Transaktionen

Locking & Multiversioning

Transaktionen / Isolationslevel

*One of the key challenges in developing **multiuser, database-driven** applications is to **maximize concurrent access** and, at the same time, ensure that each user is able to **read and modify the data in a consistent fashion**.*

Tom Kayle, Expert Oracle Database Architecture, 2nd Edition

SQL-Anweisungen

In Oracle laufen alle Statements automatisch in einer Transaktion

COMMIT TRANSACTION

Transaktion wird erfolgreich beendet gekennzeichnet. Alle zugehörigen Datenbankänderungen werden festgeschrieben.

Syntax: **commit [work];**

ROLLBACK TRANSACTION

Transaktion wird explizit (per Programm) oder implizit (z.B. durch Connection-Verlust) abgebrochen. Alle zugehörigen Datenbankänderungen werden rückgängig gemacht und auf den letzten konsistenten Zustand des Systems zurückgesetzt (**Checkpoints / Savepoints**).

Syntax: **rollback [work];**

ACID

Atomicity (Atomarität)

Transaktionen haben atomaren Charakter: Sie werden ganz oder gar nicht ausgeführt („alles oder nichts“).

Consistency (Konsistenz)

Transaktionen bewahren die Konsistenz der Datenbank. Die Datenbank wird durch eine Transaktion von einem konsistenten Zustand in den nächsten überführt.

Isolation (Isolation)

Transaktionen werden bei konkurrierendem Zugriff (concurrency) untereinander getrennt, d.h. jede Transaktion läuft in einem simulierten Single-User-Betrieb.

Durability (Dauerhaftigkeit)

Datenbank-Updates bleiben nach einem Commit dauerhaft erhalten, auch wenn nach diesem Commit ein Systemausfall stattgefunden haben sollte.

Problem: Lost Update

T1: read x=100

T2: read x=100

T1: buche 1 Platz: x:=x-1=99

T1: **update ... set x = 99;**

T2: buche 2 Plätze: x:=x-2=98

T2: **update ... set x = 98;**

Problem: Dirty Read

T1: read x=100

T1: buche 1 Platz: x:=x-1=99

T1: **update ... set x = 99;**

T2: read x=99

T1: **rollback**

Problem: Nonrepeatable Read

T1: read **x=100**

T2: read x=100

T2: buche 2 Plätze: x:=x-2=98

T2: **update ... set x = 98;**

T1: read **x=98**

Problem: Phantom Read

T1: count = 0

T1: read LH 4225: count = 1

T1: read KL 1775: count = 2

T2: insert into ...

values (‘AF 1018‘,250, ...);

T1: read BA 914: **count = 3 (≠ 4)**

T1: SELECT count(*) → 4

Probleme - Zusammenfassung

Lost Update

Das zweite Update überschreibt das erste.

Dirty Read

Un-committed Daten werden angezeigt.

