Sorting the Results of a Query

- **ORDER BY** *column* [ ASC | DESC ] [, ...]

  ```sql
  SELECT  S.rating, S.sname, S.age
  FROM  Sailors S, Boats B, Reserves R
  WHERE  S.sid=R.sid
  AND R.bid=B.bid AND B.color='red'
  ORDER BY  S.rating, S.sname;
  ```

- **Can order by any column in SELECT list, including expressions or aggs:**

  ```sql
  SELECT S.sid, COUNT (*) AS redrescnt
  FROM  Sailors S, Boats B, Reserves R
  WHERE  S.sid=R.sid
  AND R.bid=B.bid AND B.color='red'
  GROUP BY S.sid
  ORDER BY  redrescnt DESC;
  ```
Views (repeat from last class)

CREATE VIEW view_name
AS select_statement

Makes development simpler
Often used for security
Not instantiated - makes updates tricky

CREATE VIEW Reds
AS SELECT B.bid, COUNT (*) AS scount
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid
Views Instead of Relations in Queries

CREATE VIEW Reds
AS SELECT  B.bid,  COUNT (*) AS scount
  FROM Boats B, Reserves R
  WHERE  R.bid=B.bid AND  B.color='red'
  GROUP BY  B.bid

SELECT  bname, scount
  FROM  Reds R, Boats B
  WHERE  R.bid=B.bid
  AND scount < 10
Discretionary Access Control

GRANT privileges ON object TO users
[WITH GRANT OPTION]

- Object can be a Table or a View
- Privileges can be:
  - Select
  - Insert
  - Delete
  - References (cols) – allow to create a foreign key that references the specified column(s)
  - All
- Can later be REVOKE
- Users can be single users or groups
- See Chapter 17 for more details.
Two more important topics

- Constraints

- SQL embedded in other languages
Integrity Constraints (Review)

• An IC describes conditions that every *legal instance* of a relation must satisfy.
  – Inserts/deletes/updates that violate IC’s are disallowed.
  – Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)

• **Types of IC’s:** Domain constraints, primary key constraints, foreign key constraints, general constraints.
  – *Domain constraints:* Field values must be of right type. Always enforced.
  – *Primary key and foreign key constraints:* you know them.
General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Checked on insert or update.
- Constraints can be named.

```
CREATE TABLE Sailors
    ( sid INTEGER,
      sname CHAR(10),
      rating INTEGER,
      age REAL,
      PRIMARY KEY (sid),
      CHECK ( rating >= 1
              AND rating <= 10 )
    )

CREATE TABLE Reserves
    ( sname CHAR(10),
      bid INTEGER,
      day DATE,
      PRIMARY KEY (bid,day),
      CONSTRAINT noInterlakeRes
      CHECK (`Interlake' <>
            ( SELECT B.bname
              FROM Boats B
              WHERE B.bid=bid))
    )
```
Constraints Over Multiple Relations

CREATE TABLE Sailors
( sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100 )

- Awkward and wrong!
- Only checks sailors!
- Only required to hold if the associated table is non-empty.
- ASSERTION is the right solution; not associated with either table.
- Unfortunately, not supported in many DBMS.
- Triggers are another solution.

CREATE ASSERTION smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
Triggers (Active database)

- **Trigger**: A procedure that starts automatically if specified changes occur to the DBMS

- Analog to a "daemon" that monitors a database for certain events to occur

- Three parts:
  - **Event** (activates the trigger)
  - **Condition** (tests whether the triggers should run) [Optional]
  - **Action** (what happens if the trigger runs)

- **Semantics**:
  - When event occurs, and condition is satisfied, the action is performed.
Triggers – Event, Condition, Action

- **Events could be:**
  
  \[
  \text{BEFORE|AFTER\ INSERT|UPDATE|DELETE\ ON<\table\ Name>}
  \]

  e.g.: \text{BEFORE\ INSERT\ ON\ Professor}

- **Condition is SQL expression or even an SQL query (query with non-empty result means TRUE)**

- **Action can be many different choices:**
  
  – SQL statements, body of PSM, and even DDL and transaction-oriented statements like “commit”.
Example Trigger

Assume our DB has a relation schema:

\textbf{Professor (pNum, pName, salary)}

We want to write a trigger that:

\textbf{Ensures that any new professor inserted has salary $\geq 60000$}
Example Trigger

CREATE TRIGGER minSalary BEFORE INSERT ON Professor

for what context ?

