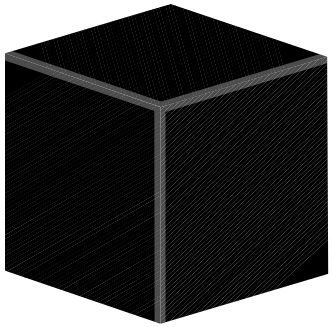
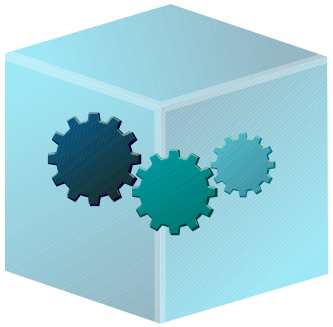
Транзакции и одновременный доступ: сравнение реализаций в PostgreSQL и Oracle

Егор Рогов, Postgres Professional

Зачем все это?

запрос  результат

Зачем все это?

запрос  результат

Транзакции

Атомарность — все или ничего

Согласованность — целостность

Изоляция — никого не вижу

- serializable..... фантомное чтение
- repeatable read.....неповторяющееся чтение
- read committed..... «грязное» чтение
- read uncommitted.....потерянные изменения

Долговечность — ничто не забыто

Зачем четыре уровня?
Почему именно такие проблемы?

Изоляция — никого не вижу

- serializable..... фантомное чтение
- repeatable read.....неповторяющееся чтение
- read committed..... «грязное» чтение
- read uncommitted.....потерянные изменения

Попробуем придумать свою реализацию...

Попытка 0

Вообще не будем управлять доступом.

Одна крайность: полный хаос и непредсказуемость

Это не СУБД, а файл

Попытка 1

Одна эксклюзивная блокировка на уровне всей СУБД; устанавливается и записью, и чтением.

Другая крайность: идеальная изоляция... за счет строго последовательного выполнения

Истина где-то посередине

- попробуем ослабить блокировки и сделать их мельче
- как минимум нужна блокировка одновременного изменения

Попытка 2

Блокировки на строках таблицы;
устанавливаются записью, но чтение на них не смотрит.

```
select sum(amount)
from t;
```

acc_id	amount
1	60
2	40

Попытка 2

Блокировки на строках таблицы;
устанавливаются записью, но чтение на них не смотрит.

```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

Попытка 2

Блокировки на строках таблицы;
устанавливаются записью, но чтение на них не смотрит.

```
select sum(amount)
from t; -- 60
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	140

Попытка 2

Блокировки на строках таблицы;
устанавливаются записью, но чтение на них не смотрит.

```
select sum(amount)
from t; -- 200
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	140

Попытка 2

Блокировки на строках таблицы;
устанавливаются записью, но чтение на них не смотрит.

```
select sum(amount)
from t; -- 200
```

acc_id	amount
1	60
2	40

```
update t
set amount=amount+100
where acc_id=2;

rollback;
```

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t;
```

acc_id	amount
1	60
2	40

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t; -- 60
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	**140**

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t; -- 60
```

```
update t
set amount=amount+100
where acc_id=2;
```

	acc_id	amount
	1	60
WAIT	2	**140**

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

```
update t
set amount=amount+100
where acc_id=2;

rollback;
```

Попытка 3

Эксклюзивные блокировки на строках таблицы; устанавливаются записью.

```
select sum(amount)
from t; -- 100
```

acc_id	amount
1	60
2	40

```
update t
set amount=amount+100
where acc_id=2;

rollback;
```

READ COMMITTED :)

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t;
```

acc_id	amount
1	60
2	40

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

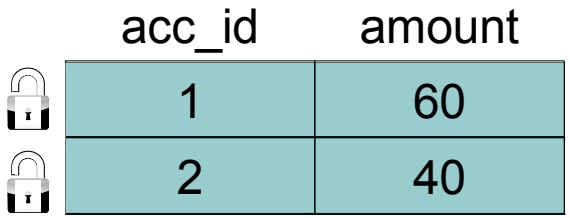
```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100
```



Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
```

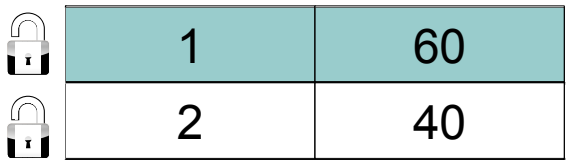
acc_id	amount
1	60
2	40

WAIT

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100

select  sum(amount)
from t; -- 60
```

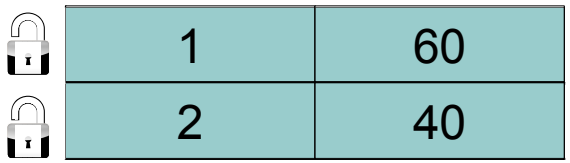



```
update t
set amount=amount+100
where acc_id=2;

WAIT
```

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100

select  sum(amount)
from t; -- 100
```




```
update t
set amount=amount+100
where acc_id=2;

WAIT
```

Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100

select  sum(amount)
from t; -- 100

commit;
```

acc_id	amount
1	60
2	40

```
update t
set amount=amount+100
where acc_id=2;
```

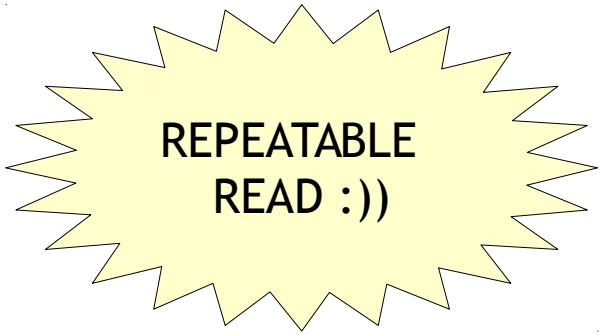
Попытка 4

Эксклюзивные блокировки устанавливаются записью, разделяемые — чтением.

```
select sum(amount)
from t; -- 100

select  sum(amount)
from t; -- 100

commit;
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	**140**



Промежуточные выводы

Чем выше уровень изоляции, тем больше нужно блокировок

Чем больше блокировок, тем

- хуже производительность
- выше вероятность взаимоблокировок

Фантомное чтение не устраняется блокировками уровня строк

Есть сложности с согласованным чтением

Нужен другой подход!

Итак, многоверсионность!

На низком уровне — блоки данных
- при изменении строк сохраняем все предыдущие версии
- аналогично и при удалении

На высоком уровне — снимки данных
- в снимок попадает максимум одна версия каждой строки
- согласованные данные на какой-то момент времени

Транзакции работают только со снимками

Снимок для каждой операции

```
select sum(amount)
from t;
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

Снимок для каждой операции

```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

Снимок для каждой операции

```
select sum(amount)
from t; -- 60
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

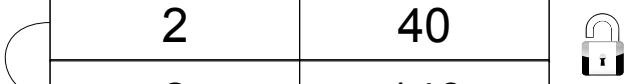
Снимок для каждой операции

```
select sum(amount)
from t; -- 60
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40
2	140



acc_id	amount
1	60
2	140

Снимок для каждой операции

