



Бази даних та інформаційні технології

Тема 15. Транзакції

СумДУ, каф.КН
2020

Зміст

- ▶ CAP-теорема
- ▶ ACID
- ▶ Транзакції та SQL

Навіщо потрібні транзакції

За мотивами: [CAP-теорема простым, доступным языком](#) by Артем Попов

Идея: сервис – «Позвони, напомним!» — «Никогда не забывайте, даже если вы не помните, что забыли!»

моя записная книжка как БД

Первый звонок, первый клиент

- Я Джон, хочу потом посмотреть "Начало" ...

... время ... новый звонок ...

- Я Джон, напомнимки есть? - *Ага, вы хотели "Начало" посмотреть. О, спасибо!*

... народу нравится..., растём ...

... даже скорее так: **РАСТЁМ!** :)

Неприятности в раю

растем ... но народ уже в очереди висит ... *не справляюсь!*
... вчера болел - вообще никого не обслужил
вчерашний народ рассержен, да еще и знакомым рассказывает
... ????

... Надо что-то делать!!!

Решение №1.

Дорогая, ты видела какую я штуку сделал?
Поможешь?! Вливайся!
Вот твой стол, твоя книжка, твоя гарнитура.
Я поднял (микро АТС - единый входной номер - **балансер**)
... полетели!!!
... в первый же день обслужили в *два раза* больше клиентов.

Неприятности в раю, дубль 2

~~Звонок: Я Джим, напомни есть сегодня встречи?~~

... посмотрел, говорю "Нет, нету, спасибо за ваш звонок!"

Это я, Джим, из-за вас я пропустил визит к стоматологу!!!!

анализ: когда Джим оставлял инфу, он попал не ко мне, а к жене в моей книге записи нет ...

Смотрим книгу жены - ага, человеку не повезло.

Система not consistent (не согласована) - показания не сходятся

Решение несогласованности

заводим протокол (оговоренная последовательность шагов и процедур):

до того, как сказать "Спасибо, до связи!" мы записываем данные друг другу

когда получаем звонок - все будет хорошо, ведь все данные есть у всех -
правильный быстрый ответ!

... вот оно РЕШЕНИЕ !!!

Неприятности в раю, дубль 3

проблема: на время синхронизации записи - всем приходится прерваться и потормозить - клиенты в очереди ждут, те, кто сейчас на линии злятся на отрывающегося оператора

... да не проблема - все равно большую часть все ищут информацию - а тогда мы отвечаем *быстро!*

... прерывание всех ... синхронность ...

протокол надо соблюдать *всегда*, а что, если кто-то **недоступен** (телефон, заболел, в магазин отошел)?

наш сервис unavailable - висит "подождите, я щас подожду пока смогу передать вашу информацию...", а потом клиент бросает трубку

.. надо что-то делать ...

► Consistent + Available

идея: если человек рядом - мы ему говорим (== *быстро!*), а если нет – шлем mail (*и ответили, и надежно*), он утром прочитает, внесет в свою книгу (*ответ согласован!*).

... и наступило у нас долгое счастье...

А кто сказал, что это сказка с хорошим концом?

Потеря коммуникаций: жена разговаривать не хочет.

Уже два дня! ... письма пишу, а она их не читает ...

А работать надо!!! ... и она тоже работает - есть несогласованные ответы!

произошло разделение сети (partition) и наши апдейты "не доходят".

Когда-то базы будут синхронизированы, но когда?

► Итоги

Сервис - одна точка входа, один процессор, одна база

... упало - ничего не работает

Делаем "два" мастера - один упал, второй работает

... разъехалось состояния (consistency)

Делаем синхронный протокол - отвечаем согласованно

... не можем принять апдейт когда *всех* нет

Даешь асинхронность - можно принять апдейт всегда

... есть шанс, что до всех еще не дошло (eventually)

