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

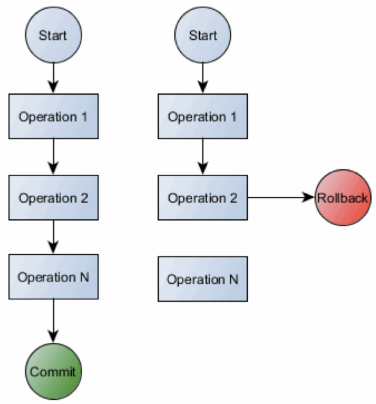
- *A **transaction** is a **logical unit of work** in DBMSs*
  - *It is the execution of a **program segment** that performs some function or task by accessing shared data (e.g., a db)*
  - logical grouping of query and update requests needed to perform a task
- Examples:
  - banking transaction
   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

Transaction's ACID Properties

- Atomicity (alias failure atomicity)
  Either all the operations associated with a transaction happen or none of them happens
- Consistency Preservation
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- Isolation (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
- Durability (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

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

Integrity Constraints in SQL

Structural Constraints

- Constraints (on Attributes):
  - NOT NULL
  - DEFAULT value
    - without the DEFAULT-clause, the default value is NULL
  - PRIMARY KEY (attribute-list)
  - UNIQUE (attribute list)
    - allows the specification of alternative key
  - FOREIGN KEY (key) REFERENCES table (key)

Referential Triggered Actions

- Actions if a Referential Integrity constraint is violated
  - SET NULL
  - CASCADE (propagate action)
  - SET DEFAULT
- Qualify actions by the triggering condition:
  - ON DELETE
  - ON UPDATE
- Note: Oracle does not support ON UPDATE & SET DEFAULT

Create Table with RI Trigger Actions

```
CREATE TABLE LIBRARIAN            /* or Micro_db.LIBRARIAN */
(   Name      name_dom,
    SSN       ssn_dom,
    Section   INTEGER,
    Address   address_dom,
    Gender    gender_dom,
    Birthday  DATE,
    Salary    DEC(8,2),
    CONSTRAINT librarian_PK  PRIMARY KEY (SSN) DEFERRABLE,
    CONSTRAINT librarian_FK
      FOREIGN KEY (Section)  REFERENCES SECTION (SNO)
      On Delete SET DEFAULT On Update CASCADE
      DEFERRABLE
);
```

Semantic Integrity Constraints

- A constraint is expressed as a Predicate, a condition similar to the one at the WHERE-clause of a query
- Three DDL constructs
  - Checks
  - Assertions
  - Triggers

Check Constraints

- CHECK *prohibits* an operation on a table that would violate the constraint. It is a *local* constraint.
- **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)
   DEFERRABLE,
```


Check Constraints…

```
CONSTRAINT section_budget_IC1
   CHECK ((Budget >= 0) AND (Budget IS NOT NULL))
   DEFERRABLE,
CONSTRAINT section_budget_IC2
   CHECK (NOT EXISTS
          (SELECT * FROM SECTION WHERE budget <
          (SELECT SUM (Salary) FROM LIBRARIAN)))
   DEFERRABLE,
CONSTRAINT Head_Lib_IC3
   CHECK (HeadSSN <> ALL (SELECT SSN FROM Retiree))
   DEFERRABLE
);
```


Assertions

- Similar to CHECK but they are **global** constraints

  CREATE OR REPLACE ASSERTION <assertion_name>

  CHECK <Predicate> [Mode of Evaluation];

  - **Predicate** usually involves EXISTS and NOT EXISTS
- E.g., **CREATE OR REPLACE ASSERTION** budget_constraint

```
CHECK (NOT EXISTS
   (SELECT * FROM SECTION WHERE budget <
    ( SELECT SUM (Salary) FROM LIBRARIAN)));
```

VQuery that violates IC

- Dropping an assertion…

  DROP ASSERTION budget_constraint;

Triggers

- A trigger consists of 3 parts:
  1. Event(s),
  2. Condition, and
  3. Action

- E.g., Notify the Dean whenever the number of students in any major exceeds 1800

Triggers vs. Assertions

- Assertion
  - Condition must be true for each database state
  - DBMS rejects operations that violate such condition

- Trigger
  - DBMS takes a certain **action** when condition is true
  - Action could be: stored procedure, SQL statements, Rollback, etc.

My First Trigger

- Notify the Dean when the # of students in any major exceeds 1800

CREATE TRIGGER Major_Limit

Event(s)
Condition
Action

Example

- Notify the Dean when the # of students in any major exceeds 1800

CREATE TRIGGER Major_Limit

Event(s)

```
WHEN( EXISTS (
        SELECT Major_Code, COUNT(*)
        FROM Student
        GROUP BY Major_Code
        HAVING COUNT(*) > 1800 ))
```

Action

Example

- Notify the Dean when the # of students in any major exceeds 1800

CREATE TRIGGER Major_Limit

Event(s)

```
WHEN( EXISTS (
        SELECT Major_Code, COUNT(*)
        FROM Student
        GROUP BY Major_Code
        HAVING COUNT(*) > 1800 ))
```

```
CALL email_dean(Major_code);
```


Example

- Notify the Dean when the # of students in any major exceeds 1800

