

Constraints (1)

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 number NOT NULL,  
  B number NULL,  
  C number  
);
```

```
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 a
ORA-01400: cannot insert NULL into ("*userschema*".*ABC*".*A*").

The **not null/null** constraint can be altered with: **ALTER TABLE ABC MODIFY A null;**
After this modification, the column A can contain null values.

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 NUMBER UNIQUE,  
  B NUMBER  
);
```

```
insert into AB values (4, 5);  
insert into AB values (2, 1);  
insert into AB values (9, 8);  
insert into AB values (6, 9);  
insert into AB values (null,9);  
insert into AB values (null,9);  
insert into AB values (2, 7);
```

- The last statement issues a
ORA-00001: unique constraint (THOMO.SYS_C006985) violated

Constraint names

- Every constraint, has a name. In this case, the name is: **THOMO.SYS_C006985**.
- To find the constraint names and corresponding tables execute:

```
SELECT constraint_name, table_name  
FROM user_constraints;
```

- We can explicitly name the constraint for easier handling.

```
CREATE TABLE ABC (  
    A number,  
    B number,  
    C number,  
    CONSTRAINT my_unique_constraint2 UNIQUE (A,B)  
);
```

It can't have the same name as another constraint even if it is in another table.

Dropping/Adding

Examples

```
ALTER TABLE AB DROP CONSTRAINT SYS_C006985;
```

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

Foreign key constraints

- Specify a column or a list of columns as a foreign key referencing a table.
- Referencing table is called the **child table**, and the referenced table is called the **parent table**.
- **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 childtable has to satisfy one of the following two conditions:

- Foreign key value must either
 - appear as a primary key value in the parenttable, **or**
 - be **null**
 - in case of a composite foreign key, at least one attribute value of the foreign key is **null**
- So, for table **Emp**, an employee must not necessarily work in a department, i.e., for the attribute **deptno**, NULL is admissible.
- 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 table.
- **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,  
    ...  
);
```

Enforcing Foreign-Key Constraints

- If there is a foreign-key constraint from table R to S , two violations are possible:
 1. An insert or update to R introduces values not found in S .
 2. A deletion or update to S causes some tuples of R to “dangle.”

Example.

```
CREATE TABLE Beers (  
    name CHAR(20) PRIMARY KEY,  
    manf CHAR(20)  
);
```

Relation S

```
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    FOREIGN KEY (beer) REFERENCES Beers (name)  
);
```

Relation R

Action taken

- An insert or update to **Sells** that introduces a nonexistent beer must be rejected.
- A deletion or update to **Beers** that removes a beer value found in some tuples of **Sells** can be handled in three ways.
 1. *Default* : Reject the modification.
 2. *Cascade* : Make the same changes in Sells.
 - Deleted beer: delete Sells tuples referencing that beer.
 - Updated beer: change values in Sells.
 3. *Set NULL* : Change the beer to NULL.

Example

Cascade

- Delete **Bud** from **Beers**:
 - Then delete all tuples from **Sells** that have **beer** = 'Bud'.
- Update the **Bud** tuple by changing 'Bud' to 'Budweiser':
 - Then change all **Sells** tuples with **beer** = 'Bud' so that **beer** = 'Budweiser'.

Set NULL

- Delete the **Bud** tuple from **Beers**:
 - Change all tuples of **Sells** that have **beer** = 'Bud' to have **beer** = NULL.
- Update the **Bud** tuple by changing 'Bud' to 'Budweiser':
 - Same change.

Choosing a Policy

Follow the foreign-key declaration by:

ON [UPDATE, DELETE] [SET NULL, CASCADE]

```
CREATE TABLE Sells (  
    bar          CHAR(20) ,  
    beer         CHAR(20) ,  
    price        REAL ,  
    FOREIGN KEY (beer) REFERENCES Beers (name)  
        ON DELETE SET NULL  
        ON UPDATE CASCADE  
);
```

*Oracle doesn't allow ON UPDATE
options*

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

- To work around this problem, first, create chicken and egg without foreign key declarations.

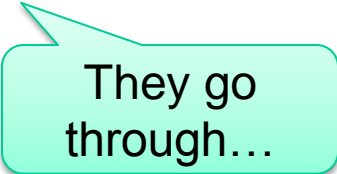
```
CREATE TABLE chicken(  
    cID INT PRIMARY KEY,  
    eID INT  
);
```

```
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 at line 1:

ORA-02291: integrity constraint (THOMO.CHICKENREFEGG) violated -
parent key not found

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

*

ERROR at line 1:

ORA-02291: integrity constraint (THOMO.EGGREFCHICKEN) violated -
parent key not found

Deferrable Constraints

- Solving the problem:
 1. We need the ability to group several SQL statements into one unit called **transaction**.
 2. Then, we need a way to 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.

In ORACLE SQLPlus, when we login, a transaction is automatically created. We can (positively) finish this transaction by calling “commit” and a new transaction is created.

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.
 - But, because the constraint is deferrable, we have the option of later deciding to defer checking (SET CONSTRAINT MyConstraint DEFERRED).

Example

Here we declare the constraints DEFERRABLE and INITIALLY DEFERRED.

```
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:

```
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;
```

Another Example of Deferring

- Studio and President