Advanced SQL

- Accessing SQL From a Programming Language
  - Dynamic SQL: JDBC and ODBC
  - Embedded SQL
  - PHP Overview
- Accessing metadata
- Text Operations
- Functions and Procedural Constructs
- Triggers
- Advanced Aggregation Features
- OLAP

JDBC and ODBC

- API (application programming interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
- JDBC (Java Database Connectivity) works with Java

JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a "statement" object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();
            … Do Actual Work ….
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

JDBC Code (Cont.)

- Update to database

```
try {
    stmt.executeUpdate(
        "insert into instructor values('77987', 'Kim', 'Physics',
98000)");
} catch (SQLException sqle)
{
   System.out.println("Could not insert tuple. " + sqle);
}
```

- Execute query and fetch and print results

```
    ResultSet rset = stmt.executeQuery(
                        "select dept_name, avg (salary)
                         from instructor
                         group by dept_name");
    while (rset.next()) {
        System.out.println(rset.getString("dept_name") + " " +
                               rset.getFloat(2));
    }
```

Prepared Statement

- ```
  PreparedStatement pStmt = conn.prepareStatement(
                                  "insert into instructor values(?,?,?,?)");
  pStmt.setString(1, "88877");
  pStmt.setString(2, "Perry");
  pStmt.setString(3, "Finance");
  pStmt.setInt(4, 125000);
  pStmt.executeUpdate();
  pStmt.setString(1, "88878");
  pStmt.executeUpdate();
  ```
- WARNING: always use prepared statements when taking an input from the user and adding it to a query
  - NEVER create a query by concatenating strings
  - "insert into instructor values(’ " + ID + " ’ , ’ " + name + " ’ , " + " ’ + dept name + " ’ , " ’ balance + ")“
  - What if name is “D’ Souza”?
  ```

ODBC

- Open DataBase Connectivity(ODBC) standard
  - standard for application program to communicate with a database server.
  - application program interface (API) to
    - open a connection with a database,
    - send queries and updates,
    - get back results.
- Applications such as GUI, spreadsheets, etc. can use ODBC

Embedded SQL

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, Java, and Cobol.
- A language to which SQL queries are embedded is referred to as a **host language**, and the SQL structures permitted in the host language comprise *embedded* SQL.
- **EXEC SQL** statement is used to identify embedded SQL request to the preprocessor

  EXEC SQL <embedded SQL statement > END_EXEC

  Note: this varies by language (for example, the Java embedding uses # SQL { …. };)

Example Query

- From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable credit_amount.
- Specify the query in SQL and declare a *cursor* for it

EXEC SQL

declare *c* **cursor for**
select *ID, name*
from *student*
where tot_cred *> :credit_amount*

END_EXEC

PHP Overview

- Server side scripting language
- Designed for web development but also general purpose programming language
- PHP code can be mixed with HTML
- Client runs browser, which sends HTTP requests, receives HTTP responses, and renders the HTML document from the response
- Web server (e.g. Apache) calls PHP script that requested url points to, and incorporates output into the response
  - Script is html mixed with executable code fragments
- Optionally, script connects to DBMS and uses query results to produce its output
  - Works well with MySQL DBMS for most applications (recommended for project)

Executing SQL from PHP

- Connect to server
  - mysql_connect
- Select the database
  - mysql_select_db
- Run query
- Retrieve row of results
  - mysql_fetch_array
- Retrieve attributes
  - Foreach
- PHP interactive tutorial
  - http://www.w3schools.com/php/default.asp

Typical Application

- Login page:
  - Collect credentials and pass them to setup page via POST
- Setup page:
  - Check credentials
  - Initialize session and session variables
  - Redirect to welcome page
- Application pages
  - Call session_start(), authenticate the session, and use/ update session variables, as needed
- Logout page
  - Calls session_destroy()
  - Redirects to “goodbye” page

Some Security Issues

- Detailed treatment is beyond the scope of this class, but you should be aware that issues exist.
- HTTP sends data in the clear. For real applications that handle sensitive data, should use HTTPS
  - authenticate server
  - encrypt data sent over network via SSL
- Session hijacking
  - Adversary who discovers session ID can take over a session
  - Checking IP address of each request helps mitigate this threat, but doesn't eliminate it

Security Issues (Cont.)

- SQL injection
  - Malicious user enters input that results in execution of an SQL statement other than the intended one, e.g.
    - Select * from T where name= 'joe' or '1' = '1' ;

    Instead of
    - Select * from T where name= 'joe' ;
- Cross-site scripting
  - Malicious user gives input that hides script in content that others will download
- Application code should check that input is of the expected form and or "clean" the data,

Metadata

- The dictionary or catalog stores information about the database itself.
- This is data about data or ‘metadata’.
- Almost every aspect of a DBMS uses this metadata
- The dictionary holds:
  - Description of database objects (tables, users, rules, views, indexes,..)
  - Information about who is using which data (locks)
  - Schemas and mapping
  - The dictionary itself

Query Exercises in MySQL

- Based on the following schema:

ACTOR(<u>AID</u>, FIRSTNAME, LASTNAME)

MOVIE(<u>MID</u>, MNAME, BUDGET, GROSS)

ACTED_IN(<u>AID, MID</u>, STARRING, WAGE)

1. List all tables that exist in the test schema

