

# Constraints

Primary Key, Unique, Foreign Key, Not Null

# Primary Keys

```
CREATE TABLE Movies (  
    title CHAR(40) PRIMARY KEY,  
    year INT,  
    length INT,  
    type CHAR(2)  
);
```

```
CREATE TABLE Movies (  
    title CHAR(40),  
    year INT,  
    length INT,  
    type CHAR(2),  
    PRIMARY KEY (title, year)  
);
```

# NOT NULL

```
CREATE TABLE ABC (  
  A int NOT NULL,  
  B int NULL,  
  C int  
);
```

```
insert into ABC values ( 1, null, null);  
insert into ABC values ( 2,  3,  4);  
insert into ABC values (null,  5,  6);
```

The first two records can be inserted, the **third cannot**, throwing an error:

*ERROR: null value in column "a" violates not-null constraint Detail: Failing row contains (null, 5, 6).*

# UNIQUE

- UNIQUE constraint doesn't allow duplicate values in a column.
  - If it encompasses more columns, no two equal combinations are allowed.
  - However, nulls can be inserted multiple times
    - this is the only difference from PRIMARY KEY

```
CREATE TABLE AB (  
  A int UNIQUE,  
  B int  
);
```

```
insert into AB values (2, 1);  
insert into AB values (null,9);  
insert into AB values (null,9);  
insert into AB values (2, 7);
```

- The last statement throws an error.

*ERROR: duplicate key value violates unique constraint "ab\_a\_key" Detail: Key (a)=(2) already exists.*

# Constraint names

- Every constraint, has a name. For last table, the name is: ***ab\_a\_key***
- We can explicitly name a constraint for easier handling.

```
CREATE TABLE ABC (  
  A int,  
  B int,  
  C int,  
  CONSTRAINT my_unique_constraint UNIQUE (A,B)  
);
```

# Dropping/Adding Constraints

## Examples

```
ALTER TABLE AB DROP  
    CONSTRAINT ab_a_key;
```

```
ALTER TABLE AB ADD  
    CONSTRAINT unique_a UNIQUE (A);
```

# Listing constraints

```
SELECT conname  
FROM pg_constraint  
WHERE conrelid = 'movies'::regclass
```



Converts table  
name to table oid

**Result from movies table:**

```
movies_pkey  
movies_studio_name_fkey
```

# Foreign key constraints

**Example:** Each employee in table **Emp** must work in a department that is contained in table **Dept**.

```
CREATE TABLE Emp (  
    empno INT PRIMARY KEY,  
    ... ,  
    deptno INT REFERENCES Dept(deptno)  
);
```

**Dept** table has to exist first!



# Longer syntax for foreign keys

**Remark.** If you don't specify **primary keys** or **unique** constraints in the parent tables, you can't specify **foreign keys** in the child tables.

```
CREATE TABLE MovieStars(  
    name VARCHAR2(20) PRIMARY KEY,  
    address VARCHAR2(30),  
    gender VARCHAR2(1),  
    birthdate VARCHAR2(20)  
);
```

```
CREATE TABLE Movies (  
    title VARCHAR2(40),  
    year INT,  
    length INT,  
    type VARCHAR2(2),  
    PRIMARY KEY (title, year)  
);
```

```
CREATE TABLE StarsIn (  
    title VARCHAR2(40),  
    year INT,  
    starName VARCHAR2(20),  
    CONSTRAINT fk_movies FOREIGN KEY(title,year) REFERENCES Movies(title,year),  
    CONSTRAINT fk_moviestars FOREIGN KEY(starName) REFERENCES MovieStars(name)  
);
```

Naming Constraints  
is Optional

# Satisfying a Foreign key constraint

Each row in the child table must satisfy one of the following two conditions:

- Foreign key value must either
  1. appear as a primary key value in the parent table, **or**
  2. be **null**
    - If composite foreign key, at least one attribute must be **null**
- If we don't want NULL's in a foreign key **we must say so**.
  - **Example:** There should always be a project manager, who **must** be an employee.

```
CREATE TABLE Project (  
    pno INT PRIMARY KEY,  
    pmno INT NOT NULL REFERENCES Emp,  
    ...  
);
```

When only the name of the parenttable is given, the primary key of that table is assumed.