```
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	40

READ COMMITTED :)

acc_id	amount
1	60
2	40
2	140

acc_id	amount
1	60
2	140

Снимок для каждой транзакции

```
select sum(amount)
from t;
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 60
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
```

acc_id	amount
1	60
2	40

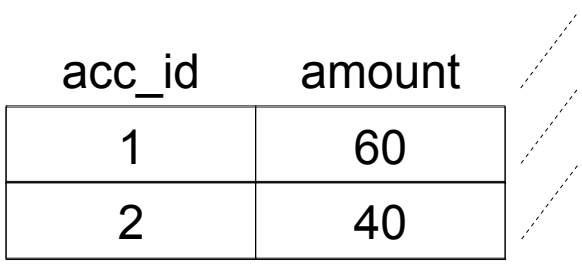
acc_id	amount
1	60
2	40

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	40



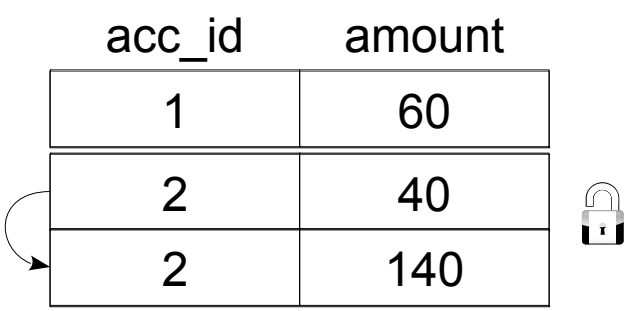
acc_id	amount
1	60
2	40

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	40



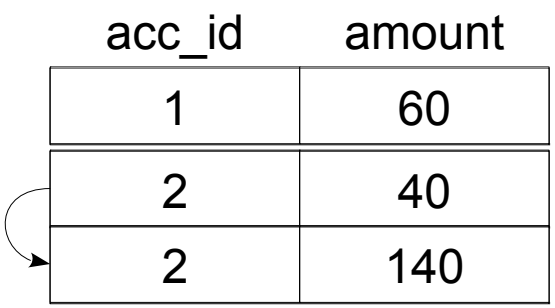
acc_id	amount
1	60
2	140

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
commit;
```

acc_id	amount
1	60
2	40



acc_id	amount
1	60
2	140

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
select sum(amount)
from t; -- 60
```

тот же снимок

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40
2	140

```
update t
set amount=amount+100
where acc_id=2;
commit;
```

acc_id	amount
1	60
2	140

Снимок для каждой транзакции

```
select sum(amount)
from t; -- 100
select sum(amount)
from t; -- 100
```

```
update t
set amount=amount+100
where acc_id=2;
commit;
```

acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40
2	140

acc_id	amount
1	60
2	140

REPEATABLE READ :))

Блокировки

Читатели не блокируют читателей
Читатели не блокируют писателей
Писатели не блокируют читателей

Блокируется только одновременное изменение строки

Необходимость в уровне read uncommitted отпадает

Чудес не бывает

От старых версий строк надо физически избавляться
(если они больше не видны ни в одним снимке)

Снимок, сделанный в начале транзакции, решает проблему фантомных чтений, но не обеспечивает уровень serializable
(нужен другой механизм, например, предикатные блокировки)

Не все так просто с изменением строк

Трудности одновременного изменения

```
update t
set amount=amount+100
where acc_id=2;
```

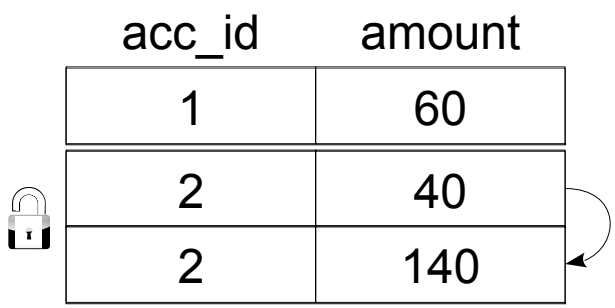
acc_id	amount
1	60
2	40

acc_id	amount
1	60
2	40

Трудности одновременного изменения

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	140

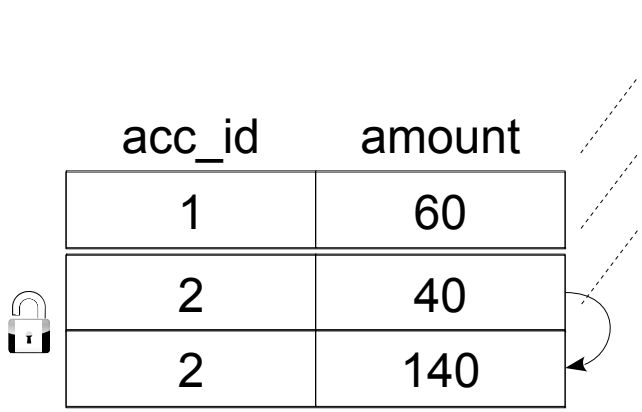


Трудности одновременного изменения

```
update t
set amount=amount+100
where acc_id=2;
```

```
update t
set amount=amount+100
where acc_id=2;
```

acc_id	amount
1	60
2	140

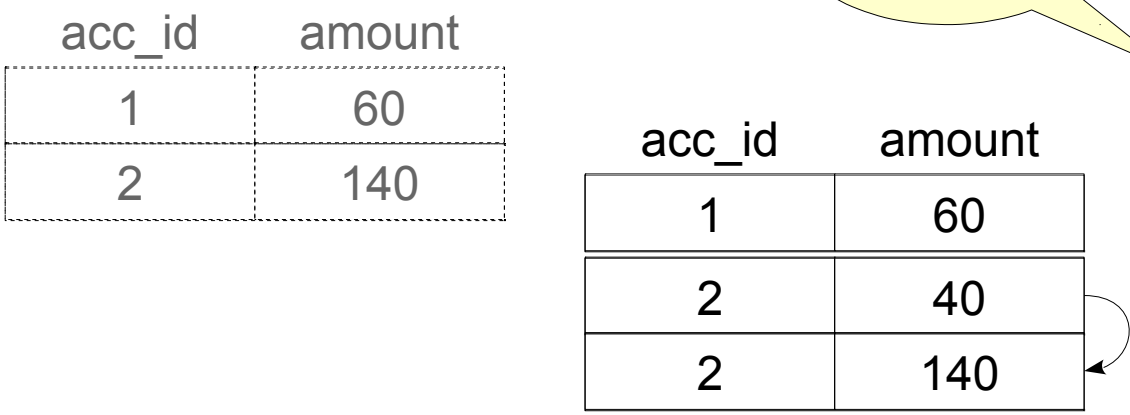


acc_id	amount
1	60
2	40

Трудности одновременного изменения

```
update t
set amount=amount+100
where acc_id=2;
commit;
```

```
update t
set amount=amount+100
where acc_id=2;
```



Трудности одновременного изменения

В таких случаях операция прозрачно «перезапускается», чтобы сделать снимок заново

Но это возможно только для read committed;
в случае repeatable read (и serializable) получим ошибку — транзакцию просто так не перезапустишь

Лирическое отступление

Какое значение это все имеет для разработчика?

Уровень read uncommitted практически бесполезен

Уровень read committed является стандартом де-факто

- обеспечивает согласованность
- иногда требует ручных блокировок в коде

Уровень serializable может быть полезен

- если транзакция правильно работает одна,
  то будет правильно работать и одновременно с другими
- код должен уметь перезапускать транзакции

Возможные реализации

Путь PostgreSQL — хранить старые версии строк в блоках

Снимок определяет, какие из версий должны быть видны

Старые версии постепенно удаляются

Путь Oracle — хранить в блоках только актуальные данные

Если для снимка нужна старая версия, она восстанавливается из журнала отката

Журналы отката циклически перезаписываются

На что обратим внимание?

Чем определяется момент времени, как упорядочены события?