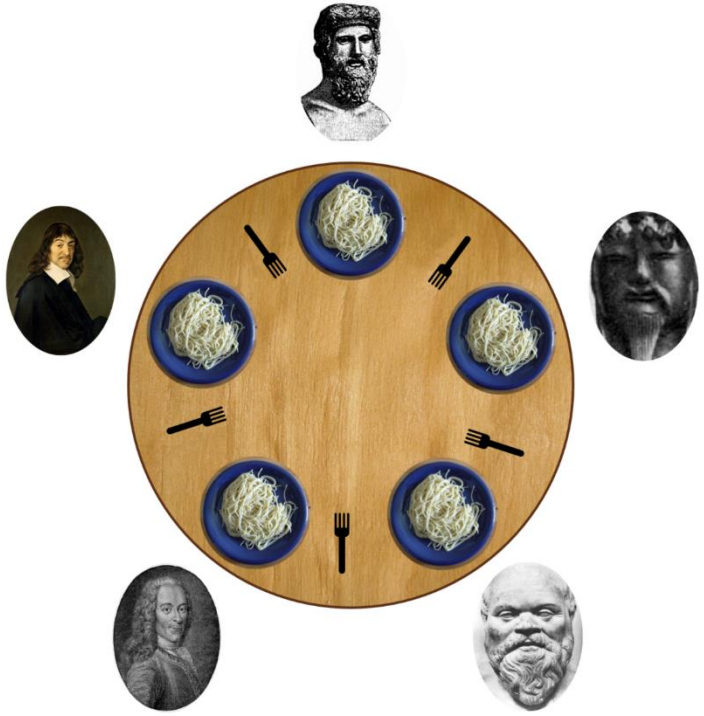
Non-Repeatable Read

Die selbe Zeile liefert unterschiedliche Attribute.

Phantom Read

Das selbe SELECT liefert eine unterschiedliche Anzahl an Zeilen.

Lösungsansätze

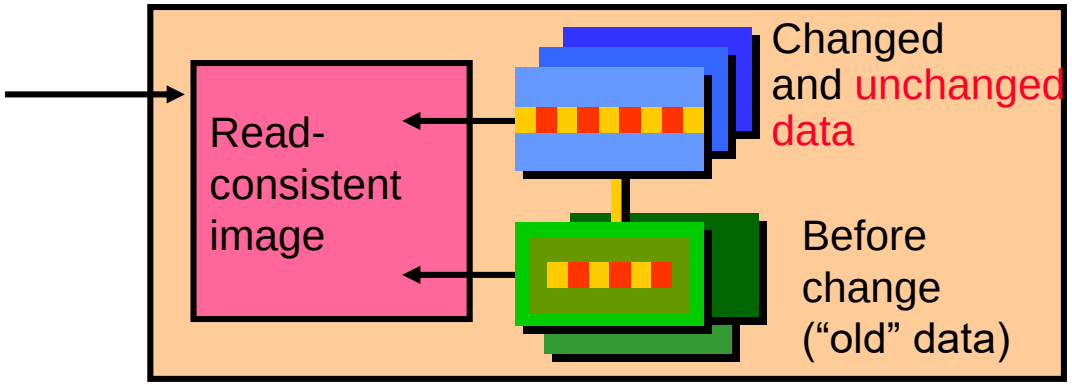


Sperren (locks), die andere blockieren (blocking)

- Pessimistisch (pessimistic)
- Problem: Deadlocks

Jeder bekommt seine Version (versioning)

- Optimistisch (optimistic)
- Problem: gleichzeitiges UPDATE



Sperren (Locks)

Mechanismus, um

- konkurrierenden Zugriff auf
- gemeinsame Ressource zu steuern

Ressourcen

- Zeilen einer Tabelle, Stored Procedure, Speicherbereiche, …

Sperren (Locks)

Pessimistische Sperren

```
select empno, ename, sal from emp
where deptno = 10;
```

```
select empno, ename, sal
from emp
where empno = :empno
  and ename = :ename
  and sal =   :sal
for update nowait
```

```
update emp
set ename = :ename, sal = :sal
where empno = :empno;
```

Optimistische Sperren

```
select empno, ename, sal from emp
where deptno = 10;
```

```
update emp
set ename= :ename, sal= :sal
where empno = :empno
  and ename = :ename
  and sal =   :sal
```