# Foreign key constraints (cont.)

- A foreign key constraint may also refer to the same table, i.e., parent table and child table are the same.
- **Example:** Every employee must have a manager who must be an employee.

```
CREATE TABLE Emp (  
    empno INT PRIMARY KEY,  
    ...  
    mgrno INT NOT NULL REFERENCES Emp,  
    ...  
);
```

# Chicken and egg

- Suppose we want to say:

```
CREATE TABLE chicken (  
    cID INT PRIMARY KEY,  
    eID INT REFERENCES egg(eID)  
);
```

```
CREATE TABLE egg(  
    eID INT PRIMARY KEY,  
    cID INT REFERENCES chicken(cID)  
);
```

- But, if we simply type the above statements, we'll get an error.
  - The reason is that the `CREATE TABLE` statement for chicken refers to table egg, which hasn't been created yet!
  - Creating egg won't help either, because egg refers to chicken.

# Some first attempt

- First, create chicken and egg without foreign key declarations.

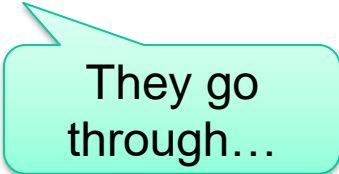
```
CREATE TABLE chicken(  
    cID INT PRIMARY KEY,  
    eID INT  
);
```

```
CREATE TABLE egg(  
    eID INT PRIMARY KEY,  
    cID INT  
);
```

- Then, add foreign key constraints.

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg  
FOREIGN KEY (eID) REFERENCES egg(eID);
```

```
ALTER TABLE egg ADD CONSTRAINT eggREFchicken  
FOREIGN KEY (cID) REFERENCES chicken(cID);
```



They go  
through...

# However, inserting fails...

```
INSERT INTO chicken VALUES(1, 2);
```

\*

```
ERROR ... parent key not found
```

```
INSERT INTO egg VALUES(2, 1);
```

\*

```
ERROR ... parent key not found
```

# Deferrable Constraints

- Solving the problem:
  1. Group several SQL statements into one unit called **transaction**.
  2. Then, tell the SQL system not to check the constraints until the transaction is committed.
- Any constraint may be declared “**DEFERRABLE**” or “**NOT DEFERRABLE**.”
  - **NOT DEFERRABLE** is the default, and means that every time a database modification occurs, the constraint is immediately checked.
  - **DEFERRABLE** means that we have the **option** of telling the system to wait until a transaction is complete before checking the constraint.

# Initially Deferred / Initially Immediate

- If a constraint is **deferrable**, then we may also declare it
  - **INITIALLY DEFERRED**, and the check will be deferred to the end of the current transaction.
  - **INITIALLY IMMEDIATE**, (default) and the check will be made before any modification.

## Example

```
ALTER TABLE chicken ADD  
CONSTRAINT chickenREFegg FOREIGN KEY (eID) REFERENCES egg(eID)  
DEFERRABLE INITIALLY DEFERRED;
```

```
ALTER TABLE egg ADD  
CONSTRAINT eggREFchicken FOREIGN KEY (cID) REFERENCES chicken(cID)  
DEFERRABLE INITIALLY DEFERRED;
```



# Successful Insertions

Now we can finally insert:

```
BEGIN TRANSACTION;  
    INSERT INTO chicken VALUES(1, 2);  
    INSERT INTO egg VALUES(2, 1);  
COMMIT;
```

# Dropping

- Finally, to get rid of the tables, we have to drop the constraints first, because we can't to drop a table that's referenced by another table.

```
ALTER TABLE egg DROP CONSTRAINT eggREFchicken;  
ALTER TABLE chicken DROP CONSTRAINT chickenREFegg;
```

```
DROP TABLE egg;  
DROP TABLE chicken;
```

```
CREATE TABLE chicken (  
    cID INT PRIMARY KEY,  
    eID INT  
);
```

```
CREATE TABLE egg(  
    eID INT PRIMARY KEY,  
    cID INT  
);
```

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg  
FOREIGN KEY (eID) REFERENCES egg(eID) deferrable initially deferred ;
```

```
ALTER TABLE egg ADD CONSTRAINT eggREFchicken  
FOREIGN KEY (cID) REFERENCES chicken(cID) deferrable initially deferred;
```

```
BEGIN TRANSACTION;  
INSERT INTO chicken VALUES(1, 2);  
INSERT INTO egg VALUES(2, 1);  
COMMIT;
```

```
ALTER TABLE egg DROP CONSTRAINT eggREFchicken;  
ALTER TABLE chicken DROP CONSTRAINT chickenREFegg;
```

```
DROP TABLE egg;  
DROP TABLE chicken;
```

# Constraints

## CHECK

# 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:

```
CREATE TABLE Emp (  
  empno int,  
  ename varchar(30)  
  sal int  
  deptno int  
);
```

```
CHECK( ename = UPPER(ename) ),  
CHECK( sal >= 500 ),  
CHECK(deptno BETWEEN 10 AND 100)
```

