Database Technology for E-commerce Applications

Three-tier Architecture for Web Applications

- **Presentation layer**
  - Human-computer interface

- **Processing layer**
  - Application processing: interprets client requests, sends commands to database server, passes back results to clients

- **Data layer**
  - Manages permanent data
Outline

• ER conceptual data model
• Relational data model and SQL
• Programming databases
• JDBC
• HSQL

Data Models

The primitives used to describe the structure of the information in the database

Conceptual and implementation models
Entity-Relational Model

- Proposed by P. Chen, 1976
- High-level conceptual data model for database design
  - no implementation details, easier to communicate with nontechnical users
  - more formal than natural language specifications
  - independent of particular implementation
- Conceptual design in ER model transformed into logical design in an implementation data model, e.g. relational data model

Entities and Attributes

- Entity: a ‘thing’ with physical or conceptual existence
- Attributes: properties describing an entity
  - Domain: set of valid values
- Entity type: a set of entities with the same attributes
- Key attribute(s): identify uniquely each entity in the set
Relationship Types

- A relationship type among entity types $E_1, E_2, \ldots, E_k$: set of associations among entities from these entity types
- Degree: the number of participating entity types
- Cardinality Ratios for binary relationships
  - 1:1
  - 1:M
  - M:N
- Can have attributes

Example ER-modelling
Relational Data Model

- Proposed by Dr. Edgar F. Codd, 1970
- Data presented as two-dimensional tables, relations
  - one or more named columns
  - rows (tuples)
  - crossing points of columns and rows contain data values of atomic types
- Operations apply to relations and produce new relations

\[
\begin{array}{cccc}
\text{column}_1 & \ldots & \text{column}_n \\
\text{<row 2>} & \quad & \text{<row n>}
\end{array}
\]

Translation from ER to Relational Model

<table>
<thead>
<tr>
<th>Articles</th>
<th>Suppliers</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtNr</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>ArtNr</td>
<td>SNr</td>
<td>Qty</td>
</tr>
<tr>
<td>FK</td>
<td>FK</td>
<td></td>
</tr>
</tbody>
</table>
Querying Relational Data

• Two formalisms
  – procedural relational algebra
  – declarative relational calculus

• SQL, Structured Query Language
  – Users specify what data to be retrieved
  – SQL query processing module translates queries into efficient execution plans specifying how data is retrieved
  – Increases efficiency of database programming
  – Provides data independence

SQL

Data Definition Language (DDL)
  – Define/re-define database structure
Data Manipulation Language (DML)
  – Updates
  – Queries
Additional facilities
  – Views
  – Security, authorization
  – Integrity constraints
  – Transaction constraints
Background

History
– SEQUEL (Structures English QUery Language)
– early 70’s, IBM Research
– SQL (ANSI 1986), SQL1 or SQL86
– SQL2 or SQL92
– SQL3 or SQL:1999
– SQL:2003
  • Core specification and optional specialized packages

Standard language for all commercial DBMS
– Each DBMS has features outside standard

DDL: Creating Tables

Articles

<table>
<thead>
<tr>
<th>ArtNr</th>
<th>Name</th>
<th>Description</th>
<th>Category</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
</table>

CREATE TABLE articles (  
  ArtNr INTEGER,  
  Name VARCHAR(64),  
  Description VARCHAR(256),  
  Category VARCHAR(64),  
  Quantity INTEGER,  
  Price DOUBLE,  
  PRIMARY KEY (ArtNr));
Basic query statement: select – from – where

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE $P$;

- $A_1, A_2, \ldots, A_n$ – list of attribute names to be retrieved
- $r_1, r_2, \ldots, r_m$ – List of tables required to process the query
- $P$ – Conditional expression identifying the tuples to be retrieved
  - AND, OR, NOT, $<$, $<=$, $=$, $>=$, $>$
- Result of the query is a table

The SQL SELECT clause

select * from articles;
- the star denotes all attributes

select Name, Price from articles;
- projection on the specified attributes

select distinct Category from articles;
- duplicate removal

select Name, Price, Price*0.25 as VAT from articles;
- arithmetic expressions
The SQL WHERE clause