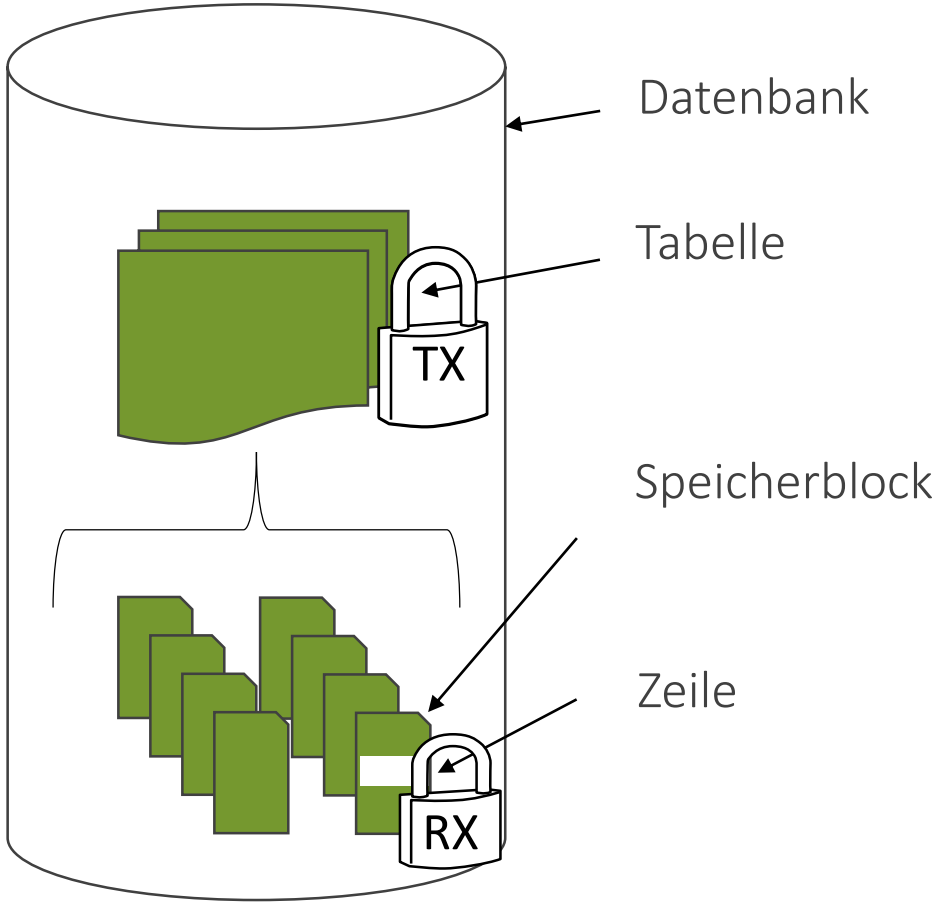
Locking in Oracle

- Sperren auf Satz-Ebene
- Keine Eskalation auf höhere Ebene
  - Jedoch lock conversion (auch: lock promotion)
- Sperren auf Tabellenebene
  - Bei Manipulationen einer Parent-Tabelle wird uU. Child-Tabelle gesperrt
- SHARED vs. EXCLUSIVE
  - SHARED LOCK erlaubt gleichzeitig andere SHARED LOCKs
  - SHARED LOCK erlaubt gleichzeitig **keinen** EXCLUSIVE LOCK
  - EXCLUSIVE LOCK erlaubt gleichzeitig **keinen** anderen EXCLUSIVE LOCK

Lock Conversion

```
SELECT  EmpNo, Street
FROM EMP
WHERE Street= 'Limesstraße 10’
AND EMPNO=2
FOR UPDATE
```

