Продвинутая группировка GROUPING SET, ROLLUP, CUBE

Common table expression

Иерархические и рекурсивные запросы

ORM

Explain – просмотр плана выполнения запроса и связанных с ним накладных расходов.

Explain analize – исполняет запрос и показывает план этого запроса и его реальный результат

Команда use

Union

Партиционирование/шардирование

Sql - инъекция

Можно создать таблицу используя like, в postgre есть наследование

no action – не удалять

restrict -??

Cascade – изменить каскадно

Set null – если удаляешь установить в null

Set default – установить в def

Именовать ограничения нужно, если база большая, ибо если будешь удалять его в будущем, то хер найдешь

[union, intersect, except] у union опция или all или distinct(по умеолчанию)

Offset – в совокупности с лимитом?

Fetch?

Разбивка на страницы и row\_number?

With – Дробление подзапроса на части и рекурсивные подзапросы

Returning

Collation – регистронезависимый поиск – правила для каждого языка свои?

Select \* from table1, table2 – тоже самое что cross join

Inner – тоже декартово произведение, только с условием совпадения строк

Как посмотреть все индексы в табилце

Join -> некорелирующий-> корелирующий

Мультиверсионность как средство изоляции

Грязное чтение в постгресе невозможно

ХА-транзакции – распределенные транзакции, менеджер транзакций

Хэш индекс не пишется в журнал, субд создает временные хэш индексы при Join

Битовый индекс – нельзя создать руками

Кластерный индекс не сортирует автоматически новые данные

Покрывающий индекс – работает не всегда, так как у него нет информации видна запись или нет и единственный способ узнать – прочитать запись. То есть покрывающий индекс может быть использован, только если СУБД может точно сказать, что не существует невидимых записей

В postgre не создаются индексы для foreign key

Join-стратегии

Выполнение запроса – парсинг, планирование, исполнение

Join лучше, поскольку использует параллельное выполнение, нежели подзапрос, который использует последовательное

Оптимизация – не читать лишних полей, не анализировать лишние строки, денормализация структуры – даст ускорение чтения данных, оптимизация операторов (delete/truncate, count заменить на статистический подстчет, если можно жертвовать точностью, оптимизация limit offset – он все-равно читает пропускаемые строки и отбрасывает, заменить на оператор сравнения)

Truncate нарушает изоляцию транзакций

Секционирование данных – партиционирование, в postgre через наследование

Репликация

For share и for update – блокирует для таких же запросов

Ожидание на запись

Ожидание на чтение

Несогласованное чтение

Локаль – запись даты

Union – объединение и intersect – пересечение, except – исключение (есть all, так же вычисляет разницу, но при all будут учтены кол-во повторяющихся значений)

\*С функциями, вставка в несколько таблиц, \*

Дефолт значения для столбцов, в случае отсутствия значения при добавлении/обновлении

\* RETURNING,

:: - знак приведения типов

’12:34:56’ :: Time – результат тип данных time

DDL запросы проталкавают коммит – то есть автоматически коммитятся, если в сессии есть незакоммиченные изменения и выполнить DDL команду то они будут закоммичены

Val характеризует durability с помощью записи в журнал изменений или журнал транзакций – в журнал записать быстрее, чем в базу, и если она падает, то записи остаются в журнале.

**Merge – оператор, вызывающий на каждую строку update else insert\*\*\***

RESTART IDENTITY – начинает последовательность с начала (при вызове на таблице truncate он не обнулит последовательность, а пойдет дальше)

**Select for update** – Блокировка на уровне строки, исключение конкурентного доступа\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Returning в insert, update, delete для возврата значений

Домены – были на картинке в таблице – есть ли связь с нормализацией?

\*\*\*\*\*Материализуемые представления -

содержат данные, повышают скорость, при обновлении данных в таблице само не обновится, надо рефрешить

Используется для кэширования с помощью материализации

checkpointer