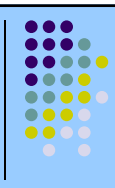
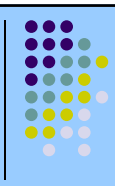
Constraints

Keys: Primary keys and unique



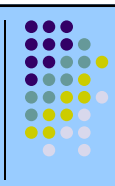
```
CREATE TABLE Student (
  sNum int, sName varchar (20), dept char (2),
  CONSTRAINT key PRIMARY KEY (sNum),
  CONSTRAINT uniqueName UNIQUE (sName));
```


Unique vs. primary keys



- Attribute values may be null even if they are declared unique (primary key attributes should not be null).
- We can have any number of unique constraints for a table (only one primary key constraint can be defined for a table).

Foreign Keys: Referential Integrity Constraints



- Specified for a table as

```
[CONSTRAINT <fkName>] FOREIGN KEY (<a1List>)
REFERENCES <tableName> (<a2List>)
```

- eg: for example, for table R, we can specify

```
FOREIGN KEY (x, y) REFERENCES S (a, b)
```

- Requires (a, b) be unique or primary key of S.
- Consider a row in R with values of x as a1, and y as b1 where a1, b1 are both non-null. There must be a row in S with values for (a, b) as (a1, b1).

Maintaining referential integrity: Inserts/Deletes/Updates

- Default: reject any modification that violates the constraints.
- We can specify other policies for delete/update as set null/cascade.
- Eg: for Student relation

```
FOREIGN KEY (prof) references Professor
(pNum) ON DELETE SET NULL ON UPDATE
CASCADE
```


ON DELETE

- Let us consider the foreign key on table R referencing table S such as
  `FOREIGN KEY (x, y) REFERENCES S (a, b)`
- SET NULL
  - If a delete is performed on S, any rows in R that reference that row in S have their (x, y) attributes set to null
- CASCADE
  - If a delete is performed on S, any rows in R that reference that row in S are also deleted.

ON UPDATE

- SET NULL
  - If an update is performed on S, any rows in R that reference that row in S have their (x, y) attributes set to null
- CASCADE
  - If a delete is performed on S, any rows in R that reference that row in S are also updated.
- ON UPDATE constraints are not supported by Oracle

Example

```
CREATE TABLE Student (
  sNum int, sName varchar (20), prof int,
  CONSTRAINT pk PRIMARY KEY (snum),
  CONSTRAINT uk1 UNIQUE (sname),
  CONSTRAINT FOREIGN KEY (prof) REFERENCES
  Professor (pNum) ON DELETE SET NULL);
```

Column Check constraints

- Constraints specified on a column
- We can specify attributes as NULL or NOT NULL.

  eg: `sName varchar (20) NOT NULL`
- We can specify CHECK constraints.

  eg: `gender char (1) CHECK (gender IN ('F', 'M'))`

```
salary int CONSTRAINT minSalary CHECK (salary >=
  60000)
CONSTRAINT minSalary check (salary >= 60000)
```


Other tips

- While dropping a table such as S, where S is referenced by a FK from R, we can specify as

```
ALTER TABLE S DROP COLUMN a CASCADE
  CONSTRAINTS;

DROP TABLE S CASCADE CONSTRAINTS;
```

Altering Constraints

- Constraints can be added to an existing table.

```
ALTER TABLE ADD CONSTRAINT [<cName>]
<cBody>
```

- Any constraint that has a name can be dropped

```
ALTER TABLE DROP CONSTRAINT <cName>
```


Constraints on the entire relational schema

Assertions:

```
CREATE ASSERTION <assertionName> CHECK
  (<condition>)
```

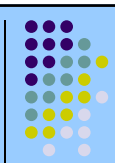
eg:

```
CREATE ASSERTION CHECK (
     NOT EXISTS (SELECT *
          FROM PROFESSOR
          WHERE salary < 60000));
```

Condition is any condition that can appear in WHERE clause. For any database modification, the assertion must be true.

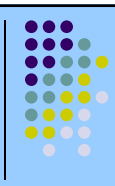
Assertions not supported by Oracle – could be very inefficient.

Triggers (Event, Condition, Action rules)



- We specify triggers as Event, Condition, Action rules; condition is optional.
- When event occurs, and condition is satisfied, the action is performed.