BEGIN

    check for violation here ?

END;
Example Trigger

CREATE TRIGGER minSalary BEFORE INSERT ON Professor

FOR EACH ROW

BEGIN

Violation of Minimum Professor Salary?

END;
Example Trigger

CREATE TRIGGER minSalary BEFORE INSERT ON Professor

    FOR EACH ROW

BEGIN

    IF (:new.salary < 60000)
       THEN RAISE_APPLICATION_ERROR (-20004,
       'Violation of Minimum Professor Salary');

    END IF;

END;

END;
Example trigger

CREATE TRIGGER minSalary BEFORE INSERT ON Professor
   FOR EACH ROW

DECLARE temp int; -- dummy variable not needed

BEGIN
   IF (:new.salary < 60000)
      THEN RAISE_APPLICATION_ERROR (-20004,
         'Violation of Minimum Professor Salary');
   END IF;

   temp := 10; -- to illustrate declared variables

END;

.
Details of Trigger Example

• **BEFORE INSERT ON Professor**
  – This trigger is checked before the tuple is inserted
• **FOR EACH ROW**
  – specifies that trigger is performed for each row inserted
• **:new**
  – refers to the new tuple inserted
• **If (:new.salary < 60000)**
  – then an application error is raised and hence the row is not inserted; otherwise the row is inserted.
• **Use error code: -20004;**
  – this is in the valid range
Example Trigger Using Condition

CREATE TRIGGER minSalary BEFORE INSERT ON Professor
FOR EACH ROW
WHEN (new.salary < 60000)
BEGIN
  RAISE_APPLICATION_ERROR (-20004,
    'Violation of Minimum Professor Salary');
END;

- Conditions can refer to old/new values of tuples modified by the statement activating the trigger.
Triggers: REFERENCING

CREATE TRIGGER minSalary BEFORE INSERT ON Professor

REFERENCING NEW as newTuple

FOR EACH ROW

WHEN (newTuple.salary < 60000)

BEGIN

RAISE_APPLICATION_ERROR (-20004,
    'Violation of Minimum Professor Salary');

END;
Example Trigger

CREATE TRIGGER minSalary
    BEFORE UPDATE ON Professor
REFERENCING OLD AS oldTuple NEW as newTuple
FOR EACH ROW
WHEN (newTuple.salary < oldTuple.salary)
BEGIN
    RAISE_APPLICATION_ERROR (-20004, ‘Salary Decreasing !!’);
END;

• Ensure that salary does not decrease
Another Trigger Example (SQL:99)

CREATE TRIGGER youngSailorUpdate
AFTER INSERT ON SAILORS
REFERENCING NEW TABLE AS NewSailors
FOR EACH STATEMENT

INSERT
INTO YoungSailors(sid, name, age, rating)
SELECT sid, name, age, rating
FROM NewSailors N
WHERE N.age <= 18
Row vs Statement Level Trigger

- **Row** level: activated once per modified tuple
- **Statement** level: activate once per SQL statement

- **Row** level triggers can access new data, statement level triggers cannot always do that (depends on DBMS).