Несколько версий чего именно поддерживается?

Как происходят фиксация и откат?

Как устроен снимок данных?

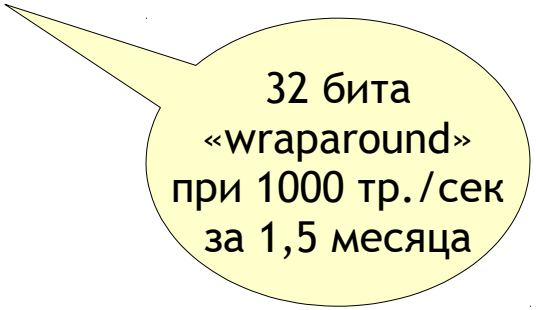
Как удаляются ненужные версии?

Как это выглядит на низком уровне?

PostgreSQL

Чем определяется момент времени?

Последовательные номера транзакций

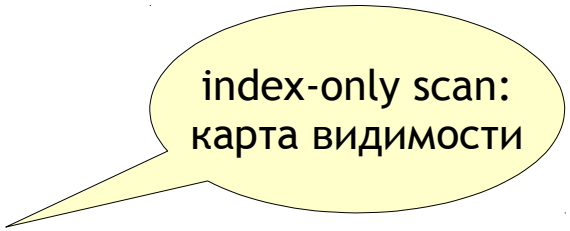


PostgreSQL

Несколько версий чего именно поддерживается?

Единица многоверсионности — строка (tuple)
- начальный номер транзакции (xmin)
- конечный номер транзакции (xmax)

При вставке номер транзакции записывается как начальный
При удалении номер транзакции записывается как конечный
Обновление = удаление + вставка



Индексные блоки не содержат информацию
о версионности, записи ссылаются на каждую из версий строк

PostgreSQL

Как происходят фиксация и откат?

Статус каждой транзакции отмечается в списке CLOG
- wraparound: кольцевой буфер, «начало отсчета» сдвигается

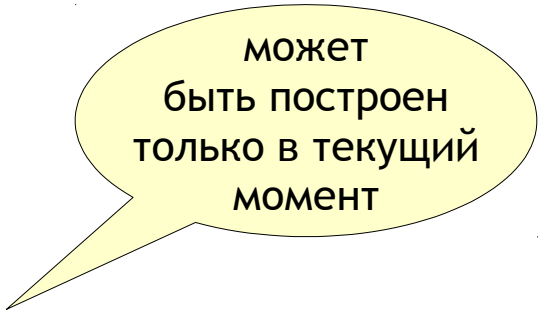
Фиксация и откат — изменение статуса в списке

выполняются быстро :)

Если транзакция отменена,
надо исправить xmin и xmax в строках
- это можно делать постепенно

PostgreSQL

Как устроен снимок данных?



Снимок:

- ближайший номер транзакции (определяет момент времени)
- список активных транзакций на текущий момент

Этой информации достаточно, чтобы выбрать версии, которые

- уже были зафиксированы (xmin)
- и еще не были удалены (xmax)
- или принадлежат самой транзакции

Точные правила видимости достаточно сложны

PostgreSQL

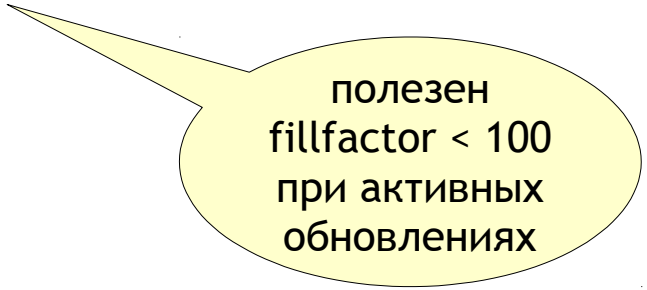
Как удаляются ненужные версии?

Периодический процесс очистки (VACUUM)

- удаляет версии, не входящие ни в один снимок
- удаляет ссылки из индексных блоков

Оптимизация HOT update

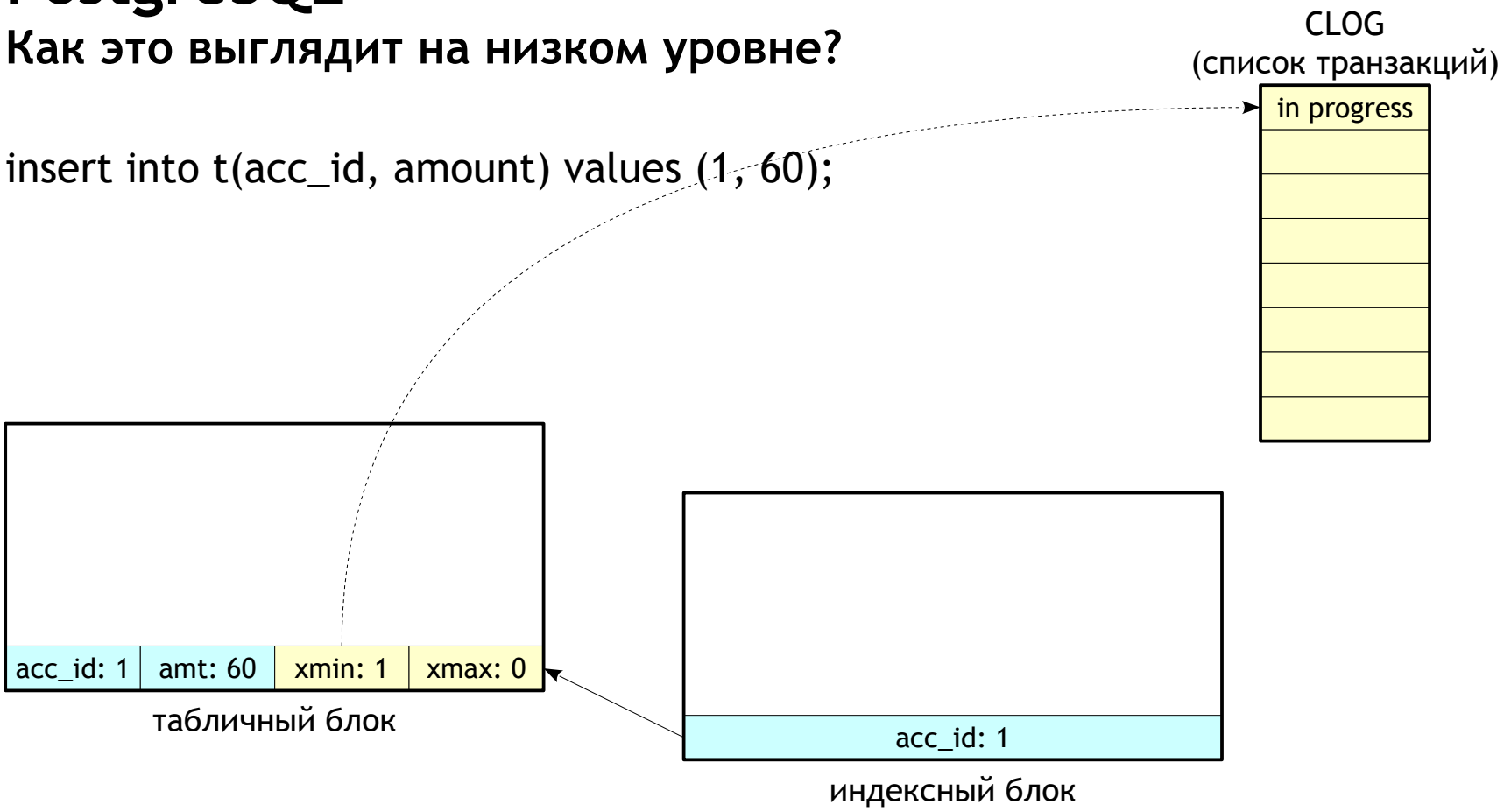
- не создает лишних ссылок из индексных блоков, если не менялись индексированные поля
- действует в пределах блока



PostgreSQL

Как это выглядит на низком уровне?

