Integrity Constraints &Transactions in SQL

Queries and Transactions

- Queries: requests to the DBMS to retrieve data from the database
- Updates: requests to the DMBS to insert, delete or modify existing data
- Transactions: logical grouping of query and update requests to perform a task
  - Logical unit of work (like a function/subroutine)

Execution Abstraction

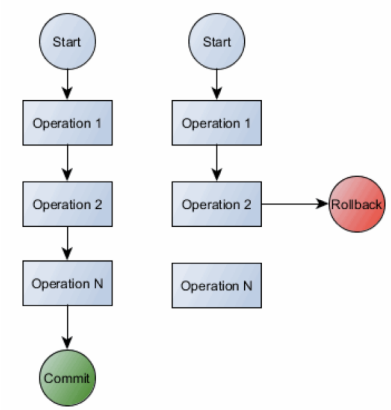
- *A **transaction** is a **logical unit of work** in DBMSs*
  - *It is the execution of a **program segment** that performs some function or task by accessing shared data (e.g., a db)*
  - logical grouping of query and update requests needed to perform a task
- Examples:
  - banking transaction
    - Deposit, withdraw, transfer $
  - (airline reservation
    - reserve a seat on a flight
  - inventory transaction
    - Receive, Ship, Update

Transaction's ACID Properties

- **A**tomicity (alias failure atomicity)
  Either all the operations associated with a transaction happen or none of them happens
- **C**onsistency Preservation
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- **I**solation (alias concurrency atomicity / serializability)
  Transactions are independent, the result of the execution of concurrent transactions is the same as if transactions were executed serially, one after the other
- **D**urability (alias persistence / permanence)
  The effects of completed transactions become permanent surviving any subsequent failures

SQL TRANSACTIONS

- Start:
  - SET TRANSACTION
- COMMIT ;
- ROLLBACK;



SQL TRANSACTIONS

- Basic transaction statements:
  - SET TRANSACTION READ WRITE NAME <name>;
    (SQL1: DECLARE TRANSACTION READ WRITE;)
  - SET TRANSACTION READ ONLY NAME <name>;
    (SQL1: DECLARE TRANSACTION READ ONLY;)
  - COMMIT ;
  - ROLLBACK;
- ROLLBACK default action

Standard SQL transactions vs PostgreSQL

- START TRANSACTION … should be unnecessary according to the SQL standard
  - Each SQL statement should implicitly start a transaction
  - Unless START TRANSACTION is issued, PostgreSQL implicitly issues a COMMIT after each SQL statement
    - This functionality is sometimes referred to as *autocommit*
  - In Oracle you need to issue COMMIT (to be safe)
- You cannot effectively have a multi-statement transaction without issuing a START TRANSACTION

Transaction Atomicity

- What do we expect with Atomicity?
  - "All or nothing"
- Consider a transaction:

```
set transaction read write name 'test';
insert into Student values (23, 'John', 'CS');
insert into Dept values ('CS', 501);
Commit;
```

- What happens if the first insert fails, e.g., due to a referential constraint violation?
  - Is the new tuple inserted into Department? No?
- If no error happens at commit time, the second insert is still committed!!!

Modes of Constraints Enforcement

- **NOT DEFERRABLE** or **IMMEDIATE**
  - Evaluation is performed at input time
  - By default constraints are created as NON DEFERRABLE
  - It *cannot* be changed during execution
- **DEFERRED**
  - Constraints are not evaluated until commit time
- **DEFERRABLE**
  - It can be changed within a transaction to be DEFERRED using SET CONSTRAINT
- Modes can be specified when a table is created.
  - INITIALLY IMMEDIATE: constraint validation to happen immediate
  - INITIALLY DEFERRED: constraint validation to defer until commit

Specifying Initial Eval. Mode inTables

```
CREATE TABLE SECTION
( SectNo sectno_dom,
  Name section_dom,
  HeadSSN ssn_dom,
  Budget budget_dom,
 CONSTRAINT section_PK
   PRIMARY KEY (SectNo) DEFERRABLE,
 CONSTRAINT section_FK
   FOREIGN KEY (HeadSSN) REFERENCES LIBRARIAN(SSN)
     INITIALLY DEFERRED DEFERRABLE,
 CONSTRAINT section_name_UN UNIQUE (Name)
     DEFERRABLE INITIALLY IMMEDIATE
);
```


Changing Constraint Evaluation Mode

- It is permitted only for deferrable constraints
- Setting the constraint validation mode within a transaction
  - set mode of all deferrable constraints

```
SET CONSTRAINT ALL IMMEDIATE;
SET CONSTRAINT ALL DEFERRED;
```

  - set mode of specific deferrable constraints (list)

```
SET CONSTRAINT section_budget_IC1 IMMEDIATE;
SET CONSTRAINT section_budget_IC1 DEFERRED;
```


Specifying Transaction Atomicity

- Errors at commit time: only when **deferred constraints** are violated
  - Constraints can be deferred if specified as **deferrable** in the table schema, and
  - deferred in the scope of the transaction
  - E.g., *assume the constraints are deferrable*

```
set transaction read write name 'test';
set constraints all deferred;
insert into Student values (23, 'John', 'CS');
insert into Dept values ('CS', 501);
Commit;
```

- No constraint violation of the first insert is detected at *commit time* → the whole transaction is committed

Specifying Transaction Atomicity (2)

- E.g. 2, *assume the constraints are deferrable and assume SID 23 exists in that Database*

```
set transaction read write name 'test';
set constraints all deferred;
insert into Student values (23, 'John', 'CS');
insert into Dept values ('CS', 501);
Commit;
```

- The constraint violation of the first insert is detected at *commit time* → the whole transaction is rollback

The chicken and the egg problem…

- `CREATE TABLE Chicken (ID INT PRIMARY KEY, eID INT REFERENCES Egg(ID));`

  `CREATE TABLE Egg(ID INT PRIMARY KEY, cID INT REFERENCES Chicken(ID));`
- We can't create these tables using these commands!
  - Do we know commands that could create these tables?
- But how can we insert into either table??
  - Need to treat two inserts into both tables as one logical unit of work...

Solving our problem

```
CREATE TABLE Chicken (ID INT PRIMARY KEY, eID INT);
CREATE TABLE Egg(ID INT PRIMARY KEY, cID INT);

ALTER TABLE Chicken ADD CONSTRAINT Chicken_FK
            FOREIGN KEY (eID) REFERENCES Egg(ID)
            DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE Egg ADD CONSTRAINT Egg_FK
            FOREIGN KEY (cID) REFERENCES Chicken(ID)
            DEFERRABLE INITIALLY IMMEDIATE;
```