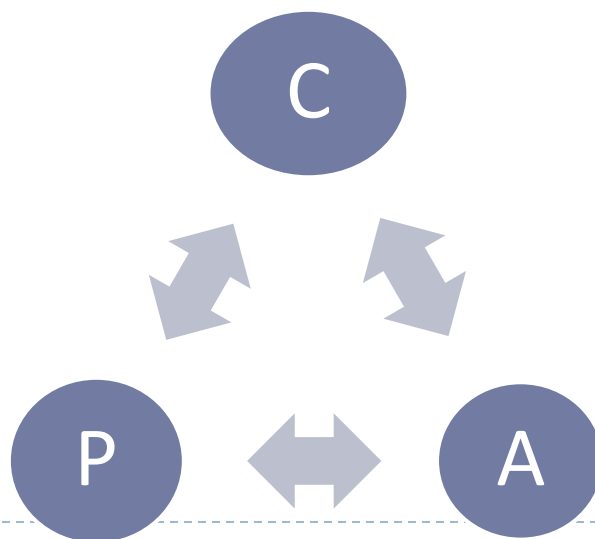
CAP-теорема

Розподілена система не може гарантувати одночасного виконання наступних трьох властивостей:

Consistency. **Узгодженість** - всі користувачі в будь-який момент часу бачать однакові дані;

Availability. **Доступність** - при виході з ладу будь-яких вузлів, вузли що залишилися продовжують функціонувати;

Partition (tolerance). **Стійкість до розділу** - при розпаді системи на окремі групи вузлів через збій мережі, кожна група продовжує працювати.



Транзакція

- ▶ Це неподільна послідовність дій, яка переводить базу даних з одного несуперечливого стану в інший несуперечливий стан.
- ▶ Гарантує збереження цілісності бази даних.

Транзакції у реальному житті:

- ▶ миттєвий перехід зі стану "до" в стан "після" - ніхто не повинен бачити процес переходу (як сталося).
- ▶ кілька дій за один захід: зв'язані рахунки - з одного списали, на інший записали.
- ▶ складний стан, що змінюється - не тільки дані, а ще журнали, індекси, лічильники, ...

ACID / АСИД

Транзакція повинна відповідати принципам ACID:

Atomicity = Атомарність

Consistency = Узгодженість / Коректність

Isolation = Ізольованість / Невидимість

Durability = Довговічність / Стійкість

“Дій багато, але я покажу або все, або нічого;

Коли закінчу - все буде ОК, все буде підчищено;

Ви працюйте, а я вам своє поки не покажу, у мене своя копія світу;

Якщо сказав зроблено, значить не пропаде, все записано надійно.”

Атомарність

- ▶ Транзакція неподільна.
- ▶ Атомарність гарантує, що будь-яка транзакція буде зафіксована тільки цілком (повністю) - або все, або нічого.
- ▶ Якщо одна з операцій в послідовності не буде виконана, то вся транзакція буде скасована. Тут вводиться поняття "відкату" (**rollback**).

Узгодженість

Після виконання транзакції всі дані повинні перебувати в узгодженому стані, зберігається логічна цілісність даних.

- ▶ внутрішньо (індекси, формат етс).
- ▶ зовні - все суми пораховані, все зовнішні ключі проставлені.

Ізольованість

Транзакція ізольована, оскільки її результати самодостатні. Незакінчена (непідтверджена) транзакція повинна бути невидима ззовні.

Результати транзакції стають доступні для інших транзакцій тільки після її фіксації.

- ▶ свій маленький світ;
- ▶ я бачу свої зміни;
- ▶ я не бачу змін сусіда;
- ▶ початок часів;
- ▶ ілюзія послідовності подій;
- ▶ рівні ізоляції.

ДОВГОВІЧНІСТЬ

Транзакція стійка (довговічна), її дія постійна навіть при збої системи.


Після завершення транзакції, внесені зміни повинні стати доступні всім і стають постійними.

- ▶ я сказав "ок, значить ок"
 - ▶ світло вимкнули
 - ▶ винт обсипався
 - ▶ у вогні не горить, у воді не тоне
- ▶ Приклад: журнал реєстрації операцій з нерухомістю - з печаткою, пронумеровані, прошиті

Рівні ізоляції

Під «рівнем ізоляції транзакцій» розуміється ступінь захисту від різних видів неузгодженості даних, що виникають при паралельному виконанні транзакцій, яка забезпечується внутрішніми механізмами СУБД

► Згідно ANSI SQL існують 4 рівня ізоляцій транзакцій :

- 
1. **READ UNCOMMITTED** / dirty read — Читання непідтверджених даних (брудне читання)
 2. **READ COMMITTED** - Читання підтверджених даних
 3. **REPEATABLE READ** / phantom read - Повторюване читання
 4. **SERIALIZABLE** - Впорядкований