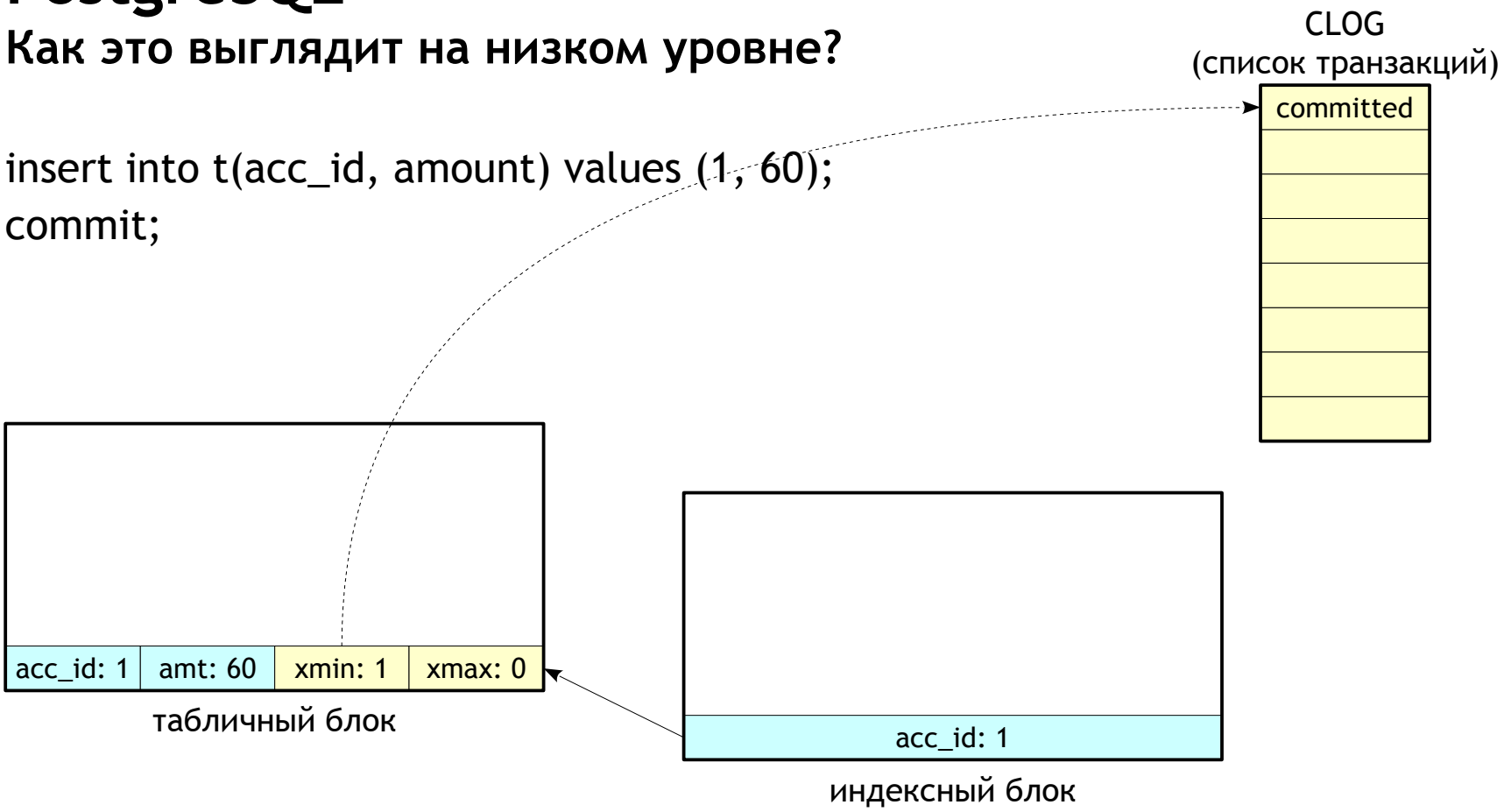
insert into t(acc_id, amount) values (1, 60);



PostgreSQL

Как это выглядит на низком уровне?

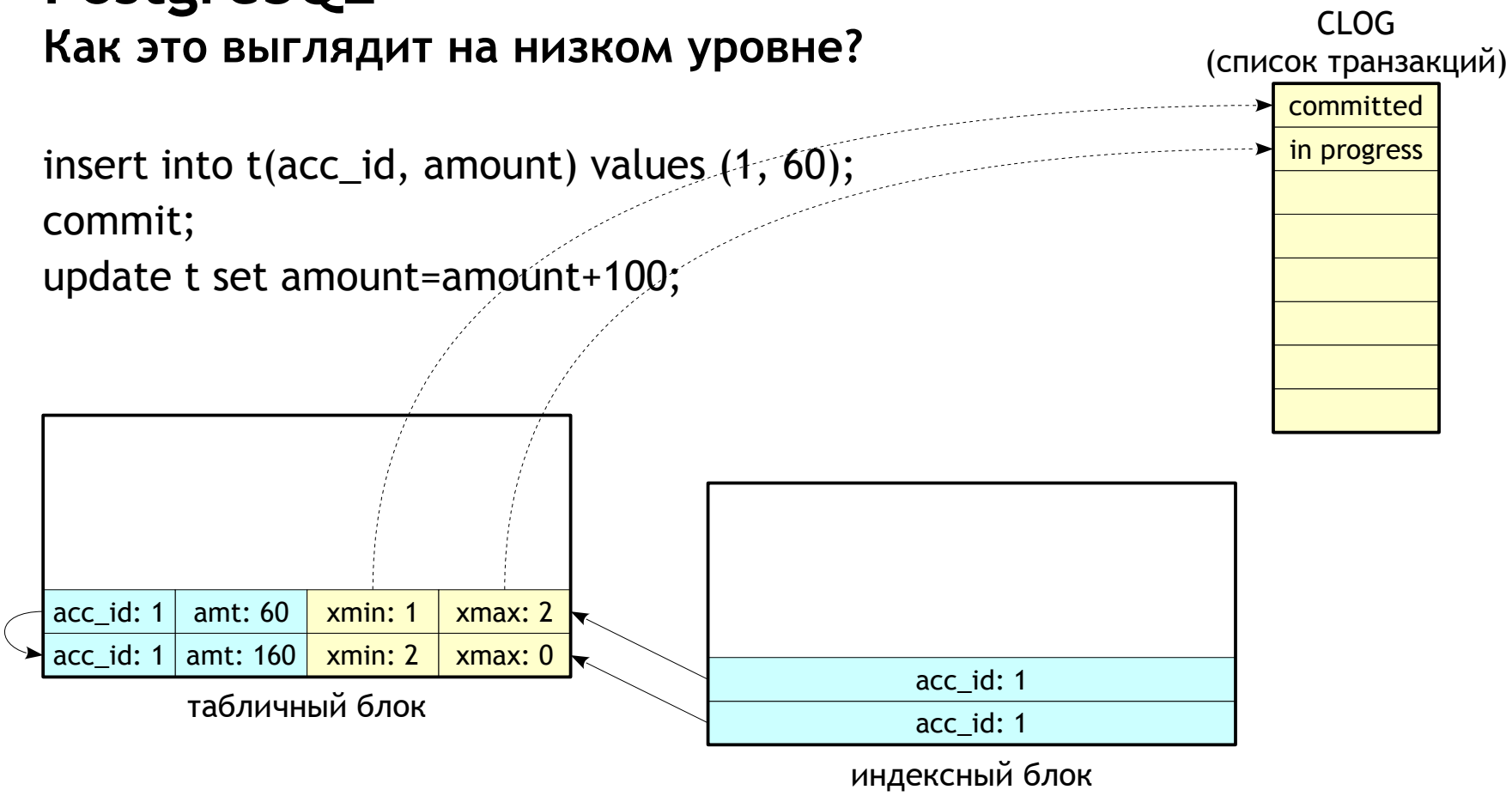
insert into t(acc_id, amount) values (1, 60);
commit;



PostgreSQL

Как это выглядит на низком уровне?