- **Statement** level triggers will be more efficient if we do not need to make row-specific decisions
Row vs Statement Level Trigger

- Example: Consider a relation schema

  `Account (num, amount)`

  where we will allow creation of new accounts only during normal business hours.
Example: Statement level trigger

CREATE TRIGGER MYTRIG1
BEFORE INSERT ON Account
FOR EACH STATEMENT        --- is default
BEGIN
    IF (TO_CHAR(SYSDATE,'dy') IN ('sat','sun'))
       OR
      (TO_CHAR(SYSDATE,'hh24:mi') NOT BETWEEN '08:00'
       AND '17:00')
    THEN
        RAISE_APPLICATION_ERROR(-20500,'Cannot create new account now !!');
    END IF;
END;
When to use BEFORE/AFTER

- Based on efficiency considerations or semantics.

- Suppose we perform statement-level after insert, then all the rows are inserted first, then if the condition fails, and all the inserted rows must be “rolled back”

- Not very efficient !!
Combining multiple events into one trigger

CREATE TRIGGER salaryRestrictions
AFTER INSERT OR UPDATE ON Professor
FOR EACH ROW
BEGIN
IF (INSERTING AND :new.salary < 60000) THEN
    RAISE_APPLICATION_ERROR (-20004, 'below min salary'); END IF;
IF (UPDATING AND :new.salary < :old.salary) THEN RAISE_APPLICATION_ERROR (-20004, 'Salary Decreasing !!'); END IF;
END;
Summary: Trigger Syntax

CREATE TRIGGER <triggerName>
BEFORE | AFTER INSERT | DELETE | UPDATE
[OF <columnList>] ON <tableName> | <viewName>
[REFERENCING [OLD AS <oldName>] [NEW AS <newName>]]
[FOR EACH ROW] (default is “FOR EACH STATEMENT”)
[WHEN (<condition>)]
<PSM body> ;
Some Points about Triggers

• Check the system tables:
  - user_triggers
  - user_trigger_cols
  - user_errors

• ORA-04091: mutating relation problem
  - In a row level trigger, you cannot have the body refer to the table specified in the event

• Also instead of triggers can be specified on views
To Show Compilation Errors

SELECT line, position, text
FROM user_errors
WHERE name = 'MY_TRIGGER'
    AND TYPE = 'TRIGGER'

• In SQL*Plus, you can also use the following shortcut:

SQL> SHOW ERRORS TRIGGER MY_TRIGGER
Constraints versus Triggers

- **Constraints** are useful for database consistency
  - Use IC when sufficient
  - More opportunity for optimization
  - Not restricted into insert/delete/update

- **Triggers** are flexible and powerful
  - Alerters
  - Event logging for auditing
  - Security enforcement
  - Analysis of table accesses (statistics)
  - Workflow and business intelligence ...

- **But can be hard to understand ……**
  - Several triggers (Arbitrary order → unpredictable !?)
  - Chain triggers (When to stop ?)
  - Recursive triggers (Termination?)
Writing Applications with SQL

- **SQL is not a general purpose programming language.**
  - Tailored for data retrieval and manipulation
  - Relatively easy to optimize and parallelize
  - Can’t write entire apps in SQL alone

**Options:**

- Make the query language “turing complete”
  - Avoids the “impedance mismatch”
    - but, loses advantages of relational lang simplicity
- Allow SQL to be embedded in regular programming languages.

Q: What needs to be solved to make the latter approach work?
Embedded SQL

• **DBMS vendors usually provide “host language bindings”**
  – E.g. for C or COBOL
  – Allow SQL statements to be called from within a program
  – Typically you preprocess your programs
  – Preprocessor generates calls to a proprietary DB connectivity library

• **General pattern**
  – One call to `connect` to the right database (login, etc.)
  – SQL statements can refer to host variables from the language

• **Typically vendor-specific**
  – We won’t look at any in detail, we’ll look at standard stuff

• **Problem**
  – SQL relations are (multi-)sets, no *a priori* bound on the number of records. No such data structure in C.
  – SQL supports a mechanism called a `cursor` to handle this.
Just to give you a flavor

EXEC SQL SELECT S.sname, S.age
    INTO :c_sname,:c_age
FROM Sailors S
WHERE S.sid = :c_sid
Cursors