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

# 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 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,  
    ... ,  
    CHECK (pend > pstart)  
);
```



Table constraint

# 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.

$p \rightarrow q$  is the same as  $(\text{not } p) \text{ OR } q$ .



# 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)  
);
```

```
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)  
);
```

# Constraints

Enforcing business rules with **views**

# Updateable Views - **WITH CHECK OPTION**

## **Only when:**

1. There is only one table in FROM (of the query defining the view).
2. Attribute list in SELECT includes enough attributes that for every tuple inserted into the view, we can fill out the other attributes of the tuple with NULL, and have a tuple in the base table that will yield the inserted tuple in the view.

```
CREATE VIEW ParamountMovies AS
  SELECT title, year
  FROM Movies
  WHERE studioName = 'Paramount'
WITH CHECK OPTION;
```

```
INSERT INTO ParamountMovies VALUES ('Star Trek', 1979);
```

Insertion fails!

Why?

**Rationale** for this behavior is:

- Were the insertion allowed, it would insert a tuple with NULL for studioName in base table Movie.
- However, such a tuple doesn't satisfy the condition for being in the ParamountMovie view! (“**invisible tuple to the view**”)
- Thus, it shouldn't be allowed to get into the database through the ParamountMovie view.

```
CREATE VIEW ParamountMovies2 AS  
    SELECT studioName, title, year  
    FROM Movies  
    WHERE studioName = 'Paramount'  
WITH CHECK OPTION;
```

```
INSERT INTO ParamountMovies2 VALUES ('Paramount', 'Star Trek', 1979);
```

Now it succeeds. Why?

# Views with check option for specialized constraints

- Note that in most DBMS'es 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: Cars

-- Another example: Cars

```
CREATE TABLE cars (  
  car_id serial primary key,  
  car_name varchar(255),  
  brand VARCHAR(20)  
);
```

```
CREATE VIEW audi_cars AS  
  SELECT car_id, car_name, brand  
  FROM cars  
  WHERE brand = 'Audi'  
WITH CHECK OPTION;
```

```
CREATE VIEW ford_cars AS  
  SELECT car_id, car_name, brand  
  FROM cars  
  WHERE brand = 'Ford'  
WITH CHECK OPTION;
```

```
INSERT INTO audi_cars (car_name,brand) VALUES('Q2','Audi');  
INSERT INTO audi_cars (car_name,brand) VALUES('S1','Audi');
```

```
INSERT INTO ford_cars (car_name,brand) VALUES('Edge','Ford');  
INSERT INTO ford_cars (car_name,brand) VALUES('Mustang','Ford');
```

```
INSERT INTO ford_cars(car_name,brand) VALUES('RS6 Avant','Audi');  
-- ERROR: new row violates check option for view "ford_cars"
```

# Another Example

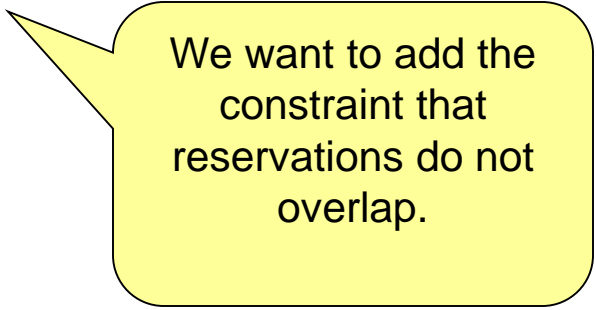
```
CREATE TABLE Hotel (  
    room_nbr INT NOT NULL,  
    arrival_date DATE NOT NULL,  
    departure_date DATE NOT NULL,  
    guest_name CHAR(15) NOT NULL,  
    PRIMARY KEY (room_nbr, arrival_date),  
    CHECK (departure_date > arrival_date)  
);
```

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



# Exercise – Hotel Stays

```
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 H1.guest_name <> H2.guest_name
        (H2.arrival_date < H1.arrival_date AND H1.arrival_date < H2.departure_date)
)
WITH CHECK OPTION;
```



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

# Exercise – Hotel Stays – Inserting

```
INSERT INTO HotelStays (room_nbr, arrival_date, departure_date, guest_name)  
VALUES(1, '2021-01-01', '2021-01-03', 'Larry');
```

This goes Ok.

```
INSERT INTO HotelStays (room_nbr, arrival_date, departure_date, guest_name)  
VALUES(1, '2021-01-02', '2021-01-04', 'Harry');
```

\*

ERROR