаємся. Цей механізм можна тюнити.

xa-транзакція - транзакція, яка може охопити декілька баз даних під час свого виконання. При чому БД повинні підтримувати для цього xa-транзакції.

**JDBC** - Java database conectivity. Стандарт взаємодії Джава додатків з БД. Пакети java.sql, javax.sql.

JCP - стандартизована специфікація для розробників.

JDBC основано на концепції драйверів, які дозволяють отримувати зв’язок з БД по URL через ТСР\ІР. Драйвери можуть підгружатися автоматично по ходу виконання програми. Коли дайвер загружається, то він сам реєструє себе і викликається автоматично, коли програма запрошує URL з протоклом до якого прив’язаний драйвер.

JDBC API основано на двох типах інтерфейсів: одні для розробки додатків, інші для розробки драйверів (більш низкорівневі).

Наш аплікейшн працює з БД через JDBC Driver Manager (JDBC API) -- JDBC Postgres Driver -- Database. Таким чином ми не залежимо від реалізації конкретної БД бо набір методів для роботи з будь-якою БД буде одинаковим і реалізується в драйверы конкретної БД.

Зв’язок з базою реалізується класом, який розширяє інтерфейс java.sql.Conection.

Коли ми маємо з’єднання з БД, ми можемо створювати об’єкти типу Statement, які використовують для виконання запитів до БД на мові SQL.

|  |  |  |
| --- | --- | --- |
| Connection | Statement (надісланий в БД SQL) | ResultSet (реляційне відношення яке вернулося на запит. Тобто таблиця) |
| PreparedStatement (надісланий в БД SQL). Працює з прекомпільованим SQL. | ResultSet (реляційне відношення яке вернулося на запит. Тобто таблиця.) |
| CallbaleStatement (для виклику хранимих процедур. CALL процедура, або EXECUTE процедура) | |

**Хранима процедура** - об’єкт в БД, який представляє собою набір SQL інструкцій, який компілюється один раз і зберігається на сервері. Крім DDL і DML в них можуть виконуватися цикли розгалуження, тобто можна керувати процесом виконання. Можуть мати вхідні параметри, локальні змінні, виконувати обчислення, тощо...

Реалізація хранимих процедур для PostgreSQL - PL/pgSQL, PL/TCL, PL/Perl, PL/Phyton. Тобто це підмови, які дозволяють писати функції і тригери для СУБД PostgreSQL.

**SQL** DML - підмножина мови SQL. SELECT, UPDATE, INSERT, DELETE. Тобто оператори маніпулювання з данними таблиці.

**SQL** DDL - підмножина мови SQL. Створення таблиць, видалення таблиць, зміна структури таблиць, добавлення індексів тощо. Мають автокомміт. Не можуть використовувати транзакцію в силу цього.

**CRUD** (**create**, **read**, **update**, **delete**) - 4 базові функції, які використовуються при роботі з персистентними сховищами даних. Тобто створення, читання, редактування і видалення.

**CREATE**

CREATE TABLE users (

'id' NOT NULL AUTO\_INCREMENT,

'name' VARCHAR(20) NOT NULL,

'age' INT(3) NULL,

'email' VARCHAR(64) NOT NULL,

PTIMARY KEY ('id')

);

INSERT INTO users (name, age, email) VALUES ('Mike', 25 'examp@gmail.com')

**READ**

SELECT name, age FROM users WHERE id =1;

#SELECT \* FROM users; //all

**UPDATE**

UPDATE users SET name ='Steave', age=26 where id=1;

**DELETE**

DELETE FROM users WHERE id=2;

#DELETE FROM users; //all