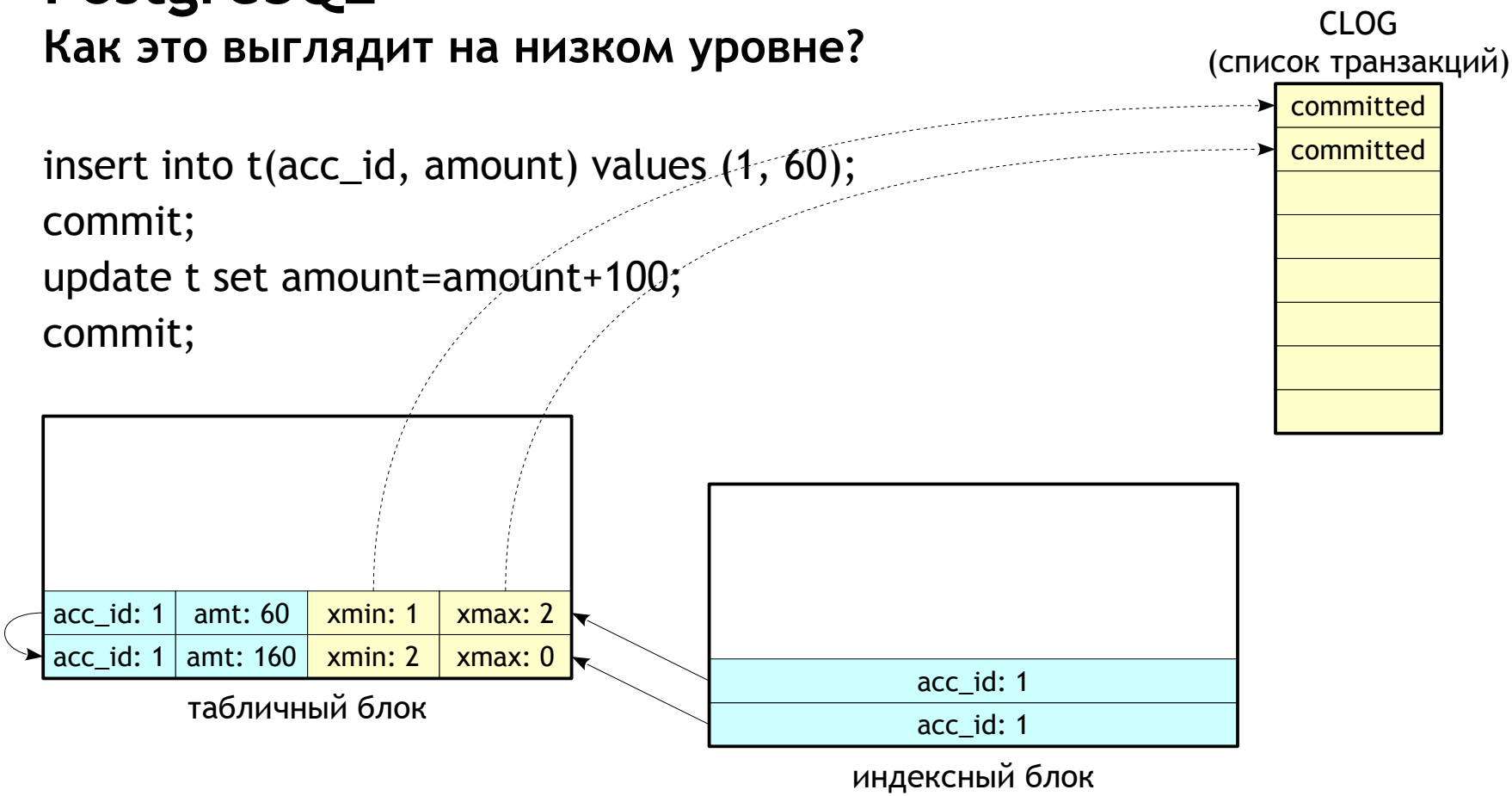
```
insert into t(acc_id, amount) values (1, 60);
commit;
update t set amount=amount+100;
```



PostgreSQL

Как это выглядит на низком уровне?

```
insert into t(acc_id, amount) values (1, 60);
commit;
update t set amount=amount+100;
commit;
```



Oracle

Чем определяется момент времени?

Счетчик System Change Number (SCN)

48 битов

Увеличивается как минимум при каждой фиксации и откате

Номера транзакций не последовательны

Oracle

Несколько версий чего именно поддерживается?

Единица многоверсионности — блок
(и табличный, и индексный)

index-only scan :)

При любом изменении в блоке в журнал отката (undo log) пишется минимальная информация, необходимая для отмены:

- при вставке строки — указание о ее удалении
- при изменении — значения изменившихся полей
- при удалении — вся строка

меньше,
чем все строка

Номер транзакции — ссылка на цепочку записей в журнале отката

Oracle

Как устроен снимок данных?

Снимок = SCN

можно вернуться в прошлое (flashback)

Блоки содержат Interested Transactions List (ITL):

допол-нительная блокировка

- список транзакций, меняющих блок
- их статус и SCN фиксации

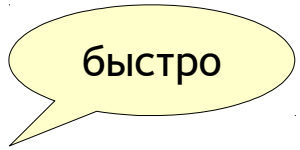
ITL ограничен, но завершенные транзакции перезаписываются

После чтения блока надо получить в нем согласованные данные:

- откат всех активных транзакций из ITL
- откат зафиксированных транзакций из ITL до тех пор, пока SCN блока не достигнет SCN снимка

Oracle

Как происходят фиксация и откат?



Фиксация — смена статуса транзакции в журнале отката

Откат транзакции приводит к откату всех ее изменений

При фиксации или откате надо изменить статус в ITL блоков

- это можно делать постепенно

Oracle

Как удаляются ненужные версии?

