Topic 10:

Transactions and Assertions

What is a Transaction?

The situation:

Individual SQL statements are often pieces of multi–step actions that a DBMS must manage.

Definition: Transaction

The ACID Properties of Transactions

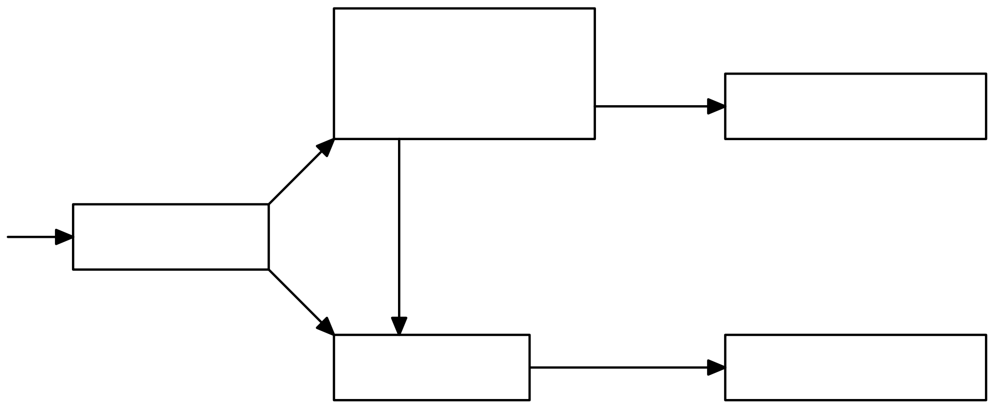
A is for __________________:

C is for __________________:

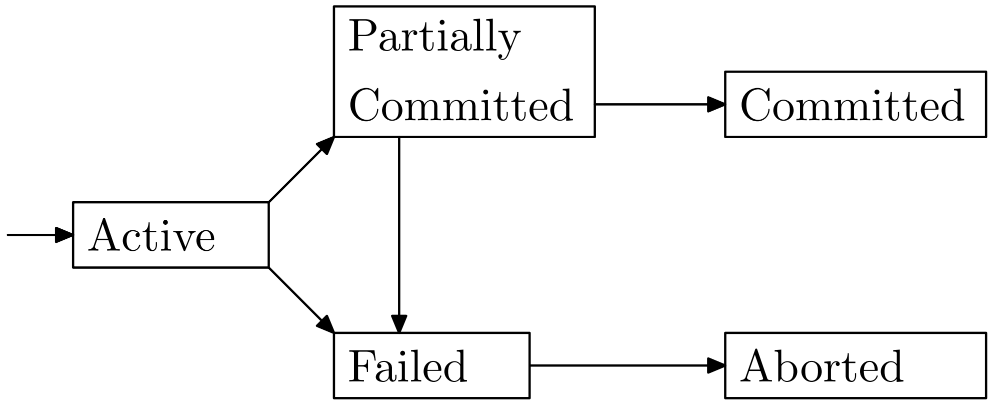
I is for __________________:

D is for __________________:

Transaction Lifetime (1 / 2)



Transaction Lifetime (2 / 2)



Transaction Isolation

Observation: In Oracle's PL/SQL, every action is automatically part of a transaction.

To stop a transaction (and start a new one), either:

To make each PL/SQL statement its own transaction:

Transaction Isolation Demo (1 / 2)

Each ‘user’ is an Oracle login in a separate terminal window:

	User 1	User 2
(1)	`@ xact.sql`	—
(2)	`show autocommit;` ⇒ “autocommit OFF”	—
(3)	`select * from score;`	—
(4)	—	`select * from score;` ⇒ no rows are selected
(5)	—	`select table_name from user_tables;` ⇒ yes, score exists!
(6)	`commit;`	—
(7)	—	`select * from score;` ⇒ score’s content is visible

(Continues . . .)

Transaction Isolation Demo (2 / 2)

	User 1	User 2
(8)	`insert into score values (5,460,'B');`	—
(9)	`select * from score;` ⇒ shows it	—
(10)	—	`select * from score;` ⇒ doesn’t show it
(11)	`rollback;`	—
(12)	`select * from score;` ⇒ like it was never there	—
(13)	`set autocommit on;`	—
(14)	`insert into score values (4,453,'B');`	—
(15)	—	`select * from score;` ⇒ shows it

Constraints in SQL

Consider:

```
create table applicant (
  id    integer,
  email char(30) not null,
  ...
  primary key (id)
);
```

Assertions (1 / 2)

The SQL standard provides for general assertions.

Example(s): No one in 460 can receive an 'E':

```
create assertion no_460_Es
check (not exists (select *
                     from score
                    where course = 460
                      and grade = 'E') );
```

Assertions (2 / 2)

. . . Oracle supports a form of general constraint within ‘create table’:

Example(s): No one can receive an ‘E’:

```
create table score (
  ...
  constraint no_fail check (grade <> 'E')
);
```

Trigger Basics (1 / 2)

- Triggers support the idea of ‘active databases’ (events initiate predetermined actions)
- Oracle does support these (stay tuned!)
- Triggers follow the “ECA” model:
  - ○
  - ○
  - ○
- Useful for input validation and update logging tasks

Trigger Basics (2 / 2)

Some Disadvantages of Triggers:

1. Hard to write the appropriate actions

2. Specified separately from relation(s)

3. Can reduce the DBMS' concurrency

4. Generally hard to anticipate how the triggers will interact

Triggers in Oracle (1 / 4)

Oracle's basic trigger definition syntax:

```
create trigger <name>
{before/after} {insert/delete/update of <attr>} on <relation>
[ [ for each row ] when ( < condition > ) ]
< PL/SQL block > ;
```

Component meanings:

- "**`for each row`**" gives row–level triggers (vs. statement–level):
  - "row–level": trigger executes when a row is changed
    - '**`before`**' – fires before a new value is written
    - '**`after`**' – fires after value is written; good for validation
  - "stmt–level": trigger executed when SQL statement is executed
- The PL/SQL block can be a compound statement
- Only use triggers when necessary – execution order not guaranteed!

Triggers in Oracle (2 / 4)

Oracle's Create Trigger command does only that — creates.

To activate the trigger, follow it with either:

(a)	`.`	← (period) terminates subprogram creation
	`run;`	← execute PL/SQL subprogram
(b)	`/`	← (slash) merges [.] and [run;]

Triggers in Oracle (3 / 4)

We want to know if someone tries to add a 460 'E' in score:

Example(s):

```
create trigger no_460_Es
after insert on score
for each row
when ( new.course = 460 and new.grade = 'E' )
 begin
  raise_application_error (-20000, 'message');
 end no_460_Es;
/
```

Triggers in Oracle (4 / 4)

Notes:

1. Could we use a trigger to change an inserted ‘E’ to a ‘D’?
   - No. We can’t change the table that triggered the rule currently being executed. Oracle will report a “mutating table” error.
2. It’s easy to create syntax errors when writing triggers
   - Use `sho err` to see the last compilation error
3. Removing a trigger is easy
   - Use `drop trigger <name>;`