- Can declare a cursor on a relation or query
- Can **open** a cursor
- Can repeatedly **fetch** a tuple (moving the cursor)
- Special return value when all tuples have been retrieved.
- **ORDER BY** allows control over the order in which tuples are returned.
  - Fields in ORDER BY clause must also appear in SELECT clause.
- **Can also modify/delete tuple pointed to by a cursor**
  - A “non-relational” way to get a handle to a particular tuple
- **There’s an Embedded SQL syntax for cursors**
  - DECLARE <cursorname> CURSOR FOR <select stmt>
  - FETCH FROM <cursorname> INTO <variable names>
  - But we’ll use JDBC instead

34
Database APIs: alternative to embedding

- Rather than modify compiler, add a library with database calls (API)
  - special procedures/objects
  - passes SQL strings from language, presents result sets in a language-friendly way
  - ODBC a C/C++ standard started on Windows
  - JDBC a Java equivalent
  - Most scripting languages have similar things
    - E.g. For Perl there is DBI, “oraPerl”, other packages

- Mostly DBMS-neutral
  - at least try to hide distinctions across different DBMSs
A lookup service maps “data source names” (“DSNs”) to drivers
  - Typically handled by OS
Based on the DSN used, a “driver” is linked into the app at runtime
The driver traps calls, translates them into DBMS-specific code
Database can be across a network
ODBC is standard, so the same program can be used (in theory) to access multiple database systems
Data source may not even be an SQL database!
Various vendors provide drivers
- MS bundles a bunch into Windows
- Vendors like DataDirect and OpenLink sell drivers for multiple OSes

Drivers for various data sources
- Relational DBMSs (Oracle, DB2, SQL Server, Informix, etc.)
- “Desktop” DBMSs (Access, Dbase, Paradox, FoxPro, etc.)
- Spreadsheets (MS Excel, Lotus 1-2-3, etc.)
- Delimited text files (.CSV, .TXT, etc.)

You can use JDBC/ODBC clients over many data sources
- E.g. MS Query comes with many versions of MS Office (msqry32.exe)

Can write your own Java or C++ programs against xDBC
JDBC

• Part of Java, very easy to use
• Java comes with a JDBC-to-ODBC bridge
  – So JDBC code can talk to any ODBC data source
  – E.g. look in your Windows Control Panel for ODBC drivers!
• JDBC tutorial online
JDBC Basics: Connections

• A **Connection** is an object representing a login to a database

```java
// GET CONNECTION
Connection con;
try {
    con = DriverManager.getConnection(
        "jdbc:odbc:sailorsDB",
        userName, password);
} catch (Exception e) { System.out.println(e); }
```

• **Eventually you close the connection**

```java
// CLOSE CONNECTION
try { con.close(); }
catch (Exception e) { System.out.println(e); }
```
JDBC Basics: Statements

• You need a Statement object for each SQL statement

    // CREATE STATEMENT
    Statement stmt;
    try {
        stmt = con.createStatement();
    } catch (Exception e){
        System.out.println(e);
    }

    Soon we’ll say stmt.executeQuery("select ...");
CreateStatement cursor behavior

- **Two optional args to createStatement:**
  - `createStatement(ResultSet.<TYPE>, ResultSet.<CONCUR>)`
  - Corresponds to SQL cursor features

- **<TYPE> is one of**
  - `TYPE_FORWARD_ONLY`: can’t move cursor backward
  - `TYPE_SCROLL_INSENSITIVE`: can move backward, but doesn’t show results of any updates
  - `TYPE_SCROLL_SENSITIVE`: can move backward, will show updates from this statement

- **<CONCUR> is one of**
  - `CONCUR_READ_ONLY`: this statement doesn’t allow updates
  - `CONCUR_UPDATABLE`: this statement allows updates

