

CS252: Fundamentals of Relational Databases

Lecture Slides 2

presented by
Timothy Heron*

*E-mail: theron@dcs.warwick.ac.uk

Interim Summary 2

Material covered so far:

- Everything for data stored in one table.
- Creating tables without constraints.
- Inserting rows and partial rows, NULL values.
- Selecting data from tables (`SELECT ... FROM`).
- Predicates (`WHERE`) and predicate combination (`AND` and `OR`).
- Commitment and rollback.
- Deleting (`DELETE`) and updating (`UPDATE`) rows and values.
- Subqueries and embedding `SELECT`.

Problems with One Table

So far all data is in one large table. This is *zeroth normal form* and may contain *redundancy*.

Far more useful, flexible and reusable to have several small tables and split the information between them.

For example we may want to record information about the musicians within each band. We do not want to put this information in our *Collection* table since we will have to repeat it for each artist.

Consider classical music albums where data includes :

- conductor — “Simon Rattle”
- soloist — “John Todd”
- composer — “Elgar”
- work — “Cello Concerto”
- orchestra — “Berlin Philharmonic”

These define attributes of one piece on one CD. One CD may contain several equally relevant pieces of music.

We could define separate tables for the classical albums in our Collection. These could contain columns for:

- Track ranges (e.g. 3 to 6, composer, work.)
- Performer, composer, orchestra, soloist.

Multiple Tables

We want to split the information up into separate tables but we need some way of linking the information together.

⇒ Need a unique reference number for each CD — the 12 digit *barcode*.

To record the information specific to pop albums :

Pop_albums

barcode	artist	album
042282289827	U2	The Unforgettable Fire
042284229920	U2	Rattle and Hum
731451034725	U2	Achtung Baby
026734000524	Underworld	Second Toughest in the Infants
724384491321	The Verve	Urban Hymns
724385583223	Foo Fighters	The Colour and the Shape

column barcode format 999999999999;

Example: Representing Everything

To represent everything to do with pop albums that was represented in the single table previously:

<i>CD_company</i>		<i>CD_year</i>	
barcode	company	barcode	year
042282289827	Island	042282289827	1984
042284229920	Island	042284229920	1988
731451034725	Island	731451034725	1991
026734000524	Junior	026734000524	1996
724384491321	Virgin	724384491321	1997
724385583223	Capital	724385583223	1997
724383719020	EMI	724383719020	1996
891030505032	Naxos	891030505032	1992

We also need a similar table for *Number of tracks*.

Example: Representing Band Members

We can create a new table to store information about band members :

Band_members

artist	member1	member2	member3	member4
U2	Bono	Edge	Clayton	Mullen Jr.
Foo Fighters	Grohl	Smear	Mendel	Goldsmith

Here we can use the artist attribute in our *Pop_albums* table to lookup information about each artist.

In order to be able to lookup the artist we have to make sure that the artist name is unique. We can add constraints to enforce uniqueness.

Advanced Table Creation

The description of a table can include more than just column name and data type. There are *constraints* that can be applied to columns:

- Numerical precision, string length.
- Columns with no NULL values.
- Candidate keys — unique data.
- Primary keys.
- Foreign keys.
- Check constraint — predicate.

Constraints **must** be satisfied before data is inserted into a table, using UPDATE and INSERT. Oracle does this for you.

CREATE TABLE Syntax

Syntax of CREATE statement:

```
CREATE TABLE table_name (
  (column_name data_type [column_constraint],)*
  column_name data_type [column_constraint]
  [, table_constraints] );
```

Constraints

Two kinds of constraint:

Column Constraint Applies all values in one column. Specified after each column definition.

Table Constraint Must be satisfied for the entire table. Specified at the end of the table definition after the columns have been defined.

Number Data Types

NUMBER Space for 40 digits. Also “+”, “-”, “.” and “E”.

Possible numbers: 123, 123.45, $123E - 3 = 0.123$,
 $43.2E7 = 43.2 \times 10^7$.

NUMBER(*size*) Same as NUMBER with *size* digits and no decimal point.

Maximum value for *size* is 105.

For a NUMBER(3), maximum value is “999” and minimum is “-999”.

NUMBER(*size*, *d*) Specify NUMBER *size* and number of digits *d* following decimal point.

For a NUMBER(5,2) maximum is “999.99”.