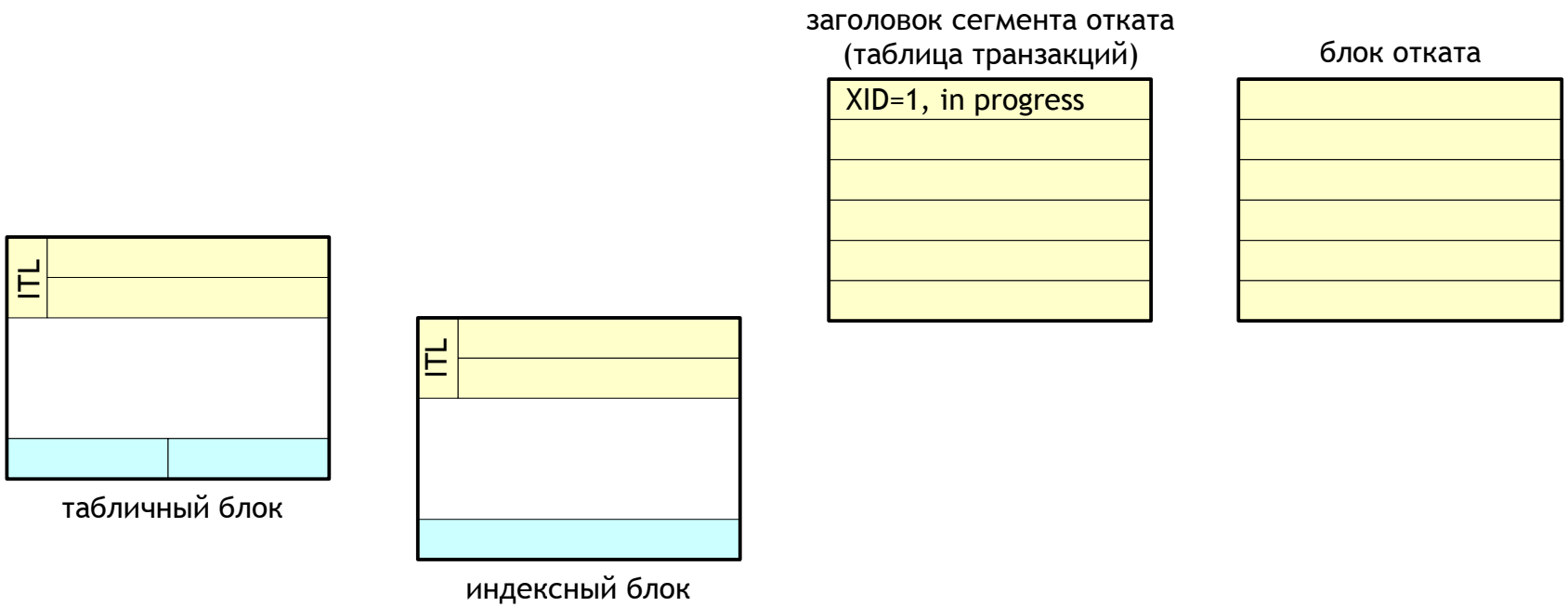
Журналы отката уже завершенных транзакций перезаписываются

- не нужна периодическая очистка
- невозможно построить «слишком старый» снимок

Oracle

Как это выглядит на низком уровне?

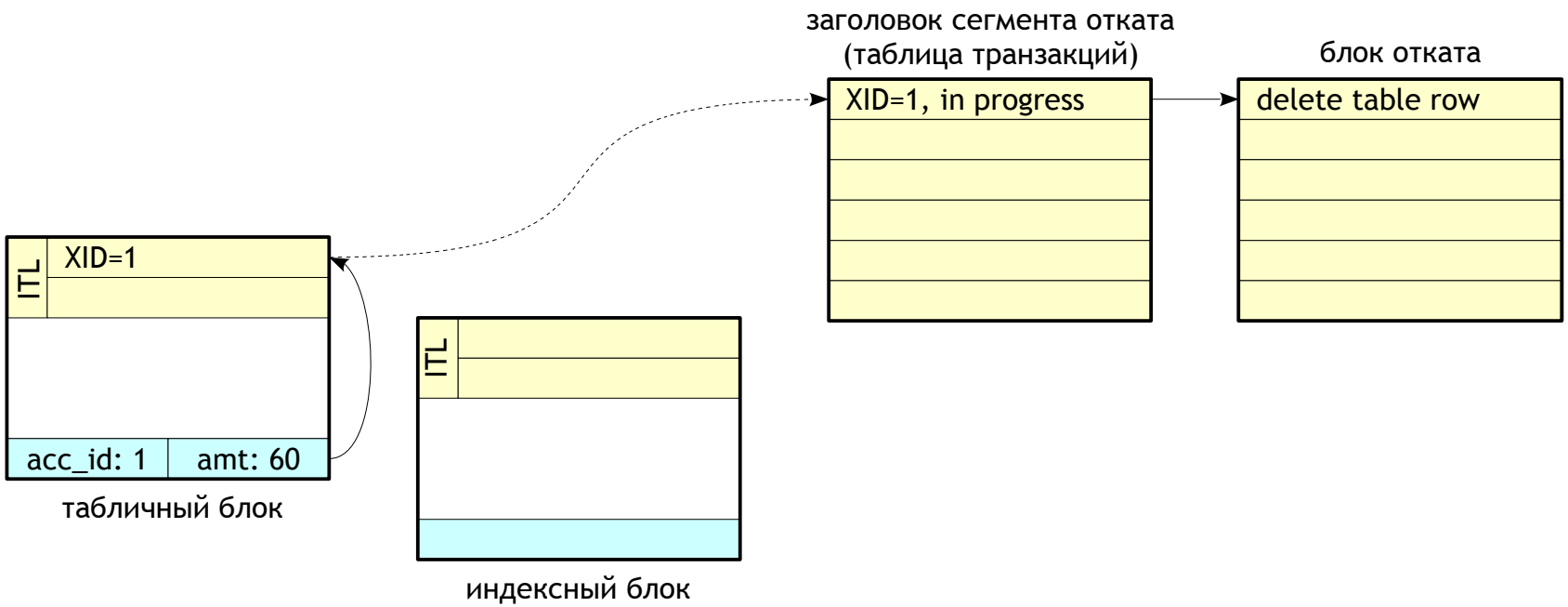
insert into t(acc_id, amount) values (1, 60);



Oracle

Как это выглядит на низком уровне?

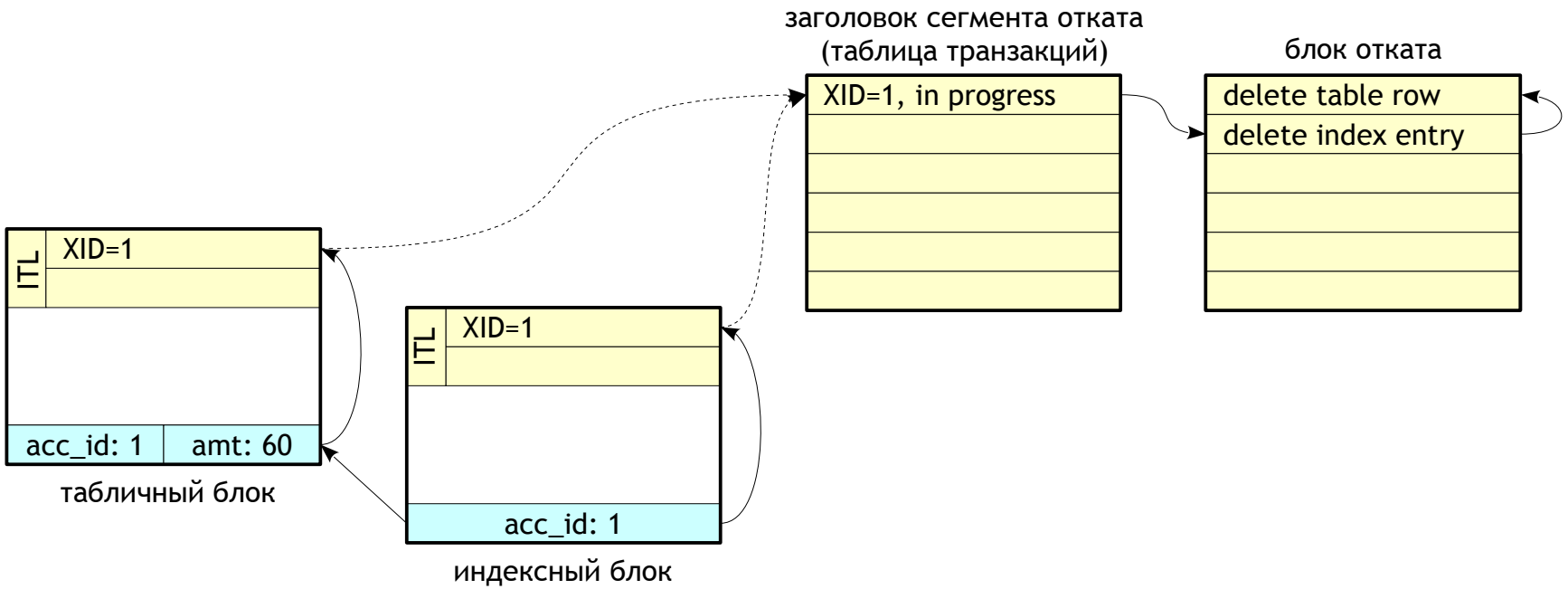
insert into t(acc_id, amount) values (1, 60);



