CS 252: Fundamentals of Relational Databases: SQL2

Dr. Alexandra I. Cristea
http://www.dcs.warwick.ac.uk/~acristea/

Interim Summary
Material covered so far:
• Everything for data stored in one table.
• Creating base tables without constraints.
• Inserting rows and partial rows, NULL.
• Expressing queries using (SELECT ... FROM ... WHERE).
• Predicates (WHERE) and predicate combination (AND and OR).
• Commitment and rollback.
• Deleting (DELETE) and updating (UPDATE) rows.
• Subqueries.

This lecture
Data description language:
– Creating tables with constraints.
– Oracle data types.
• 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.

Problems with One Table
• So far: just one example base table.
• In practice: several; usually impractical and undesirable to cram everything into one!
– E.g., some CDs contain several separately identifiable pieces of music. If we include information about these in extra columns of our Collection table, then we will have to repeat the general information about a CD containing \( n \) pieces of music \( n \) times!

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 (Base) 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:
### Example: Multiple base tables

To split information to do with pop albums that was represented in a single base table previously:

<table>
<thead>
<tr>
<th>company</th>
<th>barcode</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virgin</td>
<td>724385583223</td>
<td>1997</td>
</tr>
<tr>
<td>Junior</td>
<td>026734000524</td>
<td>1996</td>
</tr>
<tr>
<td>Island</td>
<td>731451034725</td>
<td>1988</td>
</tr>
<tr>
<td>Island</td>
<td>042284229920</td>
<td>1984</td>
</tr>
<tr>
<td>Island</td>
<td>042282289827</td>
<td>1984</td>
</tr>
</tbody>
</table>

We also need a similar table for the number of tracks.

### Example: Representing Band Members

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

<table>
<thead>
<tr>
<th>artist</th>
<th>member1</th>
<th>member2</th>
<th>member3</th>
<th>member4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FooFighters</td>
<td>Goldsmith</td>
<td>Mendel</td>
<td>Smear</td>
<td>Grohl</td>
</tr>
<tr>
<td>U2</td>
<td>Mullen</td>
<td>Jr.</td>
<td>Clayton</td>
<td>Edge</td>
</tr>
<tr>
<td>U2</td>
<td>Bono</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** Very bad design, not recommended! See the notes for explanation.

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 definition of a column can include more than just column name and data type. The definition of a table consists of a set of column definitions and a set of constraint declarations.

Constraints that can be declared include:
- Numerical precision, string length.
- Prohibiting the appearance of `NULL` in a column.
- Candidate keys - unique data.
- Primary keys.
- Foreign keys.
- Check constraint - predicate.

Constraints must be satisfied at all times and so are conceptually checked every time the database is updated (`UPDATE` and `INSERT`). The SQL DBMS does the checking (e.g., 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**: Included in a column definition. Applies to each value in that column.

**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 Type**

NUMBER Space for 40 digits. Also “+”, “-”, “.” and “E”.
Possible numbers: 123, 123.45, 123E - 3 = 0.123,
43.2E7 = 43.2 x 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”.

**Examples:**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Inserted</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER(4, 1)</td>
<td>123.4</td>
<td>123.4</td>
</tr>
<tr>
<td></td>
<td>123.45</td>
<td>123.5</td>
</tr>
<tr>
<td></td>
<td>123.5</td>
<td>FAIL</td>
</tr>
<tr>
<td>NUMBER(4)</td>
<td>123.4</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>123.45</td>
<td>1235</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>FAIL</td>
</tr>
<tr>
<td>NUMBER(4.1)</td>
<td>123.4</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>123.45</td>
<td>123.45</td>
</tr>
<tr>
<td></td>
<td>123.5</td>
<td>123.5</td>
</tr>
</tbody>
</table>

**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 ÷ 0.49 rounded to 0
- 0.5 ÷ 0.9 rounded to 1

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

Maximum number of characters is 4000.

**Not Null Columns**

Simplest constraint - NOT NULL - in a column definition, specifies that NULL is prohibited from appearing anywhere in that column.

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:

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

This specifies that bar codes are integers of no more than twelve decimal digits and quantities are integers of no more than three digits (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:

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

The following SQL is successful:

```sql
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_albums {
  barcode NUMBER(12) NOT NULL UNIQUE,
  artist VARCHAR(100) NOT NULL,
  album VARCHAR(150) NOT NULL,
  UNIQUE (artist, album)
};
```

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 use both candidate keys and primary keys – imagine table Pop albums with one of the "UNIQUE" keywords replaced by "PRIMARY KEY" – what is the difference?

```
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.

PRIMARY KEY vs UNIQUE

There are a couple of differences between a candidate key and a primary key in SQL:
- PRIMARY KEY implies NOT NULL for each column of the specified key
- A primary key has an index of that primary key that 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:

```
<table>
<thead>
<tr>
<th>artist</th>
<th>member1</th>
<th>member2</th>
<th>member3</th>
<th>member4</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>Bone</td>
<td>Edge</td>
<td>Clayton</td>
<td>Mullen Jr.</td>
</tr>
<tr>
<td>Foo Fighters</td>
<td>Grohl</td>
<td>Smear</td>
<td>Mondal</td>
<td>Goldsmith</td>
</tr>
</tbody>
</table>
```

The primary key is the artist column. In the Pop_albums table, artist is a Foreign Key. A foreign key is a primary key (or unique) from another table. Use REFERENCES to establish a foreign key constraint (also called a referential constraint).
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 columns 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;
```

<table>
<thead>
<tr>
<th>ARTIST</th>
<th>ALBUM</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>The Unforgettable Fire</td>
<td>1984</td>
</tr>
<tr>
<td>U2</td>
<td>Rattle and Hum</td>
<td>1988</td>
</tr>
<tr>
<td>U2</td>
<td>Achtung Baby</td>
<td>1991</td>
</tr>
<tr>
<td>Underworld</td>
<td>Second Toughest in the Infants</td>
<td>1996</td>
</tr>
<tr>
<td>The Verve</td>
<td>Urban Rhymes</td>
<td>1997</td>
</tr>
<tr>
<td>Foo Fighters</td>
<td>The Colour and the Shape</td>
<td>1997</td>
</tr>
<tr>
<td>CS252</td>
<td>Fundamentals of Relational Databases</td>
<td>1997</td>
</tr>
</tbody>
</table>

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 table elements (i.e., column definitions and constraints):

```
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.

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) );
```

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;
```

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 column’s width or a NUMBER column’s precision.

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.

Interim Summary
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.