Rounding on Insertion

Note that the `INTEGER` data type is the `NUMBER` data type with the constraint that there is no decimal point allowed.

Values are rounded on insertion. Rounding to an integer:

- $0.0 \leftrightarrow 0.49$ rounded to 0
- $0.5 \leftrightarrow 0.9$ rounded to 1

Examples:

Data Type	Inserted	Stored
<code>NUMBER(4,1)</code>	123.40	123.4
	123.45	123.5
	1234.50	FAIL!
<code>NUMBER(4)</code>	123.40	123
	1234.50	1235
	12345.00	FAIL!
<code>NUMBER(4,-1)</code>	123.40	120
	125.00	130
	1234.50	1230
<code>NUMBER</code>	123.445	123.445
	1234.5	1234.5

Character and Date Data Types

Data types for representing strings and dates :

`DATE` Represents dates and times. Range from January 1, 4712 B.C. to December 31, 4712 A.D..

`CHAR(size)` A string of characters of length *size*.

Choose carefully to avoid truncation ! Each row entry is of exactly this size.

Maximum number of characters is 2000.

`VARCHAR(size)` A string of characters up to the maximum length of *size*.

Storage space varies along with what is stored.

Maximum number of characters is 4000.

Not Null Columns

Simplest constraint — `NOT NULL` — describes a column that must contain a value that is not set to `NULL`.

No `INSERT` or `UPDATE` that sets this column to `NULL` will be allowed.

Creating a table to store quantities of a CD in a collection:

```
CREATE TABLE Quantity (
    barcode NUMBER(12) NOT NULL,
    quantity NUMBER(3) NOT NULL
);
```

This declares that barcodes are to be stored in 12 digits and quantity is an integer (maximum 999).

It is possible to have a negative quantity!

Candidate Keys

A *Candidate Key* is a combination of one or more columns, the values of which uniquely identify each row of the table.

Column constraint — use keyword **UNIQUE**:

```
CREATE TABLE Quantity (
  barcode NUMBER(12) NOT NULL UNIQUE,
  quantity NUMBER(3) NOT NULL
);
```

The following SQL is successful:

```
INSERT INTO Quantity
VALUES (123456789012, 2);

1 row created.
```

However, this insertion will **then** fail:

```
INSERT INTO Quantity
VALUES (123456789012, 1);
...
Error at line 1:
ORA-00001: unique constraint (XXX) violated.
```

More than one Candidate Key

For the *Pop_albums* table, there are two candidate keys:

- The **barcode**
- The **artist** with the **album** combined.

Can express these in a combination of column and table constraints.

```
CREATE TABLE Pop_album (
  barcode NUMBER(12) NOT NULL UNIQUE,
  artist VARCHAR(100) NOT NULL,
  album VARCHAR(150) NOT NULL,
  UNIQUE (artist, album)
);
```

Primary Key

Tables in *first normal form* have a *Primary Key*.

A *Primary Key* is a candidate key, except there can be only one per table.

In all tables with a **barcode** column, the primary key has been chosen to be this column.

```
CREATE TABLE Quantity (
  barcode NUMBER(12) PRIMARY KEY,
  quantity NUMBER(3) NOT NULL
);
```

It is possible to combine candidate keys and primary keys — imagine table *Pop_albums* with one of the “**UNIQUE**” keywords replaced by “**PRIMARY KEY**”.

PRIMARY KEY vs UNIQUE

There are a couple of differences between a candidate key and a primary key in SQL :

- A primary key cannot contain NULL values
- A primary key has an 'index' of that primary key is automatically added to the table.

So we can dispense with the 'PRIMARY KEY' and replace it with a column that is UNIQUE and NOT NULL and has an index on it.

Indexes are used to improve performance of certain queries.

We will cover indexes later on in the course.

Foreign Key

Imagine a table that stored information about artists:

Band_members

artist	mbr1	mbr2	mbr3	mbr4
U2	Bono	Edge	Clayton	Mullen Jr.
Foo Fighters	Grohl	Smear	Mendel	Goldsmith
⋮	⋮	⋮	⋮	⋮

The primary key is the **artist** column. In the *Pop_albums* table, **artist** is a *Foreign Key*.

A foreign key is a primary key from another table.

