Constraints

Data Integrity:

Maintaining and ensuring the accuracy and consistency of the data.

Ex:

- Students registered in a class are also in the Students relation. (Foreign key)
- Each student has a different id (primary key)
- A student cannot register to more than 5 courses.

etc. etc...

Integrity Constraints are a feature of DBMS to help guarantee integrity of the database.

Ex:

Primary keys
Attribute IS NOT NULL
Attribute IS UNIQUE
Foreign Key

} all types of constraints

Type of constraints:

- Attribute. Specified along declaration of attribute.
- Tuple: Specified along table. Applies to entire tuple

Table constraints checked everytime a tuple is inserted, updated or deleted.

- Database: Apply to entire DB. Checked everytime the DB (any of its tuples is inserted, updated, deleted.

Primary key

- As attribute constraint (key is only one attribute

```
CREATE TABLE <name> (
    :
    attname type PRIMARY KEY,
```

- As tuple constraint (multi attr. PK)

```
CREATE TABLE ....
    :
    declaration of attributes
    :
    PRIMARY KEY (list of attr).
    :
```

UNIQUE

For other candidate keys you can use UNIQUE.

For one-attribute candidate keys:

```
    :
attname type UNIQUE
    :
```

Or more generally as tuple constraint:

```
    :
UNIQUE (<att-list>)
```

↑ it can be one or more attributes.

UNIQUE is implied with Primary key constraints.

NOT NULL

Only makes sense as an attribute constraint.

```
    :
<attname> <type> NOT NULL;
    :
```

Implicit for PKs, but UNIQUE att can be NULL

Referential Integrity: Foreign key Constraint.

As attribute constraint.

```
   :
   attname type REFERENCES
         <relation>
```

As tuple constraint.

```
   :
   FOREIGN KEY (<attlist>) REFERENCES
         <relation>
   :
```

↑ can be one or more attr.

Makes a FK constraint for attribute attname to the primary key of <relation>

By default the reference is to the primary key of the other table. But we can use other attributes:

```
... REFERENCES <relation> (<attlist>)
```

But <attlist> must be declared UNIQUE

Assume

```
CREATE TABLE R (
      a int PRIMARY KEY
);
CREATE TABLE S (
    a int PRIMARY KEY,
    FOREIGN KEY (a) REFERENCES R
);
```

What if a tuple in R, referenced in S is deleted:

R

a
3
2
5

S

a
2
5

What if we delete $\sigma_{a=5} R$?

What if we change in R a = 5 to a = 6 ?!?

4 options:

1) CASCADE Delete tuple in S too or update value in S to match new value in tuple of R

2) RESTRICT Deny if there are tuples that reference tuple being deleted. Default!

3) SET NULL Set the attribute(s) in the tuple that references to NULL and allow the delete or update of the tuple to proceed.

4) SET DEFAULT Replaces values of tuple in S with default values

Syntax

...

FOREIGN KEY (...) REFERENCES

...

$$\text{ON} \left\{ \begin{matrix} \text{DELETE} \\ \text{UPDATE} \end{matrix} \right\} \left\{ \begin{matrix} \text{CASCADE} \\ \text{RESTRICT} \\ \text{SET NULL} \\ \text{SET DEFAULT} \end{matrix} \right\}$$

Default

In insertions, attributes are set to NULL if not specified

Ex.

R(a, b, c)

INSERT INTO R(b) VALUES (5);

Rejected, the Primary key (a) cannot be NULL.

INSERT INTO R(a) VALUES (3)

inserts: (3, NULL, NULL) into R

We can change this behaviour:

⋮

<attname> <type> DEFAULT <value>

If not explicitly given, attribute is set to default value.

CHECK

Every time tuple is updated or tuple inserted a predicate is evaluated.

Operation fails unless predicate is true:

Ex:

attribute CHECK

```
  :
year int CHECK (year > 1900),
  :
gender char(1) CHECK
                (gender IN ('F', 'M')),
  :
CHECK (a+b = 5)
```

assuming both att. are declared.

tuple CHECK

It can contain a subquery (as any predicate in a selection:

```
custumer id   CHAR(10),
creditlimit   REAL,
CHECK  (creditLimit <=
              SELECT sum(orders.amount)
              FROM   orders
              WHERE  orders.custid =
                        custumer id).
```

Note how we use the attribute of the tuple being operated upon in the subquery.

⋮

```
(creditLimit <=
 SELECT sum(orders.amount)
  FROM orders
  WHERE orders.custid =
           custumer id).
```

value of current tuple

This is a good use of correlated subqueries.

(In general avoid them because they tend to have <u>horrible</u> performance)

Altering Constraints

Every constraint gets a name.
We can give explicit names:

```
CONSTRAINT <name> <constraint>
```

Ex:

```
CONSTRAIN tablePK PRIMARY KEY (a)
```

Name becomes global!!

We can refer to it:

```
ALTER TABLE <tableName>
DROP CONSTRAINT <constraintName>
```

We can add constraints to an already created table:

```
ALTER TABLE ADD CONSTRAINT
<name> <constraint>;
```

Ex:

```
ALTER TABLE R ADD CONSTRAINT
myConst UNIQUE (a,b);
```

Assertions

An assertion is a predicate that must always be TRUE

```
CREATE ASSERTION <assertion Name>
   CHECK ( <condition>)
```

Ex: No movie can have more than 100 roles.

```
CREATE ASSERTION NoMovieMore100roles
  CHECK (NOT EXISTS
              (SELECT id FROM
                 Productions NATURAL
                   NATURAL JOIN Roles
                 WHERE att IS NULL
                 GROUP BY id
                 HAVING count(*) > 100);
```

At least one student pases each corse:

```
CREATE ASSERTION AtLeastOne CHECK
   ( 1 <= ALL
           (SELECT count(*) FROM
               Enrolled WHERE grade>=50
               GROUP BY cid)
```

True if $1 \leq$ every value in set.
true if set is empty.

Warning :

Assertions can slow a DB.
⇒ They are executed every time the underlying relations change.

Postgres does not support them!!

Triggers

Triggers are events that are executed in response to an action.

1) An event "triggers" a trigger

2) Once started, a trigger checks a condition (optional). If not true trigger stops.

3) An action is executed

A trigger can be used to replace an event (INSTEAD OF)

Check can be done with the state of the database before or after action.

Trigger can be executed

- per each affected tuple, or
- one for all affected tuples.

A trigger can be used to

- Alter the behavior of an operation
- Do extra work.
- To abort the operation
- To update views (INSTEAD OF)

Syntax:

Postgres, Simplified

```
CREATE TRIGGER <name>
  { BEFORE     }  { INSERT                 }
  { AFTER      }  { DELETE                 }
  { INSTEAD OF }  { UPDATE                 }
                  { UPDATE OF attname, ... }
FOR EACH { ROW       }
         { STATEMENT }
WHEN (condition)
EXECUTE PROCEDURE <name>(args)
```

Ex:

```
 CREATE TRIGGER check_update
   BEFORE UPDATE ON Accounts
   FOR EACH ROW
   WHEN (OLD.balance IS DISTINCT FROM
                  NEW.balance)
   EXECUTE PROCEDURE check_account_update;

CREATE TRIGGER log_update
  AFTER UPDATE
  FOR EACH ROW
  WHEN (OLD.* IS DISTINCT FROM NEW.*)
  EXECUTE PROCEDURE log_acct_update;
```