Oracle

Как это выглядит на низком уровне?

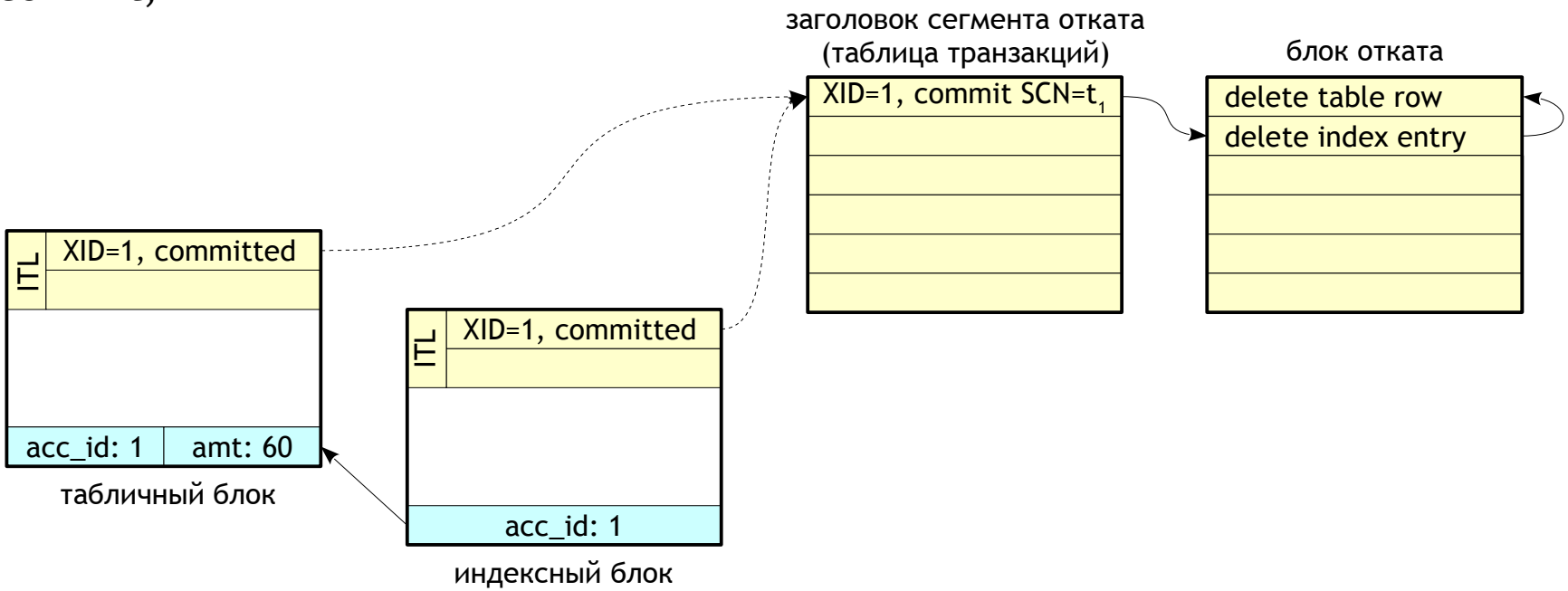
insert into t(acc_id, amount) values (1, 60);



Oracle

Как это выглядит на низком уровне?

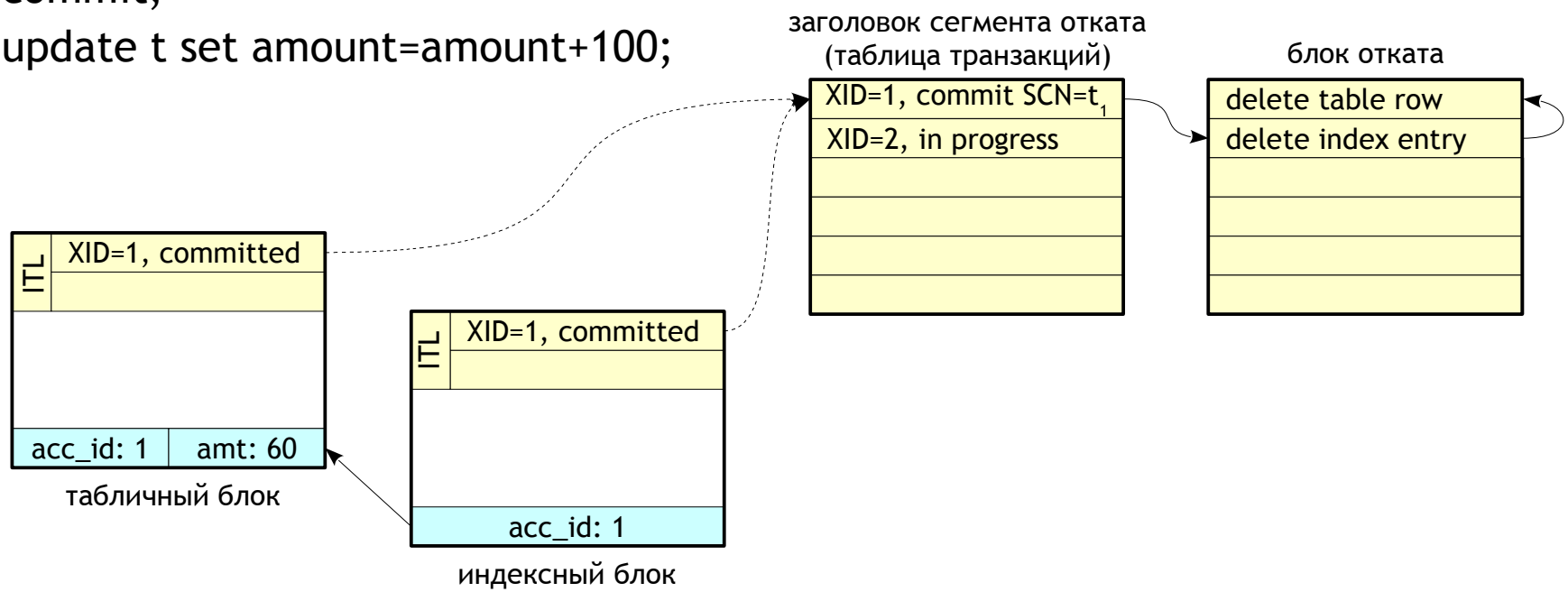
insert into t(acc_id, amount) values (1, 60);
commit;



Oracle

Как это выглядит на низком уровне?