В ORACLE - 2 рівня:

READ COMMITTED - за замовчуванням
SERIALIZABLE (режим READ ONLY)

Висновки

- ▶ Читати:

- ▶ Дом на песке Пэт Хелланд, Дейв Кэмпбел

- ▶ ACID:

Atomicity = Атомарність

Consistency = Узгодженість / Коректність

Isolation = Ізольованість / Невидимість

Durability = Довговічність / Стійкість

Транзакції в «живу»

- ▶ Транзакції у Oracle складаються з однієї з наступних альтернатив:
 - ▶ **DML**–операторів, що складають одну послідовну зміну даних.
(Наприклад, переказ коштів між двома рахунками повинен включати дебет на одному рахунку та кредит на іншому рахунку тієї ж суми. Обидві дії або потерплять невдачу, або здійсняться вдало разом; кредит не повинен бути наданий без дебету);
 - ▶ Одного **DDL**–оператора;
 - ▶ Одного **DCL**–оператора.



DML, DDL, DCL

DML

- SELECT
- INSERT
- UPDATE
- DELETE

DDL

- CREATE
- ALTER
- DROP
- RENAME
- TRUNCATE

DCL

- GRANT
- REVOKE

Початок і закінчення транзакції

- ▶ Транзакція **починається** перед виконанням будь-якого **DML**-запиту.
- ▶ Транзакція **закінчується** якщо:
 - ▶ Виконана команда **COMMIT** або **ROLLBACK**;
 - ▶ Виконана будь-яка **DDL** або **DCL** команда (авто-commit);
 - ▶ Користувач залишає SQL * Plus;
 - ▶ У випадку системної помилки.



Можливості виразів COMMIT і ROLLBACK

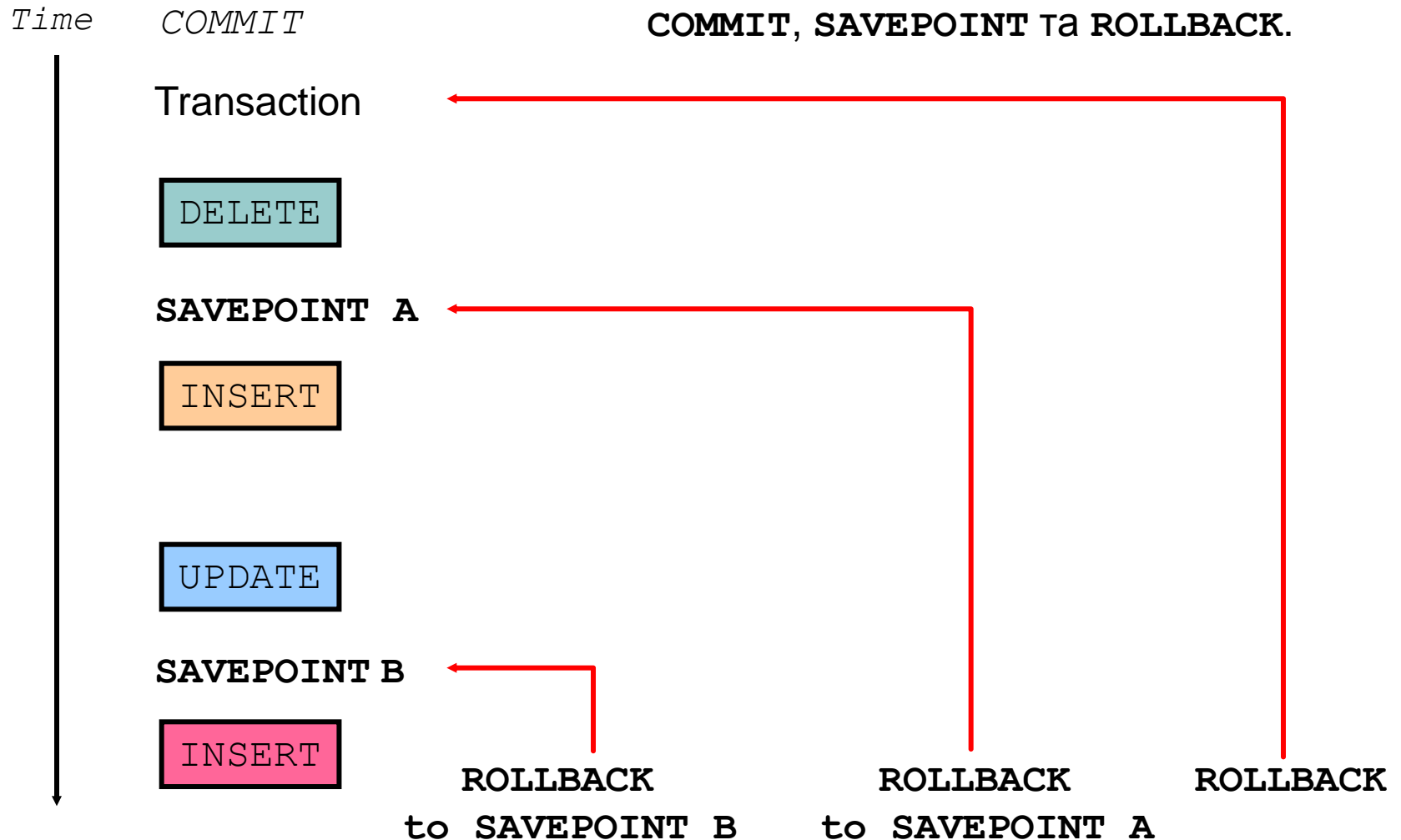
Використовуючи **COMMIT** і **ROLLBACK** ви можете:

- ▶ бути впевнені, що інформація консистентна;
- ▶ переглянути зміни, до того як вони вступили в дію;
- ▶ згрупувати логічно пов'язані операції.



Управління транзакціями

Ви можете керувати логікою транзакцій,
використовуючи оператори
COMMIT, **SAVEPOINT** та **ROLLBACK**.



Скасування змін

- ▶ Створення маркера (точки збереження) для поточної транзакції – **SAVEPOINT**.
- ▶ Повернення до маркера - **ROLLBACK TO SAVEPOINT**.

```
UPDATE . . .
```

```
SAVEPOINT update_done;
```

```
SAVEPOINT update_done succeeded.
```

```
INSERT . . .
```

```
ROLLBACK TO update_done;
```

```
ROLLBACK TO succeeded.
```

Якщо ви створюєте другий маркер з тим самим ім'ям, що і попередній, то попередній маркер видаляється.



Неявне управління транзакціями

- ▶ Автоматичний **commit** відбувається, якщо:
 - ▶ Виконується **DDL** – оператор;
 - ▶ Виконується **DCL** – оператор;
 - ▶ Користувач виходить з SQL * Plus
- ▶ Автоматичний відкат відбувається якщо:
 - ▶ Відбулася системна помилка;
 - ▶ Користувач екстрено виходить з SQL * Plus.



Дані до COMMIT або ROLLBACK

- ▶ Попередній стан інформації може бути відновлений;
- ▶ Поточний користувач може бачити результати **DML**-операторів за допомогою **SELECT**;
- ▶ Інші користувачі не можуть бачити результати **DML**-операторів поточного користувача;
- ▶ Змінені рядки **блокуються** і не можуть бути змінені іншими користувачами.



Дані після COMMIT

- ▶ Інформація збережена в БД;
- ▶ Попередній стан даних більше не доступний для звичайних SQL-запитів;
- ▶ Всі користувачі можуть бачити зміни;
- ▶ Блокування з усіх рядків знімаються, рядки тепер доступні для інших користувачів, щоб виконати нові зміни даних;
- ▶ Всі маркери (**savepoint**) стираються.



COMMIT

► Внесення змін

```
DELETE FROM emp  
WHERE empno = 99999;
```

```
1 rows deleted
```

```
INSERT INTO dept  
VALUES (290, 'Corporate Tax', NULL, 1700);
```

```
1 rows inserted
```

► Збереження змін:

```
COMMIT;
```

```
COMMIT succeeded.
```



Данні після ROLLBACK

- ▶ Після **ROLLBACK** всі дані повертаються до первинного стану:
 - ▶ Всі внесені зміни скасовуються;
 - ▶ Попередні дані відновлюються;
 - ▶ Знімаються блокування зі змінених рядків.

```
DELETE FROM copy_emp;  
ROLLBACK ;
```