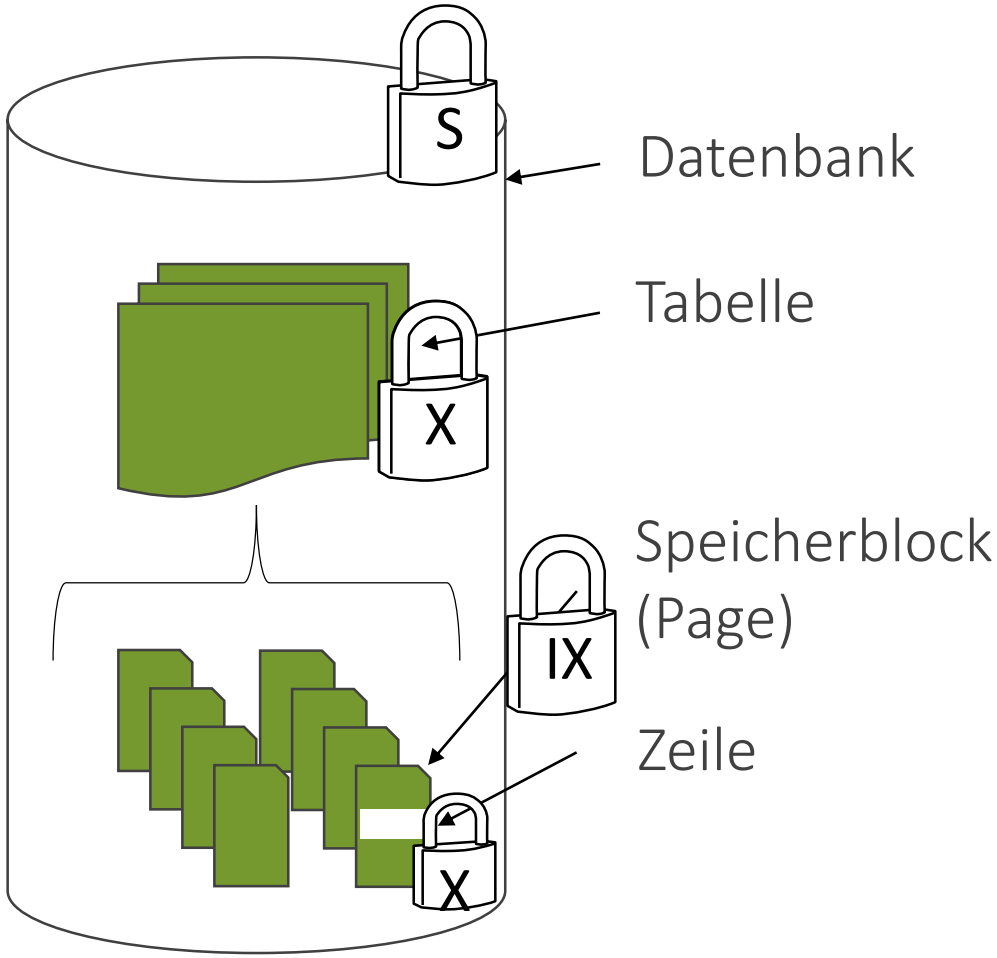
```
UPDATE EMP
SET Street= 'Limesstraße 10'
WHERE EMPNO=2
```



Lock Escalation

```
UPDATE [EMP]
SET AddressLine1= 'Leonding, Austria'
WHERE EMPNO=2
```

```
UPDATE [EMP]
SET AddressLine1= 'Leonding, Austria'
WHERE EMPNO>3
```



Lock Typen

DML Lock

- SELECT, INSERT, UPDATE, MERGE, DELETE

DDL Lock

- CREATE, ALTER, …

Interne Locks und Latches

- Query Plan, geteilte Speicherbereiche, …

Locking vs. Blocking

Locking

- Sperren veranlassen und halten
- Pessimistic Concurrency Control

Blocking

- Die Konsequenz aus Locking
- Ein Prozess wartet darauf, dass ein anderer einen Lock freigibt
- Wird nur zum Problem, wenn es zu lange dauert