Selects tuples fulfilling the predicate to be returned

```sql
select Name, Description, Price
from articles
where ArtNr=125;
```

```sql
select Name, Price
from articles
where Price>2.5 and Category = 'Coffee';
```

Special comparisons

Matching string patterns
- Use LIKE
- % for any number of arbitrary symbol
- _ for any symbol
  ```sql
  select * from articles
  where Description like '%Colombian%';
  ```

Approx math equality
- Use \( \text{abs}(x-x_j) < \varepsilon \):
  ```sql
  select * from articles
  where abs(Price-2.0) < 0.5;
  ```
- Use BETWEEN:
  ```sql
  select * from articles
  where Quantity between 10 and 20;
  ```
The SQL FROM clause

Given two database tables

<table>
<thead>
<tr>
<th>persons</th>
<th>cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>Audi</td>
</tr>
<tr>
<td>John</td>
<td>Audi</td>
</tr>
<tr>
<td>Mike</td>
<td>Audi</td>
</tr>
</tbody>
</table>

| Alex    | BMW  |
| John    | BMW  |
| Mike    | BMW  |
| Alex    | Mercedes |
| John    | Mercedes |
| Mike    | Mercedes |

select * from persons, cars;

, this SQL query generates all possible persons-cars combinations.

Join

Relational algebra notation: $R \bowtie_{C} S$

C – join condition
- C is on the form $A_R \theta A_S$
  - $\theta$ is one of {$=, <, >, \leq, \geq, \neq$}
  - Several terms can be connected as $C_1 \theta_1 C_2 \theta_2 \ldots C_K$

Special cases
- Equijoin: $\theta$ is $=$
- Natural join: All identically named attributes in relations R and S have matching values
SQL Join

For each article find the numbers of its suppliers:
```
select articles.ArtNr, articles.Name, supply.SNr
from articles, supply
where articles.ArtNr=supply.ArtNr;
```
```
select articles.ArtNr, articles.Name, supply.SNr
from articles JOIN supply on
articles.ArtNr=supply.ArtNr;
```

SQL Join more examples

Show the names and addresses of ‘big’ suppliers who make supplies with quantity > 1000
```
select Name, City, Address
from suppliers join supply on supply.SNr = suppliers.SNr
where Qty>1000;
```

Show the names and addresses of all suppliers of coffee
```
select suppliers.Name, suppliers.City, suppliers.Address
from articles join supply on articles.ArtNr = supply.ArtNr
join suppliers on supply.SNr = suppliers.SNr
where articles.Category='Coffee';
```
NULL values

Sometimes an attribute is
- Unknown  (date of birth unknown)
- Unavailable (refuses to list home phone #)
- Not applicable (last college degree)

Need to represent these cases in a DB!

Solution: NULL

Comparison of NULL values

- =, ≠, >, <, LIKE, ...
- won’t work. NULL is UNDEFINED!

SQL check for NULL
- IS NULL
- IS NOT NULL

JOIN operations
- Tuples with NULL values in the join columns
  ⇒ Not included in result
- Exception: OUTER JOIN
NULL Example

Find out articles without specified description

```sql
select ArtNr, Name
from articles
where Description is NULL;
```

Aggregate functions

- **Avg** – average value
- **Min** – minimum value
- **Max** – maximum value
- **Sum** – sum of values
- **Count** – number of values

Average price of all articles

```sql
select avg(Price)
from articles;
```
Aggregate functions – group by

Average price for each category of articles
   select Category, avg(Price)
   from articles
   group by Category;

For each supplier show his name and how many different articles he supplies
   select s.Name, count(distinct ArtNr)
   from supplier as s, supply as sup
   where s.SNr = sup.SNr
   group by s.Name;

Aggregate functions – HAVING

Show the average price and number of articles for each category with at least 3 articles
   select Category, avg(Price), count(*)
   from articles
   group by Category
   having count(*)>3;

Show the categories of articles for which the minimum price is above a threshold 10.0
   select Category, min(Price)
   from articles
   group by Category
   having min(Price)>10.0;
SQL Select Summary

Clauses:

- **SELECT** <attribute list>
- **FROM** <table list>
- **[WHERE** <condition>]
- **[GROUP BY** <grouping attributes>
- **[HAVING** <group condition>]
- **[ORDER BY** <attribute list>]

Views

- They are views of the underlying tables, not stored tables
- Defined through queries, i.e. stored queries
- Reflect changes in the underlying tables
- Used for frequently posed queries and security

Show the average price and number of articles for each category with at least 3 articles

```sql
create view NamedSupply(Aname,Sname,Qty,Date)
as select articles.Name,suppliers.Name, supply.Qty, supply.Date
from articles join supply on articles.ArtNr = supply.ArtNr
join suppliers on supply.SNr = suppliers.SNr;
```

```sql
select * from NamedSupply;
```
Modifying the Database - INSERT

**INSERT INTO** `<table>` [(column1, column2, …)]
**VALUES** (...);

insert into articles
values(121,'Colombian','good','Coffee',100, 2.25);

insert into articles (ArtNr, Name, Category, Price)
values(125,'Segafredo','Coffee', 6.18);

What is the value of Description and Category?

---

Modifying the Database - UPDATE

**UPDATE** `<table>`
**SET** `<column1>` = `<value1>` [, …]
**WHERE** `<condition>`;

Single row: increase the quantity of article numbered 121 with 10:
UPDATE articles
SET Quantity = Quantity - 10
WHERE ArtNr = 121;

Multiple rows: increase the price of all black tea by 5%:
UPDATE articles
SET Price = Price*1.05
WHERE Category = 'Black tea';
Modifying the Database - DELETE

DELETE FROM <table>
WHERE <condition>;

Single row: delete supplier numbered 10:
delete suppliers
where SNr = 10;

Multiple rows: delete all supplies before 1.1.2006:
delete supply
where Date<'1-JAN-2006';

More about SQL

Tutorials:
http://www.1keydata.com/sql/sql.html
http://www.sqlcourse.com/
http://www.sqlcourse2com/

http://www.firstsql.com/tutor.htm (SQL92)
http://hsqldb.sourceforge.net/web/hsqlDocsFrame.html SQL in HSQLDB
Accessing Databases

- Interactive interfaces
  Convenient for schema and constraint creation, ad-hoc queries
  - Textual
  - Graphical

- Application Programs or Database Applications
  Majority of database interactions
  - Embedded SQL
  - Application Programming Interface, SQL Client Interface
  - Database programming language (Oracle’s PL/SQL)

Embedded SQL

- Database statements embedded in host programming language
- Identified by special prefix
- Preprocessor or precompiler replaces database statements by function calls
SQL Client Interfaces

• Library of functions for database calls available to the host language
• Actual database statements are parameters in function calls
• Types:
  1. Vendor-specific client interfaces
  2. Standard client interfaces for DBMS-independent application
     • ODBC, SQL/CLI
     • JDBC

Impedance Mismatch

Difference between database model and programming language model

• Differences in data types require a binding for each host programming language
• SQL query result is a table, or set of tuples, and require binding to programming language data structure. Cursor iterates over tuples in a query result.
Typical Interaction Sequence

- Establish a connection with a database specifying database URL, login account and password
- Submit queries, updates and other SQL statements
- Process the results
- Close the connection

JDBC Architecture
import java.sql.*;

D. Ince, "Developing distributed and e-commerce applications", Figure 5.2, p. 113

**Establishing Connections**

1. **Load JDBC driver**
   
   ```java
   Class.forName("org.hsqldb.jdbcDriver");
   ```

2. **Make connection**

   ```java
   Connection con= DriverManager.getConnection( 
   url,   // Database URL
   "myLogin",   // login name to the DBMS
   "myPassword");   // password for the DBMS
   ```

   ```java
   Connection con= DriverManager.getConnection( 
   "jdbc:hsqldb:file:TestDB", "sa", """);
   ```
Send SQL Statements

1. Create JDBC Statement object
   Statement stmt = con.createStatement();

2. Execute SQL statements
   stmt.executeUpdate(<Update String>)
   for DDL, DML- updates statements
   stmt.executeQuery(<Query String>)
   for queries
   stmt.execute()
   general method for multiple result sets