Приклад

```
DELETE FROM test;  
25,000 rows deleted.
```

```
ROLLBACK;  
Rollback complete.
```

```
DELETE FROM test WHERE id = 100;  
1 row deleted.
```

```
SELECT * FROM test WHERE id = 100;  
No rows selected.
```

```
COMMIT;  
Commit complete.
```



Відкат на рівні запиту

- ▶ Якщо один **DML**-оператор завершується з помилкою, то здійснюється відкат тільки цієї дії.
- ▶ Oracle server реалізує це через неявні мітки.
- ▶ Всі зміни попередніми **DML**-операторами залишаються.
- ▶ Явно завершити транзакцію можна використовуючи **COMMIT** або **ROLLBACK**.



Реалізація узгодженості читання в Oracle

- ▶ Операції читання не порушують консистентності даних.
- ▶ Операції зміни даних не конфліктують між собою.
- ▶ Якщо кілька користувачів звертаються до одних і тих же даних, то:
 - ▶ Операції читання не очікують операцій запису;
 - ▶ Операції читання не очікують операцій читання;
 - ▶ Операції запису очікують завершення операцій запису.

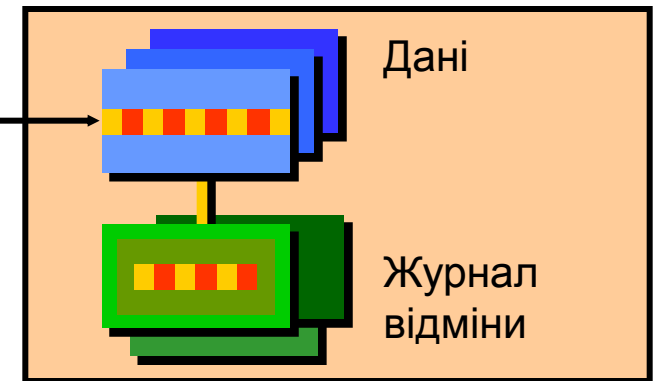


Реалізація узгодженості читання

User A

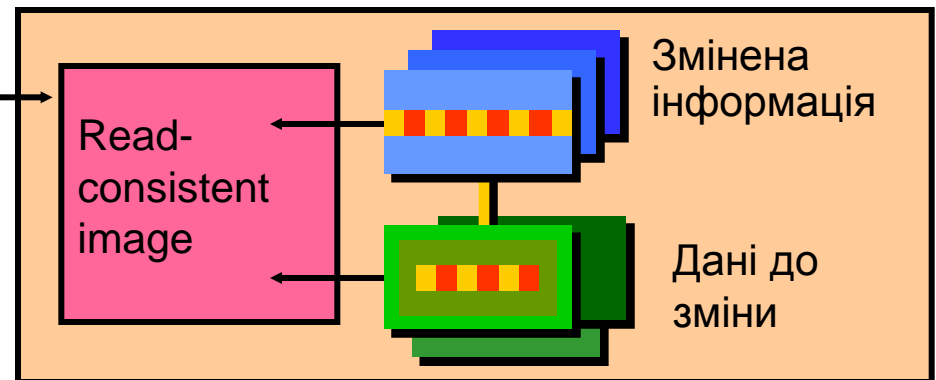


```
UPDATE emp  
SET     sal = 7000  
WHERE   ename = 'Grant';
```



```
SELECT *  
FROM userA.emp;
```

User B



Вираз FOR UPDATE в SELECT

- ▶ Заблокувати рядки з EMP де sal > 2000.

```
SELECT ename, sal  
FROM emp  
WHERE sal > 2000  
FOR UPDATE  
ORDER BY empno;
```

- ▶ Блокуються записи "лише для ваших змін"
- ▶ Блокування знімається через **ROLLBACK** або **COMMIT**.
- ▶ Якщо інший запит намагається заблокувати рядки, то він очікує поки перший запит їх не розблокує



Приклад FOR UPDATE

- ▶ Використовуючи **FOR UPDATE** можна заблокувати декілька таблиць.

```
SELECT e.empno, e.sal  
FROM emp e JOIN dept d  
USING (deptno)  
WHERE job = 'CLERK'  
AND loc = 'HONKONG'  
FOR UPDATE  
ORDER BY e.empno;
```

- ▶ Используя **FOR UPDATE OF *column_name*** можно более точно установить для строк каких из таблиц устанавливаться блокировка.