Ursachen für Blocking

SELECT FOR UPDATE

INSERT

UPDATE

DELETE

MERGE

Ursachen für Blocking
SELECT FOR UPDATE

Ausgewählte Zeile wurde bereits gesperrt

Lösung

- NOWAIT angeben

Ursachen für Blocking
INSERT

Einfügen mit gleichem Primary Key

Einfügen mit einem Fremdschlüssel

- Dessen Primary Key-Satz gerade eingefügt/gelöscht wird

Lösung

- Primary Keys automatisch generieren

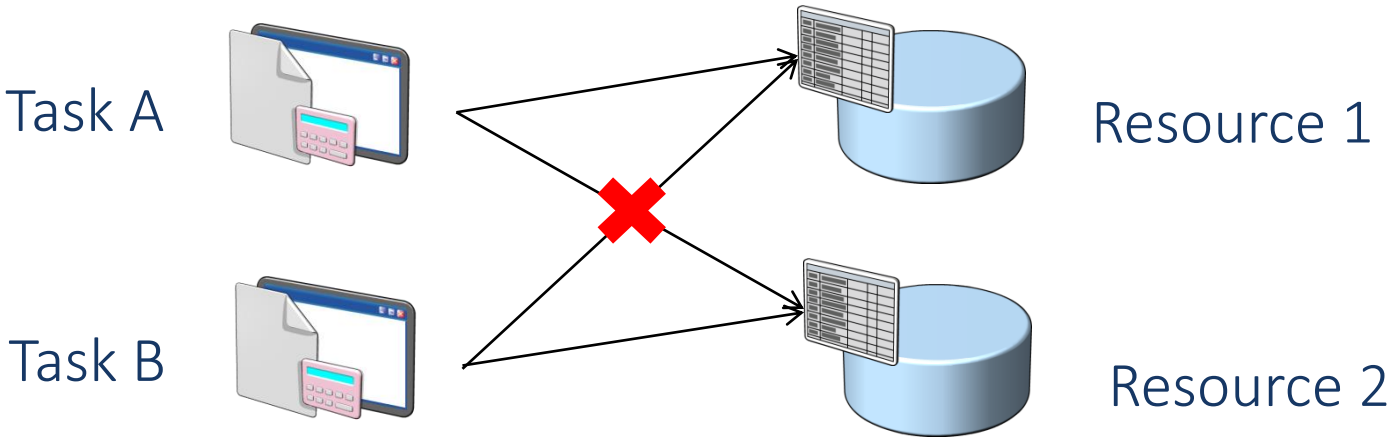
Ursachen für Blocking
UPDATE, MERGE, DELETE

Ändern/Löschen einer Zeile, die schon jemand anderer manipuliert hat

Lösung

- Zeile sperren (pessimistisch)
- Vorm Ändern/Löschen prüfen, ob Zeile geändert wurde (optimistisch)

Deadlock



Gegenseitiges blockieren

Wird von DBMS erkannt

- Ein Task “verliert” und es wird ein ROLLBACK durchgeführt

Deadlock vs. Timeout

Versionierung

- Versionsspalte
  - TIMESTAMP, Hash-Checksum
- Verwaltung durch
  - Jede Anwendung selber
  - Trigger
  - Procedure

Multi-Versioning

Simultane, multiple Versionen einer Zeile

Lese-Zugriffe werden nie geblockt

- Aber geänderte Zeilen müssen zuerst rekonstruiert werden (undo log)

Level

- Statement-Level (Default)
- Transaktions-Level

Blocking lösen

- Transaktionen kurz halten
  - Nur wirklich relevante Aktionen innerhalb der Transaktion
  - KEINE Benutzerinteraktion während Transaktion
  - Query-Tuning, Indizes, Lastverteilung auf mehrere Server
- Isolation Level mit Bedacht wählen

Isolationslevel (ISO-92)

READ UNCOMMITTED

READ COMMITTED

REPEATABLE READ

SERIALIZEABLE

READ UNCOMMITTED

Erlaubt dirty reads

- Und sämtliche andere anfangs besprochenen Probleme

Ziel: non-blocking reads

READ COMMITTED

Nur Sätze, die committed sind werden gelesen

- Verhindert dirty reads

Non-repeatable reads & phantom reads sind möglich

REPEATABLE READ

Ziel: lost updates verhindern

- IdR. durch Shared Read Locks, die simultane Updates ausschließen

SERIALIZABLE

Restriktivster Isolationslevel

Transaktion bleibt völlig unberührt von allen anderen

Isolation Level (SQL 92)

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read Uncommitted	Möglich	Möglich	Möglich
Read Commited	**Nicht möglich**	Möglich	Möglich
Repeatable Read	**Nicht möglich**	**Nicht möglich**	Möglich
Serializable	**Nicht möglich**	**Nicht möglich**	**Nicht möglich**

Isolation Level (Oracle)

