Constraints (2)

Check Constraints

[CONSTRAINT <name>] CHECK(<condition>)

allows users to restrict possible attribute values for columns to admissible ones

Example:

- The name of an employee must consist of upper case letters only;
- The minimum salary of an employee is 500;
- Department numbers must range between 10 and 100:

These three are **column constraints**, and can only refer to the corresponding column.

```
CREATE TABLE Emp (
   empno NUMBER,
   ename VARCHAR2(30) CONSTRAINT check_name
                          CHECK( ename = UPPER(ename) ),
   sal NUMBER CONSTRAINT check_sal CHECK( sal >= 500 ),
   deptno NUMBER CONSTRAINT check_deptno
                          CHECK(deptno BETWEEN 10 AND 100)
);
```

Checking

- DBMS automatically checks the specified conditions each time a database modification is performed on this relation.
  - E.g., the insertion

```
INSERT INTO emp VALUES(7999,'SCOTT',450,10);
```

causes a constraint violation

```
ORA02290: check constraint (SAL_CHECK) violated
```

and is rejected.

Check Constraints (cont'd)

- A check constraint can also be a **table constraint**, and the <condition> can refer to any column of the table.
- **Example**:
  - project's start date must be before project's end date

```
CREATE TABLE Project  (
    ... ,
    pstart DATE,
    pend DATE,
    ... ,
    CONSTRAINT dates_ok CHECK (pend > pstart)
);
```

Table constraint

Violating Tuples

```
--Adding a violating constraint

ALTER TABLE Emp DROP CONSTRAINT check_sal;

INSERT INTO Emp(empno, ename, sal, deptno)
           VALUES(9, 'ALEX', 300, 20);

ALTER TABLE Emp ADD CONSTRAINT check_sal CHECK(sal >= 500)
   EXCEPTIONS INTO Exceptions;

--The constraint cannot be created at all, because there is
--a violating tuple.
```

- To identify those tuples that violate a constraint whose activation failed, one can use the clause

  `EXCEPTIONS INTO Exceptions`.

  `Exceptions` is a table that we should create and stores information about the violating tuples.

Violating Tuples (cont.)

- First we have to create the **`Exceptions`** table:

```
CREATE TABLE Exceptions(
    row_id ROWID,
    owner VARCHAR2(30),
    table_name VARCHAR2(30),
    constraint VARCHAR2(30)
);
```

Every tuple has a (pseudo) column of type rowid that is used to identify tuples.

row_id here will reference to rowid in the Emp table.

Besides the row_id, the name of the table, the table owner, as well as the name of the violated constraint are stored.

- Then, we can query it:

```
SELECT Emp.*, constraint
FROM Emp, Exceptions
WHERE Emp.rowid = Exceptions.row_id;
```

Writing Constraints Correctly

- Create table MovieStar. If the star gender is 'M', then his name must not begin with 'Ms.'.

```
CREATE TABLE MovieStar (
   name CHAR(20) PRIMARY KEY,
   address VARCHAR(255),
   gender CHAR(1),
   CHECK (gender<>'M' OR name NOT LIKE 'Ms.%')
);
```

We can't use an "implication." We should formulate it in terms of OR.
p->q is the same as (not p) OR q.

Exercise – mutually exclusive subclasses

```
CREATE TABLE Vehicles (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type IN ('SUV', 'ATV')),
  fuel_type CHAR(4),
  door_count INT CHECK(door_count >= 0),
  UNIQUE(vin, vehicle_type)
);
CREATE TABLE SUVs (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type ='SUV'),
  FOREIGN KEY (vin, vehicle_type) REFERENCES Vehicles (vin, vehicle_type)
     ON DELETE CASCADE
);
CREATE TABLE ATVs (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type ='ATV'),
  FOREIGN KEY (vin, vehicle_type) REFERENCES Vehicles (vin, vehicle_type)
     ON DELETE CASCADE
);
```

Views with check option

- Note that in most DBMS'es (including ORACLE) only simple conditions are allowed. For example
  - It is not allowed to refer to columns of other tables
  - No queries as check conditions.

- Solution: Use views WITH CHECK OPTION

Another Example

```
CREATE TABLE Hotel (
  room_nbr INT NOT NULL,
  arrival_date DATE NOT NULL,
  departure_date DATE NOT NULL,
  guest_name CHAR(15) NOT NULL,
  PRIMARY KEY (room_nbr, arrival_date),
  CHECK (departure_date > arrival_date)
);
```

We want to add the constraint that reservations do not overlap.

Exercise – Hotel Stays

We want to add the constraint that reservations do not overlap.

```
CREATE VIEW HotelStays AS
SELECT room_nbr, arrival_date, departure_date, guest_name
FROM Hotel H1
WHERE NOT EXISTS (
  SELECT *
  FROM Hotel H2
  WHERE H1.room_nbr = H2.room_nbr AND
        (H2.arrival_date < H1.arrival_date AND H1.arrival_date < H2.departure_date)
)
WITH CHECK OPTION;
```

Exercise – Hotel Stays – Inserting

```
INSERT INTO HotelStays (room_nbr, arrival_date,
    departure_date, guest_name)
VALUES(1, '2016-01-01', '2016-01-03', 'Alex');
```

This goes Ok.

```
INSERT INTO HotelStays (room_nbr, arrival_date,
    departure_date, guest_name)
VALUES(1, '2016-01-02', '2016-01-04', 'Ben');
 *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause
    violation
```