- **Defaults:**
  - `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY`
A ResultSet object serves as a cursor for the statement's results (stmt.executeQuery() 

```java
// EXECUTE QUERY
ResultSet results;
try {
    results = stmt.executeQuery("select * from Sailors");
} catch (Exception e){
    System.out.println(e);
}
```

• Obvious handy methods:
  – results.next() advances cursor to next tuple
    • Returns “false” when the cursor slides off the table (beginning or end)
  – “scrollable” cursors:
    • results.previous(), results.relative(int), results.absolute(int), results.first(), results.last(), results.beforeFirst(), results.afterLast()
ResultSet Metadata

- Can find out stuff about the ResultSet schema via `ResultSetMetaData`

```java
ResultSetMetaData rsmd = results.getMetaData();
int numCols = rsmd.getColumnCount();
int i, rowcount = 0;

// get column header info
for (i=1; i <= numCols; i++){
    if (i > 1) buf.append(",");
    buf.append(rsmd.getColumnLabel(i));
}
buf.append("\n");
```

- Other `ResultSetMetaData` methods:
  - `getColumnType(i)`, `isNullable(i)`, etc.
Getting Values in Current of Cursor

- **getString**
  
  ```java
  // break it off at 100 rows max
  while (results.next() && rowcount < 100){
      // Loop through each column, getting the
      // column data and displaying

      for (i=1; i <= numCols; i++) {
          if (i > 1) buf.append(",");
          buf.append(results.getString(i));
      }
      buf.append("\n");
      rowcount++;
  }
  ```

- **Similarly, getFloat, getInt, etc.**
Updating Current of Cursor

- **Update fields in current of cursor:**
  
  ```java
  result.next();
  result.updateInt("Rating", 10);
  ```

- **Also updateString, updateFloat, etc.**

- **Or can always submit a full SQL UPDATE statement**
  
  - Via `executeQuery()`

- **The original statement must have been CONCUR_UPDATABLE in either case!**
Cleaning up Neatly

```java
try {
    // CLOSE RESULT SET
    results.close();
    // CLOSE STATEMENT
    stmt.close();
    // CLOSE CONNECTION
    con.close();
} catch (Exception e) {
    System.out.println(e);
}
```
Putting it Together (w/o try/catch)

```java
Connection con =
    DriverManager.getConnection("jdbc:odbc:weblog",userName,password);
Statement stmt = con.createStatement();
ResultSet results =
    stmt.executeQuery("select * from Sailors");
ResultSetMetaData rsmd = results.getMetaData();
int numCols = rsmd.getColumnCount(), i;
StringBuffer buf = new StringBuffer();

while (results.next() && rowcount < 100){
    for (i=1; i <= numCols; i++) {
        if (i > 1) buf.append(","鞏);  
        buf.append(results.getString(i));
    }
    buf.append("\n");
}
results.close(); stmt.close(); con.close();
```
Similar deal for web scripting langs

- **Common scenario today is to have a web client**
  - A web form issues a query to the DB
  - Results formatted as HTML

- **Many web scripting languages used**
  - jsp, asp, PHP, etc.
  - we’ll use JSP in our class
  - most of these are similar, look a lot like jdbc with HTML mixed in
<?php
$conn = pg_pconnect("dbname=cowbook user=jmh\password=secret");
if (!$conn) {
    echo "An error occurred.";
    exit;
}
$result = pg_query($conn, "SELECT * FROM Sailors");
if (!$result) {
    echo "An error occurred.";
    exit;
}
$num = pg_num_rows($result);
for ($i=0; $i < $num; $i++) {
    $r = pg_fetch_row($result, $i);
    for ($j=0; $j < count($r); $j++) {
        echo "$r[$j] ";
    }
    echo "<BR>";
}
?>
API Summary

APIs are needed to interface DBMSs to programming languages

- Embedded SQL uses “native drivers” and is usually faster but less standard

- ODBC (used to be Microsoft-specific) for C/C++.

- JDBC the standard for Java

- Scripting languages (PHP, Perl, JSP) are becoming the preferred technique for web-based systems.