Triggers – Events, Action



- Events could be

  ```
  BEFORE | AFTER   INSERT  | UPDATE | DELETE
    ON <tableName>
  ```

  eg: `BEFORE INSERT ON Professor`
- Action is specified as a body of PSM

Example Trigger

Assume our DB has a relation schema
Professor (pNum, pName, salary)

We want to write a trigger that ensures that any new professor inserted has salary >= 60000

Example trigger

```
CREATE OR REPLACE TRIGGER minSalary BEFORE INSERT ON
  Professor FOR EACH ROW
DECLARE temp int;       -- dummy variable not needed
BEGIN
  IF (:new.salary < 60000)
      THEN RAISE_APPLICATION_ERROR (-20004,
            'Violation of Minimum Professor Salary');
  END IF;
temp := 10;       -- to illustrate declared variables
END;
.
run;
```

Things to note

- FOR EACH ROW – specifies that for the trigger is performed for each row inserted
- :new refers to the new tuple inserted
- This trigger is checked before the tuple is inserted; if (:new.salary < 60000) then an application error is raised and hence the row is not inserted; otherwise the row is inserted.
- RAISE_APPLICATION_ERROR is built-in Oracle function.
- Use error code: -20004; this is in valid range
- Your trigger ends with a "." and a "run;"

Murali Mani

Displaying Trigger Definition Errors

- When you define the trigger, you might get

```
Warning: Trigger created with compilation errors.
```

- This means that there is/are errors in your trigger.
- To view the errors,

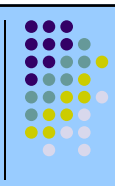
```
show errors trigger <trigger_name>;
```

- To drop a trigger

```
drop trigger <trigger_name>;
```

Murali Mani

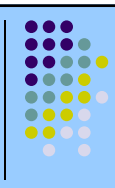
Example trigger using Condition



```
CREATE OR REPLACE TRIGGER minSalary BEFORE INSERT ON
  Professor FOR EACH ROW WHEN (new.salary < 60000)
BEGIN
  RAISE_APPLICATION_ERROR (-20004,        'Violation
  of Minimum Professor Salary');
END;
.
run;
```

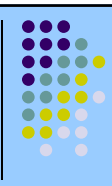
- Conditions **cannot** be arbitrary conditions; they can use new rows etc.

Triggers: REFERENCING



```
CREATE OR REPLACE TRIGGER minSalary BEFORE
  INSERT ON Professor REFERENCING NEW as
  newTuple FOR EACH ROW WHEN
  (newTuple.salary < 60000)
BEGIN
  RAISE_APPLICATION_ERROR (-20004,
     'Violation of Minimum Professor
  Salary');
END;
.
run;
```

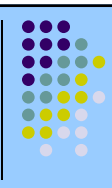

Example Trigger



- Ensure that salary does not decrease

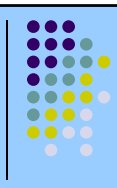
```
CREATE OR REPLACE TRIGGER minSalary BEFORE
  UPDATE ON Professor REFERENCING OLD AS
  oldTuple NEW as newTuple FOR EACH ROW WHEN
  (newTuple.salary < oldTuple.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20004,
     'Salary Decreasing !!');
END;
.
run;
```


Row level trigger vs Statement level trigger



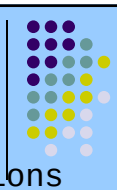
- Row level triggers can access the new data, statement level triggers cannot
- Statement level triggers will be more efficient if we do not need to check every row.
- eg: Consider a relation schema Account (num, amount) where we will allow creation of new accounts only during normal business hours.

Example: Statement level trigger



```
CREATE OR REPLACE TRIGGER MYTRIG1
BEFORE INSERT ON Account
BEGIN
   IF (TO_CHAR(SYSDATE,'dy') IN
  ('sat','sun')) OR
  (TO_CHAR(SYSDATE,'hh24:mi') NOT BETWEEN
  '08:00' AND '17:00') THEN
      RAISE_APPLICATION_ERROR(-20500,'Cannot
  create new account now !!');
   END IF;
END;
.
run;
```


Combining multiple events into 1 trigger



```
CREATE OR REPLACE TRIGGER salaryRestrictions
AFTER INSERT OR UPDATE ON Professor
FOR EACH ROW
BEGIN
IF (INSERTING AND :new.salary < 60000) THEN
  RAISE_APPLICATION_ERROR (-20004, 'below
  min salary'); END IF;
IF (UPDATING AND :new.salary < :old.salary)
  THEN RAISE_APPLICATION_ERROR (-20004,
  'Salary Decreasing !!'); END IF;
END;
.
run;
```


Triggers

```
CREATE [OR REPLACE] TRIGGER <triggerName>
BEFORE | AFTER  INSERT|DELETE|UPDATE
  [OF <columnList>] ON
  <tableName>|<viewName>
  [REFERENCING [OLD AS <oldName>] [NEW AS
  <newName>]]
[FOR EACH ROW]
[WHEN (<condition>)]
<PSM body>;
```


Trigger Tips !!

- Check the tables
  - `user_triggers`
  - `user_trigger_cols`
- **`ORA-04091`**: mutating relation problem
  - In a **row level trigger**, you **cannot** have the body refer to the table specified in the event
- Also `INSTEAD OF` triggers can be specified for view updates.