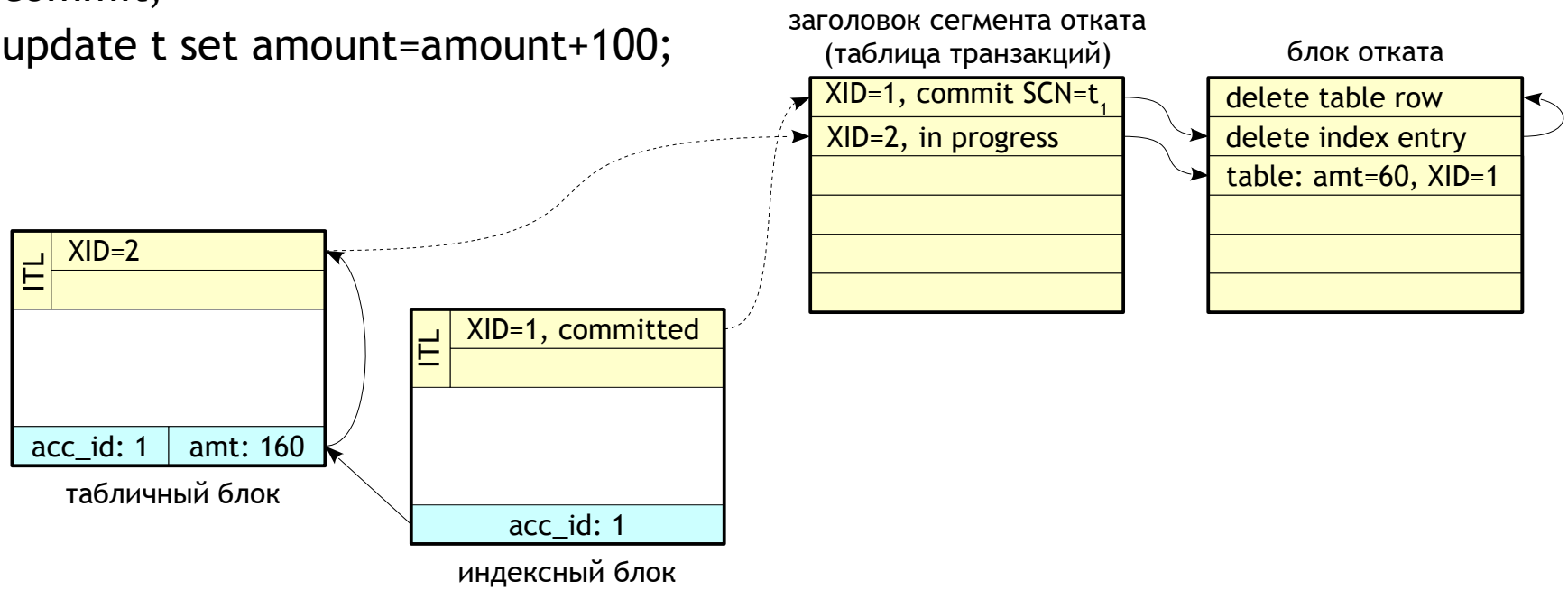
```
insert into t(acc_id, amount) values (1, 60);
commit;
update t set amount=amount+100;
```



Oracle

Как это выглядит на низком уровне?

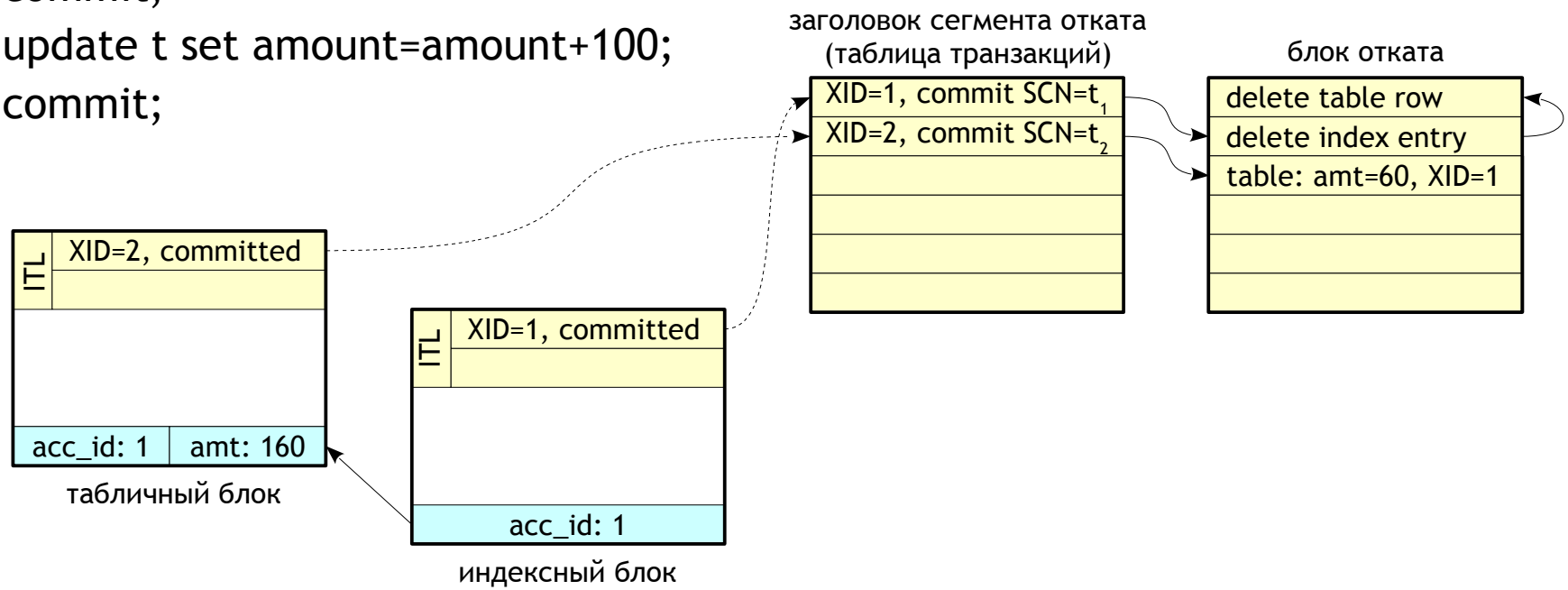
```
insert into t(acc_id, amount) values (1, 60);
commit;
update t set amount=amount+100;
```



Oracle

Как это выглядит на низком уровне?

```
insert into t(acc_id, amount) values (1, 60);
commit;
update t set amount=amount+100;
commit;
```



Поддержка уровней изоляции

PostgreSQL
- read committed
- repeatable read
- serializable (дополнительные предикатные блокировки)

Oracle
- read committed
- «serializable» (= PostgreSQL repeatable read)

Выводы

Как Oracle, так и PostgreSQL обеспечивают эффективное управление транзакциями

Две системы по-разному подходят к реализации одного и того же механизма многоверсионности

Обе реализации имеют свои особенности, о которых стоит знать

Прикладной код во многом определяется уровнем изоляции

PostgreSQL дает разработчикам наиболее широкий выбор

Спасибо за внимание!

Вопросы?

Егор Рогов, Postgres Professional
e.rogov@postgrespro.ru