Use REFERENCES to establish a foreign key constraint (also called a referential constraint).

```
CREATE TABLE Pop_albums (
  barcode NUMBER(12) PRIMARY KEY,
  artist VARCHAR(100)
  NOT NULL REFERENCES Band_members(artist),
  album VARCHAR(150) NOT NULL,
  UNIQUE (artist, album)
);
```

Albums can only be inserted into the *Pop_albums* table if the artist already exists in the *Band_members* table.

Check Constraints

Check constraints are logical conditions to check prior to insertion or update.

Use keyword CHECK followed by a predicate:

```
CREATE TABLE Quantity (
  barcode NUMBER(12) PRIMARY KEY,
  quantity NUMBER(3) NOT NULL CHECK (quantity >= 0)
);
```

This ensures that the **quantity** column is never negative.

Cannot use subqueries or some functions.

Table Constraints

Table Constraints specify constraints at the end of the table definition.

```
CREATE TABLE Quantity (
  barcode NUMBER(12),
  quantity NUMBER(3),
  CHECK (barcode IS NOT NULL),
  CHECK (quantity >= 0),
  PRIMARY KEY (barcode)
);
```

Specifying a Particular Table Within a Query

Often tables have attributes with the same name. To distinguish between these attributes use the dot notation :

```
table.column
```

```
SELECT Collection.artist FROM Collection;
```

Selecting From Multiple Tables

In relational algebra, this is a *join*.

Consider joining the tables *Pop_albums* and *CD_year*:

```
SELECT artist, album, year
FROM Pop_albums, CD_year
WHERE Pop_albums.barcode = CD_year.barcode;
```

ARTIST	ALBUM	YEAR
U2	The Unforgettable Fire	1984
U2	Rattle and Hum	1988
U2	Achtung Baby	1991
Underworld	Second Toughest in the Infants	1996
The Verve	Urban Hymns	1997
Foo Fighters	The Colour and the Shape	1997

Multiple Joins

To get back the table from the first seminar, use:

```
SELECT artist, album, tracks, company, year
FROM Pop_albums, CD_tracks, CD_year, CD_company
WHERE Pop_albums.barcode = CD_year.barcode
  AND CD_year.barcode = CD_company.barcode
  AND CD_company.barcode = CD_tracks.barcode;
```

Table Management

So far there has been no mention of how to manage (delete or modify) table structures. This is a complicated issue.

To delete a table — when you don't need it:

```
DROP TABLE table_name;
```

You can rollback such a change.

To empty a table of all its values while keeping its structure, use:

```
TRUNCATE TABLE table_name;
```

Rollback is not possible — use with extreme care.

Altering Tables

Adding Columns

Use the ALTER TABLE and ADD statements:

```
ALTER TABLE table_name ADD ( ... );
```

Body of ALTER TABLE same as body of CREATE TABLE.

It is not possible to ADD a NOT NULL column.

Altering Tables

Modifying Columns

Use the ALTER TABLE and MODIFY statements:

```
ALTER TABLE table_name MODIFY ( ... );
```

Imagine there is an International move to standardise barcodes so that they have 14 digits instead of 12.

```
ALTER TABLE Quantity MODIFY (
  barcode NUMBER(14) );
```

Altering Tables

Renaming Columns

Use the ALTER TABLE and RENAME COLUMN statements:

```
ALTER TABLE table_name RENAME COLUMN column1 TO column2;
```

```
ALTER TABLE Quantity RENAME COLUMN barcode TO serial_no;
```

Rules for Table Alteration

These are the rules for adding a column to a table:

- You may add a column at any time if **NOT NULL** is not specified.
- You may add a **NOT NULL** column in three steps:
 1. Add the column without **NOT NULL**.
 2. Fill every row of the column with data.
 3. Modify the column to be **NOT NULL**.

These are the rules for modifying a column:

- You can increase a **CHAR** column's width or a **NUMBER** column's precision at any time.
- You can add or decrease the number of decimal places in a **NUMBER** column at any time.

In addition, if a column is **NULL** for every row of a table, you can make any of these changes:

- Change the data type.
- Decrease a **CHAR** columns width or a **NUMBER** columns precision.

Interim Summary 3

- Data description language:
 - Creating tables with constraints.
 - Oracle data types (apart from **DATE**).
- Constraints: **NOT NULL**, candidate keys, primary keys, foreign keys, **CHECK** predicates.
- Single and multiple table joins using **SELECT**.
- Deleting entire tables with **DROP TABLE** or their entire contents with **TRUNCATE TABLE**.
- **ALTER TABLE** for modifying and adding to existing tables.