Deadlock

- ▶ Взаємне блокування (**deadlock**) - ситуація, при якій двоє або більше сеансів знаходяться в стані нескінченного очікування ресурсів, захоплених самими цими ж сеансами.
- ▶ Коли один з сеансів захоплює будь-якої ресурс, інші сеанси чекатимуть його звільнення, шикуючись в чергу один за одним.

Deadlock

Транзакція А	Транзакція В
UPDATE emp SET ename = 'test2' WHERE empno = 7698; 1 row updated.	
	UPDATE emp SET ename = 'test3' WHERE empno in (7698,7839);
UPDATE emp SET ename = 'test2' WHERE empno = 7839;	
	ERROR at line 1: ORA-00060: deadlock detected while waiting for resource

Оператор Flashback Table

- ▶ **Flashback** дозволяє повернути таблицю до стану в конкретний момент часу однією командою.
- ▶ **Flashback** автоматично відновлює пов'язані індекси і обмеження.
- ▶ В якості точки повернення може бути точка в часі або номер зміни бази (System Change Number, SCN).

Використання Flashback Table

► Синтаксис:

```
FLASHBACK TABLE [schema.]table [, [schema.]table ...]  
TO {TIMESTAMP | SCN } expr  
[{ENABLE | DISABLE} TRIGGERS];
```

► Особливості:

- У вас повинні бути права на: Select, Update, delete Alter.
- Не можна зробити Flashback Flashback'а.

Приклад використання

- ▶ Видалимо таблицю

```
DROP TABLE emp2;
```

- ▶ Перевіримо її наявність у кошику:

```
SELECT original_name, operation, droptime  
FROM recyclebin;
```

ORIGINAL_NAME	OPERATION	DROPTIME
-----	-----	-----
EMP2	DROP	2013-01-13 00:00:00

- ▶ Відновимо таблицю:

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
```

- ▶ або очистимо кошик:

```
PURGE RECYCLEBIN;
```

Flashback:

- ▶ Oracle Database відновить всі індекси за винятком **bitmap join і domain** індексів.
- ▶ Будуть відновлені всі індекси і обмеження, за винятком обмежень пов'язаних з іншими таблицями.
- ▶ До того, як індекси, тригери і обмеження будуть остаточно видалені, вони будуть розміщені в кошик. Їх імена доступні в **USER_RECYCLEBIN**.
- ▶ Використовуючи **TABLE ... TO BEFORE DROP** можна перейменувати об'єкти при відновленні.
- ▶ При видаленні таблиці також видаляються всі представлення. **Flashback** їх не відновлює.
- ▶ Згодом об'єкти в кошику видаляються, починаючи з індексів. Тому відновлюючи таблицю не гарантовано, що ви отримаєте всі об'єкти.
- ▶ Ключове слово **PURGE** при видаленні робить неможливим відновлення таблиці.

Flashback даних: підготовка

```
SELECT sal  
FROM emp1  
WHERE ename = 'KING';
```

SAL

3800

```
UPDATE emp  
SET sal = 4000  
WHERE ename = 'KING';  
1 row updated.
```

```
SELECT sal  
FROM emp1  
WHERE ename = 'KING';
```

SAL

4000

Flashback в дії

```
SELECT sal  
FROM emp  
AS OF TIMESTAMP  
    (SYSTIMESTAMP - INTERVAL '1' MINUTE)  
WHERE ename = 'KING';
```

```
SALARY  
-----  
3800
```

Flashback в дії

```
SELECT sal
FROM emp
VERSIONS BETWEEN TIMESTAMP
        SYSTIMESTAMP - INTERVAL '10' MINUTE
        AND SYSTIMESTAMP - INTERVAL '1' MINUTE
WHERE ename = 'KING';
```

Flashback в дії

```
UPDATE emp
SET sal = ( SELECT sal
            FROM emp
            AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '2' MINUTE)
            WHERE ename = 'KING')
WHERE ename = 'KING';
1 row updated.
```

```
SELECT sal
FROM emp
WHERE ename = 'KING';
```

SALARY

3800

Підсумки

- ▶ Поняття транзакції
- ▶ SAVEPOINT, COMMIT, ROLLBACK
- ▶ К каким точкам можно откатиться, а к каким нет
- ▶ FLASHBACK