Topics

- Quick tutorial in relational databases from a software engineering viewpoint
  - tables and referential integrity
  - conceptual versus logical database models
  - business rules and application logic
- Mapping transient objects to persistent records
  - object databases, SQL:1999, and impedance mismatch
  - object-relational mapping
- Database schema for EM
**Database characteristics**

- large
- persistent
- multi-user sharable
- recoverable
- consistent
- secure
- extensible

**Relational model**

- **Database model** – an abstraction that presents the server data to the client programs in more understandable terms than the bits and bytes
  - The dominant database model for business information systems is the **relational model**
  - **Structured Query Language** (SQL) is the relational language that the application programs must use to gain access to the database

- **The relational model**
  - presents data as **records** (rows) in **tables** (relations)
  - records in different tables (or in the same table) cannot be linked by user-visible **navigational links**
    - **referential integrity**
**Table**

Table consists of a
- fixed number of **columns**
  - must be primitive data types, such as numbers or strings of characters
- varying number of **records** (rows)

### Select statement to get table content

```sql
SQL> select * from movie;

+----------+-----------------------------+---------------------+
<table>
<thead>
<tr>
<th>MOVIE_CODE</th>
<th>MOVIE_TITLE</th>
<th>DIRECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Interview with the Vampire</td>
<td>Neil Jordan</td>
</tr>
<tr>
<td>11</td>
<td>The Birdcage</td>
<td>Mike Nichols</td>
</tr>
</tbody>
</table>
```

**Referential integrity**

Uses the notion of a foreign key to link records in one table to records in another (or even the same) table.
- **foreign key** is a set of columns (frequently just one column), which values correspond to the values of the primary key in another (or the same) table.
Conceptual ER database model

- Model at a higher-level of abstraction, which does not make a commitment to the database technology.
- The **Entity-Relationship** (ER) is the best known technique for conceptual database modeling.

![ER Diagram](image)

Conceptual UML database model

![UML Diagram](image)
**Implementing business rules**

- **Declarative referential integrity** can be used in relational database to implement many simple and repetitive business rules.
- More complex and exclusive business rules require the procedural referential integrity enforcement = triggers.

---

**Declarative referential integrity**

**MovieActor**

```sql
SQL> CREATE TABLE listed_as (  2    actor_code   NUMBER(5),  3    movie_code   NUMBER(5),  4    position NUMBER(2),  5    PRIMARY KEY (actor_code, movie_code)  6    ) ;
SQL> CREATE TABLE actor (  2    actor_code   NUMBER(5),  3    actor_name   VARCHAR2(25),  4    PRIMARY KEY (actor_code)  5    ) ;
SQL> ALTER TABLE listed_as  2    ADD CONSTRAINT fk_const3 FOREIGN KEY (actor_code)  3    REFERENCES actor  4    ON DELETE CASCADE;
```
**Procedural referential integrity**

Procedural referential integrity with delete trigger in MovieActor

```sql
create trigger tda_movie after delete
on movie for each row
begin
    -- Delete all children in "listed_as"
    delete listed_as
    where movie_code = :old.movie_code;
end;
/
```

**Programming with embedded SQL**

Embedded SQL select in MovieActor application program

```sql
-- Display the leading actor for each movie
EXECUTE SELECT movie_title, actor_name
FROM movie m, listed_as l, actor a
WHERE m.movie_code = l.movie_code
    AND l.actor_code = a.actor_code
    AND l.position = 1;
```
Programming with stored procedures

Stored procedure in MovieActor database

SQL> -- String search for a movie using stored procedure
SQL> CREATE OR REPLACE PROCEDURE string_search (string IN VARCHAR2) AS
2  CURSOR c1 IS
3    SELECT movie_title AS found
4      FROM movie
5      WHERE UPPER(movie_title) LIKE '%'||UPPER(string)||'%';
6  BEGIN
7      FOR c1rec IN c1 LOOP
8          dbms_output.put_line
9            ('Found movie title: '||LPAD(c1rec.found,30));
10     END LOOP;
11  END string_search;
12  /
Procedure created.
SQL> EXECUTE string_search('vamp');
Found movie title: Interview with the Vampire
SQL> EXECUTE string_search('the');
Found movie title: Interview with the Vampire
Found movie title: The Birdcage

Indexes

- Index is a data structure, separate from data pages that store table records, which consists of a hierarchical tree of index nodes
- physical data independence – application programs can perform their database tasks with or without indexes.
Object Databases and SQL:1999

- **Object database model** is a competitor of the relational model
  - Object DBMS can serve as Object Storage API
    - to provide a layer of software between the application program and the relational (or other) database in order to map application's objects to relational records and vice versa
    - the mapping may involve the use of a persistent object database
    - results in moving much of the responsibilities performed by the domain and foundation classes to the Object Storage API
- **SQL:1999** – the latest standard that attempts to add object-oriented features to the relational model.