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
~~**Read Uncommitted**~~	~~Möglich~~	~~Möglich~~	~~Möglich~~
Read Commited	**Nicht möglich**	Möglich	Möglich
~~**Repeatable Read**~~	~~**Nicht möglich**~~	~~**Nicht möglich**~~	~~Möglich~~
Serializable	**Nicht möglich**	**Nicht möglich**	**Nicht möglich**
Read Only	**Nicht möglich**	**Nicht möglich**	**Nicht möglich**

Read committed vs. Serializable

Read committed

- Konsistenz zum Zeitpunkt des Startes der Abfrage
- Nur abgeschlossene (committed) Ergebnisse anderer Transaktionen

Serializable

- Konsistenz zum Zeitpunkt des Startes der Transaktion
- Keine Änderungen durch andere Transaktionen während aktueller Transaktion

Read only

- Konsistenz zum Zeitpunkt des Startes der Transaktion
- Keine Änderungen durch andere Transaktionen sichtbar
- Keine Änderungen durch eigene Transaktion erlaubt

Nachteile von SERIALIZABLE und READ ONLY

SERIALIZABLE

- Kein UPDATE von Zeilen, die außerhalb der aktuellen Transaktion geändert wurden
- ORA-08177: can't serialize access for this transaction
- ORA-01555 snapshot too old error

READ ONLY

- Kein UPDATE von Zeilen
- ORA-01555 snapshot too old error

Transaktions Logging

Redo

- Transaktion im Zuge eines Recovery wiederholen

Undo

- Rollback einer Transaktion
- Basis der Multiversioning-Logik

Nicht abschaltbar

- Sondern integraler Bestandteil der Datenbank

Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

Database Transactions: Start and End

Begin when the first DML SQL statement is executed.

End with one of the following events:

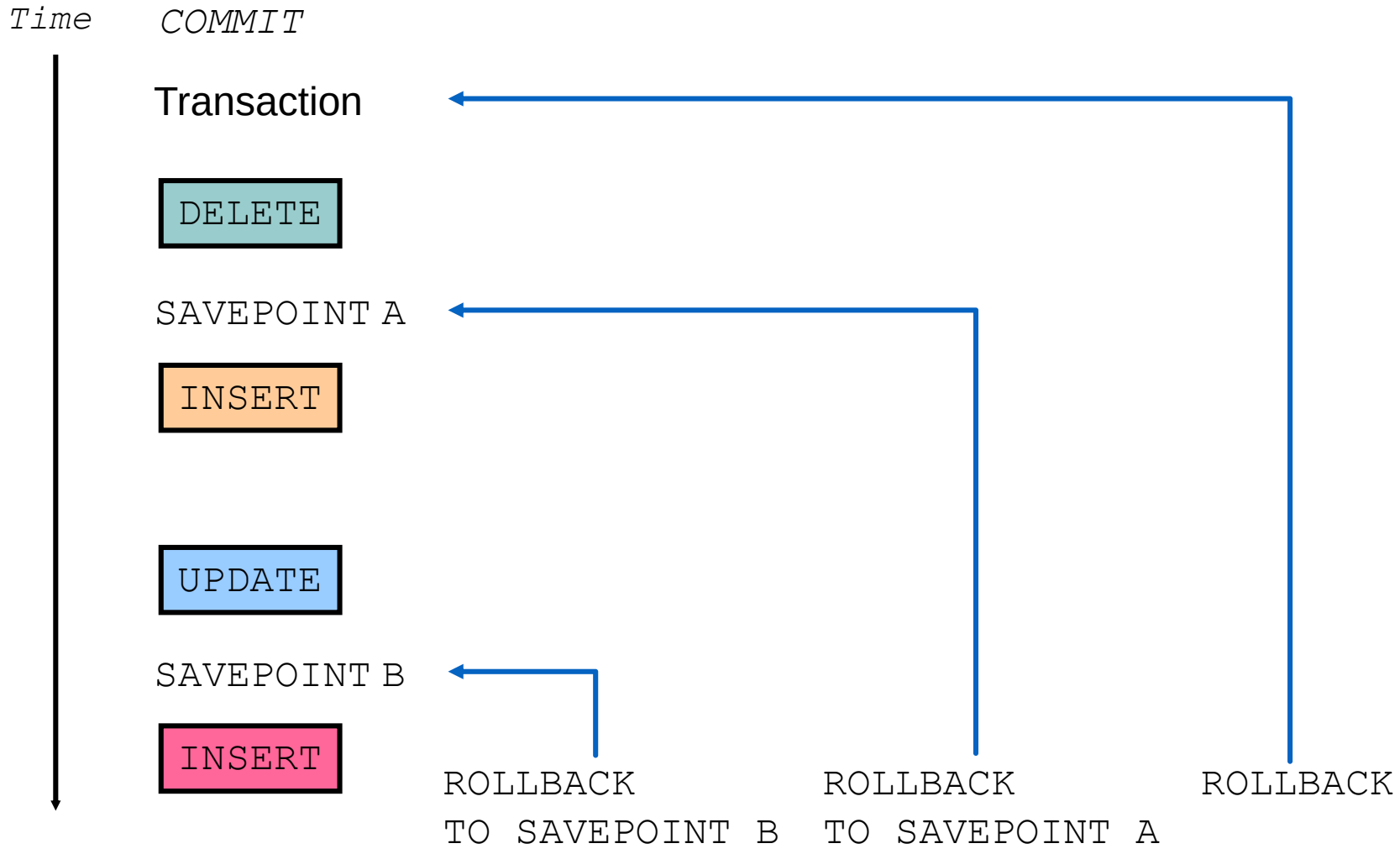
- A `COMMIT` or `ROLLBACK` statement is issued.
- A DDL or DCL statement executes (automatic commit).
- The user exits SQL Developer or SQL*Plus.
- The system crashes.

