3.30. Реализация реляционных СУБД. Индексы, применение индексов. Транзакции, свойства транзакций. Блокировки, типы блокировок.

В реляционных БД данные организованы в виде одной или нескольких таблиц.
Столбцы таблицы - поля одного из скалярных типов (например: целое число, вещественное число, строка, дата).
Набор столбцов (схема таблицы) задается при создании таблицы и во время работы с БД, как правило, не изменяется.
Строки таблицы - собственно записи с данными.

Часто в таблице выделяется специальное поле, содержащее уникальный номер записи, обычно называемое "id" или похожим образом. С помощью этого механизма можно реализовывать связи между таблицами: создать в одной таблице поле, содержащее идентификатор записи в другой таблице.

Взаимодействие с СУБД (системой управления базами данных) происходит посредством специального языка запросов (стандартом в этой области является SQL - Structured Query Language).

Основные виды SQL-запросов для манипуляций с данными:
- SELECT - выборка из одной или нескольких таблиц, результатом выполнения всегда является таблица.
- INSERT - добавление записей в таблицу
- DELETE - удаление записей из таблицы
- UPDATE - модификация значений полей у существующих записей

Для запросов SELECT, DELETE и UPDATE с помощью ключевого слова WHERE могут быть указаны условия, с какими записями выполнить действие.
(наиболее типичный пример условия - выбор записи по ее id).

В общем случае, чтобы СУБД могла определить, для каких рядов выполняются эти условия, требуется полный перебор всех записей в таблице (full table scan) и проверка условий для каждой из них. Очевидно, что при значительном количестве записей время обработки запросов становится неприемлемым. Выход из этой ситуации - использование индексов.

Индексы

Индекс - это вспомогательная структура данных в СУБД, позволяющая быстро выполнять поиск и сортировку по определенному выражению (наиболее распространенный частный случай - значение определенного поля). Для каждой таблицы определяется свой набор индексов.

Классификация индексов:

по реализации:
- hash - реализуются с помощью хэш-таблиц, не поддерживают сортировку.
- tree - реализуются с помощью разновидностей B-деревьев.

по предмету индексирования:
- простые - одно поле таблицы
- составные - несколько полей таблицы в определенном порядке
- выражение - произвольное выражение над полями (например: строка, приведенная к верхнему регистру)

Первое применение индексов - ускорение поиска. Например, при выборке записи по ее id, поиск займет время O(N) при отсутствии индекса на поле id (N - количество строк в таблице), O(logN) - при наличии tree-индекса, и O(1) - при наличии hash-индекса. СУБД автоматически определяет, можно ли использовать какой-либо индекс при обработке запроса. Иногда при малом количестве записей полный перебор сработает быстрее, чем использование индекса, СУБД может распознавать такие случаи и игнорировать наличие индекса.

Второе применение индексов - сортировка, если в запросе задан требуемый порядок сортировки результатов. Без индексов потребуется выполнять эту сортировку после выборки всех подходящих строк, что требует время O(NlogN). С помощью индекса на выражении, по которому выполняется сортировка, можно получить результат за O(N). Если же при этом еще и требуется ограничить результат первыми K строками (где K может быть много меньше N), разница еще более критична - O(NlogN) против O(K).

Третье применение индексов - возможность выполнять некоторые простые запросы без обращения к основным данным таблицы вообще, используя только индекс. Например, получить максимальный использованный id в таблице.

Четвертое применение индексов - возможность задать некоторые ограничения на данные. Например, что в таблице нет двух строк с одинаковым id (уникальность), или что значение в определенном поле - есть id реально существующей записи в другой таблице (ссылочная целостность).

Недостатки индексов:
- занимают дополнительную память
- усложняют запросы INSERT/DELETE/UPDATE, поскольку после них требуется обновлять индексы

Итого, индексы стоит применять там, где это помогает избежать полного сканирования таблицы на часто используемых запросах.

Транзакции

Транзакцией называется последовательность команд, выполняемая над базой данных как одно целое. Транзакция должна удовлетворять свойствам ACID:

* Atomicity (атомарность): транзакция является наименьшим, неделимым блоком алгоритма изменения данных. Части транзакции либо выполняются все, либо не выполняется ни одной такой части. Поскольку на самом деле невозможно одновременно и атомарно выполнить последовательность команд внутри транзакции, вводится понятие <отката> (rollback): если транзакцию не удаётся полностью завершить, результаты всех до сих пор произведённых действий должны быть отменены и система возвращается в исходное состояние.
* Consistency (непротиворечивость): по окончанию транзакция оставляет данные в непротиворечивом состоянии. Скажем, если поле в базе данных описано как имеющее только уникальные значения строк, то при любом исходе транзакции строк-дубликатов появиться не может.
* Isolation (изоляция): во время выполнения транзакции другие процессы не должны видеть данные в промежуточном состоянии. Например, если транзакция изменяет сразу несколько полей в базе данных, то другой запрос, выполненный во время выполнения транзакции, не должен вернуть некоторые из этих полей с новыми значениями, а другие с исходными.
* Durability (надежность): независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, останутся сохранёнными после возвращения системы в работу. Если пользователь получил подтверждение от системы, что транзакция выполнена, гарантируется, что сделанные им изменения не будут отменены из-за какого-либо сбоя. Достигается ведением журнала транзакций, куда пишутся все завершенные транзакции перед реальным внесением изменений в данные; при сбое по этому журналу можно повторно выполнить прерванные транзакции.

Блокировки

Наиболее простой метод обеспечения перечисленных выше свойств транзакций - блокировки таблиц или отдельных строк.

Блокировки бывают двух типов - на чтение (read lock) и запись (write lock); write lock может быть получен только при отсутствии любых других блокировок, read lock-и друг другу не мешают.

Перед началом транзакции процесс берет блокировки на все читаемые или изменяемые им данные, после окончания - снимает блокировки (это называется двухфазной блокировкой - 2-phase locking). Все другие процессы, обращающиеся к тем же данным, ждут.

Недостатки такого подхода - низкая параллельность (concurrency), и опасность возникновения deadlock-ов. Последние предотвращаются заданием единого порядка взятия блокировок на ресурсы, а если deadlock все же произошел, одна из транзакций откатывается (например, по таймауту).

Избавиться от блокировок можно с помощью хранения нескольких копий изменяемых данных, при этом надо определять, какую копию какой процесс в данный момент должен видеть.