Impedance mismatch

- **Impedance mismatch** – the impossibility of using the application language (such as Java) to directly manipulate the data in the database without the need to engage SQL
  - SQL manipulates data as sets of records
  - application languages are procedural in nature and manipulate data as individual records
  - SQL provides the cursor mechanism to address the mismatch
- Impedence mismatch is a reason to develop a mapping strategy from classes to tables and vice versa
From one-to-many association and aggregation...

One-to-many association

0..1
0..*

XXX
Attribute_1 : int
Attribute_2 : String
Attribute_3 : int

YYY
Attribute_4 : int
Attribute_5 : int
Attribute_6 : int

Attribute_1 is the primary identifier

New foreign key column introduced in YYY to match the primary key of XXX.

...to relational database

XXX
Attribute_1 INTEGER <pk> not null
Attribute_2 VARCHAR2(30) null
Attribute_3 INTEGER null

YYY
Attribute_4 INTEGER <pk> not null
Attribute_5 INTEGER <pk> not null
Attribute_1 INTEGER <fk> null
Attribute_6 INTEGER null

Attribute_1 = Attribute_1
0..*
From many-to-many association...

XXX
- Attribute_1 : int
- Attribute_2 : String
- Attribute_3 : int

YYY
- Attribute_4 : int
- Attribute_5 : int
- Attribute_6 : int

Many-to-many association

Association_1
- Attribute_1 : INTEGER <pk,fk1> not null
- Attribute_4 : INTEGER <pk,fk2> not null
- Attribute_5 : INTEGER <pk,fk2> not null

New table Association_1 introduced to maintain the many-to-many relationship between XXX and YYY. The primary key of Association_1 is a set of columns consisting of the primary key columns of XXX and YYY. The foreign keys of Association_1 refer to the primary keys of the corresponding primary tables.

...to relational database

XXX
- Attribute_1 INTEGER <pk> not null
- Attribute_2 VARCHAR2(30) null
- Attribute_3 INTEGER null

YYY
- Attribute_4 INTEGER <pk> not null
- Attribute_5 INTEGER <pk> not null
- Attribute_6 INTEGER null

Association_1
- Attribute_1 INTEGER <pk,fk1> not null
- Attribute_4 INTEGER <pk,fk2> not null
- Attribute_5 INTEGER <pk,fk2> not null
From one-to-one association...

XXX
- Attribute_1 : int
- Attribute_2 : String
- Attribute_3 : int

YYY
- Attribute_4 : int
- Attribute_5 : int
- Attribute_6 : int

One-to-one association

0..1

1..1

...to relational database

XXX
- Attribute_1 : INTEGER <pk> not null
- Attribute_4 : INTEGER <fk> null
- Attribute_5 : INTEGER <fk> null
- Attribute_2 : VARCHAR2(30) null
- Attribute_3 : INTEGER null

YYY
- Attribute_4 : INTEGER <pk> not null
- Attribute_5 : INTEGER <pk> not null
- Attribute_6 : INTEGER null

The mapping introduces foreign keys in XXX and YYY to permit both-directional "navigation" between tables. The foreign key in YYY does not accept nulls because of the 1..1 multiplicity from YYY to XXX. The foreign in XXX permits nulls because of the 0..1 multiplicity from XXX to YYY.
Recursive (self-referential) one-to-many association

0..1
Role A

XXX

Attribute_1 : int
Attribute_2 : String
Attribute_3 : int

0..*
Role B

...to relational database

Attribute_1 = XXX_Attribute_1

XXX

Attribute_1 INTEGER <pk> not null
XXX_Attribute_1 INTEGER <fk> null
Attribute_2 VARCHAR2(30) null
Attribute_3 INTEGER null

New foreign key column XXX_Attribute1 introduced to maintain a recursive one-to-many relationship to primary key values. There can be many records in XXX with a foreign key value equal to a primary key value in another record in XXX. The foreign key accepts nulls because of the 0..1 multiplicity of Role A.
From many-to-many recursive association...

