The Sequel to SQL

Level 2 – Section 1

Identifying Constraints

THE SEQUEL TO SQL

The Default Behavior of a Table

The default behavior of a table column is to allow insertion of NULL values.

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50),
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash
3	*NULL*	Merchandise

*Inserts row with a NULL value set for the **name** column*

```
INSERT INTO Promotions (id, category)
VALUES (3, 'Merchandise');
```

*When column name is omitted, the default value is **NULL**.*

Adding the NOT NULL Column Constraint

A NOT NULL column constraint ensures values **cannot** be NULL.

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50) NOT NULL,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash

```
INSERT INTO Promotions (id, category)
VALUES (3, 'Merchandise');
```

```
ERROR:  null value in column "name" violates not-null constraint
DETAIL:  Failing row contains (3, null, Merchandise).
```

We want this error to happen so we don't have bad data.

Why Use Constraints?

The default behavior of database tables can be too permissive.

Constraints can help with these shortcomings!

- Prevent **NULL** values
- Ensure column values are **unique**
- Provide additional **validations**

Preventing Unwanted Duplicates

The default behavior of a table column allows insertion of duplicate values.

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50) NOT NULL,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash
3	Giveaways	Merchandise
4	Matinee	Discount

Inserts duplicate value on the ***name*** *column:*

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Matinee', 'Discount');
```

THE SEQUEL TO SQL

Adding the UNIQUE Column Constraint

The UNIQUE constraint uniquely identifies each field in a table.

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50) NOT NULL UNIQUE,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash
3	Giveaways	Merchandise

*Cannot insert duplicate values with **UNIQUE** constraint.*

More than 1 constraint can be used on a column!

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Matinee', 'Discount');
```

```
ERROR:  duplicate key value violates unique constraint "promotions_name_key"
DETAIL:  Key (name)=(Matinee) already exists.
```

THE SEQUEL TO SQL

Assigning Constraint Names

The database will automatically assign computer-generated constraint names.

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50) NOT NULL UNIQUE,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash
3	Giveaways	Merchandise

"promotions_name_key" is the constraint name automatically assigned by the database.

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Matinee', 'Discount');
```

```
ERROR:  duplicate key value violates unique constraint "promotions_name_key"
DETAIL:  Key (name)=(Matinee) already exists.
```

THE SEQUEL TO SQL

Creating a UNIQUE Table Constraint

Assigning a name to a constraint can help you easily find it when you choose to alter your constraint.

```
CREATE TABLE Promotions
(
  id int ,
  name varchar(50)  NOT NULL ,
  category varchar(15) ,
  CONSTRAINT unique_name  UNIQUE (name)
);
```

"unique_name" is what we assigned for our custom constraint name.

Using Our Constraint Name

Assigning a constraint name we can remember helps if we have to troubleshoot any constraint errors.

```
CREATE TABLE Promotions
(
  id int ,
  name varchar(50)  NOT NULL ,
  category varchar(15) ,
  CONSTRAINT unique_name  UNIQUE (name)
);
```

Now we can see the constraint name in the error message.

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Matinee', 'Discount');
```

```
ERROR:  duplicate key value violates unique constraint "unique_name"
DETAIL:  Key (name)=(Matinee) already exists.
```

THE SEQUEL TO SQL

Column vs. Table Constraint

Except for NOT NULL, every column constraint can also be written as a table constraint.

column constraint

```
CREATE TABLE Promotions
(
  id int,
  name varchar(50) NOT NULL UNIQUE,
  category varchar(15)
);
```

table constraint

```
CREATE TABLE Promotions
(
  id int ,
  name varchar(50)  NOT NULL ,
  category varchar(15) ,
  CONSTRAINT unique_name  UNIQUE (name)
);
```

Same behavior

THE SEQUEL TO SQL

Ensuring 2 Columns Are Unique

What if we didn't want to allow an insert that has the same name *and* category?

```
CREATE TABLE Promotions
(
  id int ,
  name varchar(50)  NOT NULL ,
  category varchar(15) ,
  CONSTRAINT unique_name  UNIQUE (name, category)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Matinee	Non-cash
3	Giveaways	Merchandise

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Half Off', 'Discount');
```

We already have this unique combination.

```
ERROR:  duplicate key value violates unique constraint "unique_name"
DETAIL:  Key (name, category)=(Half Off, Discount) already exists.
```

Most Tables Should Have a Primary Key

As the primary key, the id column needs to *uniquely* identify every row in this table.

```
CREATE TABLE Promotions
(
  id int ,
  name varchar(50) ,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Reward Points	Cash Back
3	Matinee	Non-cash
4	Giveaways	Merchandise
4	Buy-1-Get-1	Discount
NULL	Buy-5-Get-2	Discount

Shouldn't allow duplicates or NULL values

Making a Column a Primary Key

```
CREATE TABLE Promotions
(
  id int PRIMARY KEY,
  name varchar(50) ,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Reward Points	Cash Back
3	Matinee	Non-cash
4	Giveaways	Merchandise

*Adding a PRIMARY KEY constraint means that column cannot be **NULL** and must be **UNIQUE**.*

A Primary Key Prevents Duplicate Entries

```
CREATE TABLE Promotions
(
  id int PRIMARY KEY,
  name varchar(50) ,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Reward Points	Cash Back
3	Matinee	Non-cash
4	Giveaways	Merchandise

```
INSERT INTO Promotions (id, name, category)
VALUES (4, 'Free Shirt', 'Merchandise');
```

Cannot insert duplicates

```
ERROR:  duplicate key value violates unique constraint "promotions_pkey"
DETAIL:  Key (id)=(4) already exists.
```

A Primary Key Prevents NULL Values

```
CREATE TABLE Promotions
(
  id int PRIMARY KEY ,
  name varchar(50) ,
  category varchar(15)
);
```

Promotions

id	name	category
1	Half Off	Discount
2	Reward Points	Cash Back
3	Matinee	Non-cash
4	Giveaways	Merchandise

```
INSERT INTO Promotions (name, category)
VALUES ('Buy-1-Get-1', 'Merchandise');
```

Cannot insert NULL values

```
ERROR:  null value in column "id" violates not-null constraint
DETAIL:  Failing row contains (null, Buy-1-Get-1, Merchandise).
```

Difference Between PK and NOT NULL + UNIQUE

A PRIMARY KEY constraint automatically accomplishes the same goals of **both** the UNIQUE and the NOT NULL constraint. However, it's *not* the same thing.

PRIMARY KEY vs. **NOT NULL + UNIQUE**

Can only be defined once per table

```sql
CREATE TABLE Promotions
(
  id int PRIMARY KEY,
  name varchar(50) NOT NULL UNIQUE,
  category varchar(15)
);
```

Can be used multiple times per table

```sql
CREATE TABLE Promotions
(
  id int PRIMARY KEY,
  name varchar(50) NOT NULL UNIQUE,
  category varchar(15) NOT NULL UNIQUE,
);
```