```
CREATE TRIGGER Major_Limit
AFTER INSERT OR UPDATE OF Major_Code
ON Student
WHEN( EXISTS (
        SELECT Major_Code, COUNT(*)
        FROM Student
        GROUP BY Major_Code
        HAVING COUNT(*) > 1800 ))
CALL email_dean(Major_code);
```


Example: Assertions Vs. Triggers

- **CREATE OR REPLACE ASSERTION** budget_constraint
  CHECK (**NOT EXISTS**
  (SELECT * FROM SECTION WHERE budget <
  (SELECT SUM (Salary) FROM LIBRARIAN)));

- **CREATE OR REPLACE TRIGGER** budget_constraint_trigger
  after INSERT, UPDATE of Salary
  ON LIBRARIAN
  WHEN (**EXISTS** (SELECT * FROM SECTION WHERE budget <
  (SELECT SUM (Salary) FROM LIBRARIAN))
  ROLLBACK;

Triggers (SQL99)

```
CREATE or REPLACE TRIGGER <trigger-name>
    <time events> ON <list-of-tables>
    REFERENCING { NEW | OLD} AS <user-name>
  [FOR EACH { ROW | STATEMENT} ]
    [WHEN (<Predicate>)]
  <action>
```

- time: **before** or **after**
- events: **Insert, Delete, Update [of <list of attributes>]**
- **NEW & OLD** refer to new & old (existing) tuples/table respectively
- The REFERENCING clause assigns aliases to NEW and OLD
- action: Stored procedure or
  BEGIN ATOMIC {<SQL procedural statements>} END

Oracle Example: Statement Trigger

- Statement-level trigger fires once by the triggering statement & defined on a single table
- No WHEN-clause in the definition of statement trigger

```
CREATE OR REPLACE TRIGGER Audit_Updater
  AFTER
  INSERT OR DELETE OR UPDATE
  ON STUDENTS
 BEGIN
   INSERT INTO AUDIT_Table VALUES (`STUDENT', sysdate);
 END;
/
```

- The end slash ("/") installs and activates the trigger

Oracle Example: Row-Level Trigger

- Row- or tuple-level trigger fires once for each row affected by the triggering statement
- In Oracle, triggers are defined on a single table.

```
CREATE OR REPLACE TRIGGER trigger_deans_list
  AFTER INSERT ON STUDENTS
  REFERENCING NEW AS newRow
 FOR EACH ROW
   WHEN (newRow.QPA > 3.5)
 BEGIN
   INSERT INTO DL VALUES ( :newRow.SID, :newRow.QPA );
 END;
/
```

- Scope Rules: In the trigger body, NEW and OLD are global "variables" and must be preceded by a colon (":"), but in the WHEN clause (triggering condition), they do not have a preceding colon!

Mutating Trigger

- Recursive call of triggers is not permitted
- Table read in a trigger it cannot be updated

```
CREATE OR REPLACE TRIGGER my_bad_auto_sid
   AFTER INSERT ON STUDENTS
 FOR EACH ROW
 BEGIN
    SELECT MAX(SID) +1 INTO : NEW.SID
    FROM Students;
 END;
/
```

- ERROR at line 1:
  ORA-04084: cannot change NEW values for this trigger type

Mutating Trigger

- **Before** does not acquire locks

```
CREATE OR REPLACE TRIGGER my_auto_sid
  BEFORE INSERT ON STUDENTS
 FOR EACH ROW
 BEGIN
    SELECT MAX(SID) +1 INTO : NEW.SID
    FROM Students;
 END;
/
```

- Trigger created.
- INTO: the tuple assignment operator in PL/SQL

Auto-increment in Oracle

- It is achieved with a trigger
- Two special tables **dual** & **sequence**

```
CREATE SEQUENCE oracle_example
    NOCYCLE MAXVALUE 99999 START With 1;
```

- dual is a table created by Oracle

```
SQL> describe dual;
   Name                    Null?   Type
   ----------------------- -------- ----------------
   DUMMY                           VARCHAR2(1)
SQL> select * from dual;      DUMMY
                              ----------
                               X
```


Auto-increment in Oracle …

- Trigger on the table with auto-increment

```
CREATE OR REPLACE TRIGGER
 auto_increment_example
    BEFORE INSERT ON STUDENTS
  FOR EACH ROW
  DECLARE next_id integer
  BEGIN
    SELECT oracle_example.NEXTVAL INTO next_id
     FROM dual;
    :new.SID := next_id;
  END;
 /
```


Enable & Disable Triggers

- Enable/Disable All Triggers:

  ALTER TABLE <table_name> ENABLE ALL TRIGGERS;

  ALTER TABLE <table_name> DISABLE ALL TRIGGERS;

  - E.g., ALTER TABLE Librarian DISABLE ALL TRIGGERS;
- Enable/Disable Individual Trigger

  ALTER TRIGGER <trigger_name> ENABLE | DISABLE;

  - E.g., ALTER TRIGGER librarian_salary_trigger DISABLE;
- Drop A Trigger

  DROP TRIGGER <trigger-name>;

  - E.g., DROP TRIGGER librarian_salary_trigger;

Dropping a Trigger & Final note on IC

- Assertions and Checks Vs. Triggers
  - Assertions and Checks support the declarative approach of supporting Integrity Constraints
  - Triggers combine the declarative and procedural approach of implementing integrity constraints