Recursive (self-referential) many-to-many association

Role A

1..*

Attribute_1 : int
Attribute_2 : String
Attribute_3 : int

Role B

0..*

XXX

Any many-to-many association, whether recursive or defined on two classes, results in a "relationship table" with two referential integrity constraints.
From generalization...

XXX
Attribute_1 : int
Attribute_2 : String
Attribute_3 : int

Generalization

YYY
Attribute_4 : int
Attribute_5 : int
Attribute_6 : int

...to relational database

XXX
Attribute_1 INTEGER <pk> not null
Attribute_2 VARCHAR2(30) null
Attribute_3 INTEGER null

Attribute_1 = Attribute_1 0..*

YYY
Attribute_1 INTEGER <pk, fk> not null
Attribute_4 INTEGER not null
Attribute_5 INTEGER not null
Attribute_6 INTEGER null

YYY "inherits" only the primary key of XXX and uses it as its own primary key as well as the foreign key to XXX.
From conceptual class diagram for EM...

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>employee_id: String, first_name: String, family_name: String, login_name: String, employee_email: String</td>
<td>creator: 0..*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sender: 0..*</td>
</tr>
<tr>
<td>Contact</td>
<td>contact_id: String, organization: String, first_name: String, family_name: String, contact_email: String</td>
<td>message_id: NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message_subject: String, message_text: String</td>
</tr>
</tbody>
</table>

OutMessage

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_id: INTEGER</td>
<td>&lt;pk&gt; not null</td>
</tr>
<tr>
<td>contact_id: CHAR(5)</td>
<td>&lt;fk1&gt; not null</td>
</tr>
<tr>
<td>creator_emp_id: CHAR(4)</td>
<td>&lt;fk2&gt; not null</td>
</tr>
<tr>
<td>sender_emp_id: CHAR(4)</td>
<td>&lt;fk3&gt; null</td>
</tr>
<tr>
<td>message_subject: VARCHAR2(40)</td>
<td>not null</td>
</tr>
<tr>
<td>message_text: VARCHAR2(255)</td>
<td>not null</td>
</tr>
<tr>
<td>date_created: DATE</td>
<td>not null</td>
</tr>
<tr>
<td>date_emailed: DATE</td>
<td>not null</td>
</tr>
</tbody>
</table>

...to logical database schema for EM

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>employee_id: CHAR(4)</td>
<td>&lt;pk&gt; not null</td>
</tr>
<tr>
<td></td>
<td>first_name: VARCHAR2(20)</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>family_name: VARCHAR2(40)</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>login_name: VARCHAR2(40)</td>
<td>&lt;&gt; not null</td>
</tr>
<tr>
<td></td>
<td>employee_email: VARCHAR2(80)</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>login_UN: &lt;&gt;</td>
<td>not null</td>
</tr>
<tr>
<td>Contact</td>
<td>contact_id: CHAR(8)</td>
<td>&lt;pk&gt; not null</td>
</tr>
<tr>
<td></td>
<td>organization: VARCHAR(80)</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>first_name: VARCHAR2(20)</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>family_name: VARCHAR2(40)</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>contact_email: VARCHAR2(80)</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>contact_id: CHAR(5)</td>
<td>&lt;&gt; not null</td>
</tr>
<tr>
<td></td>
<td>creator_emp_id: CHAR(4)</td>
<td>&lt;&gt; not null</td>
</tr>
<tr>
<td></td>
<td>sender_emp_id: CHAR(4)</td>
<td>&lt;&gt; not null</td>
</tr>
<tr>
<td></td>
<td>message_subject: VARCHAR2(40)</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>message_text: VARCHAR2(255)</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>date_created: DATE</td>
<td>not null</td>
</tr>
<tr>
<td></td>
<td>date_emailed: DATE</td>
<td>not null</td>
</tr>
</tbody>
</table>
Summary

- **Database** is large, persistent, multi-user, shareable, recoverable, consistent, secure, extensible.

- The dominant database model for business information systems is the **relational model**.

- An important software engineering task is to **map** between transient objects in an application program and persistent records in a database.