3. Close statement
   stmt.close();

Creating Tables

SQL DDL statement
Statement stmt = con.createStatement();
stmt.executeUpdate("CREATE TABLE articles ( 
   ArtNr INTEGER, 
   Name VARCHAR(64), 
   Description VARCHAR(256), 
   Category VARCHAR(64), 
   Quantity INTEGER, 
   Price DOUBLE, 
   UNIQUE(ArtNr))");
Entering Data

Statement stmt = con.createStatement();

stmt.executeUpdate("INSERT INTO articles " +
"VALUES(1, 'Colombian', 'good', 'Coffee', 100, 2.25)");

SQL Queries

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT ArtNr, Name, Price " +
"FROM Articles");
Retrieving Results

next method moves a cursor to the next row

getxxx methods retrieve a column value given column name or number in the result set. xxx is Java type of retrieved value

Example:

```java
while(rs.next()){
    int nr = rs.getInt("ArtNr");
    String nm = rs.getString("Name");
    double pr = rs.getDouble("Price");
    System.out.println(nr + " " + nm + " " + pr);
}
```

Updating Data

```java
Statement stmt = con.createStatement();
stmt.executeUpdate("UPDATE articles  SET Quantity = Quantity - 10 WHERE ArtNr = 1");

public void updateArticles(int nr, int qty) throws SQLException {
    Statement stmt = con.createStatement();
    String updString = "UPDATE articles SET Quantity = Quantity - " + qty + " WHERE ArtNr = " + nr;
    stmt.executeUpdate(updString);
    stmt.close();
}
```
**Prepared Statements**

Sent to DBMS and precompiled
Typically parameterized SQL statements

```java
String query = "SELECT ArtNr, Name, Price " +
               "FROM Articles WHERE Category = ?";
PreparedStatement pstmt = con.prepareStatement(query);
String cname;
while(...){
    // read category into cname
    pstmt.setString(1, cname);
    ResultSet rs = pstmt.executeQuery();
    while(rs.next()){*
        // print results
    }
}
```

**Using Metadata**

ResultSetMetaData:
- `getColumnCount()` - number of columns in result set
- `getColumnName(i)` - name of column #i
- `getColumnTypeName(i)` - type of column #i in DBMS
- `getTableName(i)`

```java
ResultSetMetaData meta = rs.getMetaData();
int colmax = meta.getColumnCount();
for(int k = 1; k <= colmax; k++) {
    System.out.println(meta.getColumnName(k) +
                        " of type " + meta.getColumnTypeName(k));
}
Retrieving Query Results Using Metadata

```java
ResultSet rs = stmt.executeQuery(" ...");
ResultSetMetaData meta = rs.getMetaData();
int colmax = meta.getColumnCount();
int i;
String s;
while (rs.next()) {
    for (i = 1; i <= colmax; ++i) {
        s = rs.getString(i);
        System.out.print(s + " ");
    }
    System.out.println;
}
```

HSQLDB

- Relational database engine in Java
- Supports subset of SQL:92 + enhancements from SQL:99 and 2003
- Integrated with Open Office
- Components:
  - RDBMS
  - JDBC driver
  - Database Manager
  - Query Tool
HSQldb cont.

- Database engine runs in both
  - Server modes
  - In-process (embedded, standalone) mode

- Database files
  - mydb.properties (settings)
  - mydb.script (schema and tables)
  - mydb.log (logs changes)
  - mydb.data (cashed tables)
  - mydb.backup

- Tables
  - MEMORY held entirely in memory, recreated from .script when the database is opened
  - CACHED partially held in memory, allow processing of large tables

HSQl Database Manager

```
java -cp ../lib/hsqldb.jar
    org.hsqldb.util.DatabaseManager
```
SQL and Java Data Types

<table>
<thead>
<tr>
<th>SQL Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
</tr>
<tr>
<td>CHAR</td>
<td>String</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Time</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>REAL</td>
<td>double</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
</tbody>
</table>
Using HSQLDB

Use in-process mode
Create the DB with a Java program
Set the permissions of the DB directory and files to be accessible for writing for Tomcat