CS252: Fundamentals of Relational Databases
Lecture Slides 2
presented by
Timothy Heron*

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:

<table>
<thead>
<tr>
<th>barcode</th>
<th>artist</th>
<th>album</th>
</tr>
</thead>
<tbody>
<tr>
<td>042282289827</td>
<td>U2</td>
<td>The Unforgettable Fire</td>
</tr>
<tr>
<td>042284229920</td>
<td>U2</td>
<td>Rattle and Hum</td>
</tr>
<tr>
<td>731451034725</td>
<td>U2</td>
<td>Achtung Baby</td>
</tr>
<tr>
<td>026734000524</td>
<td>Underworld</td>
<td>Second Toughest in the Infants</td>
</tr>
<tr>
<td>724384491321</td>
<td>The Verve</td>
<td>Urban Hymns</td>
</tr>
<tr>
<td>724385583223</td>
<td>Foo Fighters</td>
<td>The Colour and the Shape</td>
</tr>
</tbody>
</table>

column barcode format 999999999990;

Example: Representing Everything

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

<table>
<thead>
<tr>
<th>CD_company</th>
<th>CD_year</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>barcode</td>
<td>company</td>
<td></td>
</tr>
<tr>
<td>042282289827</td>
<td>Island</td>
<td></td>
</tr>
<tr>
<td>042284229920</td>
<td>Island</td>
<td></td>
</tr>
<tr>
<td>731451034725</td>
<td>Island</td>
<td></td>
</tr>
<tr>
<td>026734000524</td>
<td>Junior</td>
<td></td>
</tr>
<tr>
<td>724384491321</td>
<td>Virgin</td>
<td></td>
</tr>
<tr>
<td>724385583223</td>
<td>Capital</td>
<td></td>
</tr>
<tr>
<td>724383719020</td>
<td>EMI</td>
<td></td>
</tr>
<tr>
<td>891030505023</td>
<td>Naxos</td>
<td></td>
</tr>
<tr>
<td>barcode</td>
<td>year</td>
<td></td>
</tr>
<tr>
<td>042282289827</td>
<td>1984</td>
<td></td>
</tr>
<tr>
<td>042284229920</td>
<td>1988</td>
<td></td>
</tr>
<tr>
<td>731451034725</td>
<td>1991</td>
<td></td>
</tr>
<tr>
<td>026734000524</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>724384491321</td>
<td>1997</td>
<td></td>
</tr>
<tr>
<td>724385583223</td>
<td>1997</td>
<td></td>
</tr>
<tr>
<td>724383719020</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>891030505023</td>
<td>1992</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Band_members</th>
<th>member1</th>
<th>member2</th>
<th>member3</th>
<th>member4</th>
</tr>
</thead>
<tbody>
<tr>
<td>artist</td>
<td>Bono</td>
<td>Edge</td>
<td>Clayton</td>
<td>Mullen Jr.</td>
</tr>
<tr>
<td>Foo Fighters</td>
<td>Grohl</td>
<td>Smear</td>
<td>Mendel</td>
<td>Goldsmith</td>
</tr>
</tbody>
</table>
Advanced Table Creation

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

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

Constraints \textbf{must} be satisfied before data is inserted into a table, using \texttt{UPDATE} and \texttt{INSERT}. Oracle does this for you.

Constraints

Two kinds of constraint:

- \textbf{Column Constraint} Applies all values in one column. Specified after each column definition.
- \textbf{Table Constraint} Must be satisfied for the entire table. Specified at the end of the table definition after the columns have been defined.

CREATE TABLE Syntax

Syntax of \texttt{CREATE} statement:

\begin{verbatim}
CREATE TABLE table_name ( 
(column_name data_type [column_constraint]),*) 
(column_name data_type [column_constraint] 
[,] table_constraints) 

\end{verbatim}
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:

- \( 0.0 \rightarrow 0.49 \) rounded to 0
- \( 0.5 \rightarrow 0.9 \) rounded to 1

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

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.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Inserted</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123.40</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>1234.50</td>
<td>1235</td>
</tr>
<tr>
<td></td>
<td>12345.00</td>
<td>FAIL!</td>
</tr>
<tr>
<td></td>
<td>123.40</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>125.00</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>1234.50</td>
<td>1230</td>
</tr>
<tr>
<td></td>
<td>123.445</td>
<td>123.445</td>
</tr>
<tr>
<td></td>
<td>1234.5</td>
<td>1234.5</td>
</tr>
</tbody>
</table>

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:

```sql
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:

<table>
<thead>
<tr>
<th>Band_members</th>
</tr>
</thead>
<tbody>
<tr>
<td>artist</td>
</tr>
<tr>
<td>mbr1</td>
</tr>
<tr>
<td>mbr2</td>
</tr>
<tr>
<td>mbr3</td>
</tr>
<tr>
<td>mbr4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>artist</th>
<th>mbr1</th>
<th>mbr2</th>
<th>mbr3</th>
<th>mbr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>Bono</td>
<td>Edge</td>
<td>Clayton</td>
<td>Mullen Jr.</td>
</tr>
<tr>
<td>Foo Fighters</td>
<td>Grohl</td>
<td>Smear</td>
<td>Mendel</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 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:

```sql
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.

```sql
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:

```sql
SELECT table.column
```

**Selecting From Multiple Tables**

In relational algebra, this is a *join*.

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

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

**Multiple Joins**

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

```sql
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:

```sql
DROP TABLE table_name;
```

You can rollback such a change.

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

```sql
TRUNCATE TABLE table_name;
```

Rollback is not possible — use with extreme care.

**Altering Tables**

**Adding Columns**

Use the `ALTER TABLE` and `ADD` statements:

```sql
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:

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

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

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

**Renaming Columns**

Use the `ALTER TABLE` and `RENAME COLUMN` statements:

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

```sql
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.