Advantages of `COMMIT` and `ROLLBACK` Statements

With `COMMIT` and `ROLLBACK` statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

Explicit Transaction Control Statements



Rolling Back Changes to a Marker

Create a marker in the current transaction by using the `SAVEPOINT` statement.

Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...
SAVEPOINT update_done;
SAVEPOINT update_done succeeded.
INSERT...
ROLLBACK TO update_done;
ROLLBACK TO succeeded.
```

Implicit Transaction Processing

An automatic commit occurs in the following circumstances:

- A DDL statement issued
- A DCL statement issued
- Normal exit from SQL Developer or SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements

An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure.

State of the Data Before `COMMIT` or `ROLLBACK`

The previous state of the data can be recovered.

The current user can review the results of the DML operations by using the `SELECT` statement.

Other users *cannot* view the results of the DML statements issued by the current user.

The affected rows are *locked*; other users cannot change the data in the affected rows.

State of the Data After `COMMIT`

Data changes are saved in the database.

The previous state of the data is overwritten.

All users can view the results.

Locks on the affected rows are released; those rows are available for other users to manipulate.

All savepoints are erased.

Committing Data

Make the changes:

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

Commit the changes:

```
COMMIT;
COMMIT succeeded.
```

State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

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

State of the Data After ROLLBACK: Example

```
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.
```

Statement-Level Rollback

If a single DML statement fails during execution, only that statement is rolled back.

The Oracle server implements an implicit savepoint.

All other changes are retained.

The user should terminate transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

FOR UPDATE Clause in a SELECT Statement

Locks the rows in the EMPLOYEES table where job_id is SA_REP.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

Lock is released only when you issue a ROLLBACK or a COMMIT.

If the SELECT statement attempts to lock a row that is locked by another user, the database waits until the row is available, and then returns the results of the SELECT statement.

FOR UPDATE Clause: Examples

You can use the FOR UPDATE clause in a SELECT statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

Rows from both the EMPLOYEES and DEPARTMENTS tables are locked.

Use FOR UPDATE OF *column_name* to qualify the column you intend to change, then only the rows from that specific table are locked.

Practice

Controlling transactions