```
SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'TEST'
LIMIT 0 , 30
```

Result

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION	ROW_FORMAT	TABLE_ROWS
def	test	acted_in	BASE TABLE	InnoDB	10	Compact	0
def	test	actor	BASE TABLE	InnoDB	10	Compact	4
def	test	movie	BASE TABLE	InnoDB	10	Compact	0

2. List all tables that have more than 3 columns.

```
SELECT  TABLE_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'TEST’
GROUP BY TABLE_NAME
HAVING COUNT(*) > 3
```

Result

←T→				TABLE_NAME
Edit	Inline Edit	Copy	Delete	acted_in
Edit	Inline Edit	Copy	Delete	movie

3. List the first name and last name of all actors whose last name is equal to the name of an attribute in MOVIE

```
SELECT FIRSTNAME, LASTNAME
FROM TEST.ACTOR, INFORMATION_SCHEMA.COLUMNS I
WHERE TABLE_SCHEMA = 'TEST'
AND I.TABLE_NAME = 'MOVIE'
AND LASTNAME = I.COLUMN_NAME
```

Table Actor

←T→					aid	firstname	lastname
☐	Edit	Inline Edit	Copy	Delete	1	DAVID	BUDGET
☐	Edit	Inline Edit	Copy	Delete	2	TOM	GROSS
☐	Edit	Inline Edit	Copy	Delete	3	JOHN	SMITH
☐	Edit	Inline Edit	Copy	Delete	4	TOM	MIKE

Result

←T→					FIRSTNAME	LASTNAME
☐	Edit	Inline Edit	Copy	Delete	DAVID	BUDGET
☐	Edit	Inline Edit	Copy	Delete	TOM	GROSS

More Exercises

- 1. List names of the tables that contain a column called ‘AID’.
- 2. List the tables that have the most columns.
- 3. Or list the tables with more rows than columns.

Text operations in Oracle

- like: pattern matching, char-oriented
- contains: index-based, word-oriented

LIKE

- … Where Person.Name LIKE '_essi%'
- Matches: 'Bessie', 'Jessie', 'Jessica'

  ... but not 'Dressica'

- _ (underscore) means one arbitrary CHAR
- % means an arbitrary STRING

- Can choose to normalize to upper or lower case
- Where P.Description LIKE '%pen%' matches 'Appendix'
- Usually used with varchar, on small strings

CONTAINS

- Where CONTAINS (P.Description, ‘T30’, 1) > 0
- Word-oriented
- Used for clob data type, on larger pieces of text
- Much more powerful (includes ranking)
- Oracle: was part of Oracle Text (formerly Intermedia)
- Included in Oracle 9i and up
- Faster than like in many cases because uses index
- Must build index first (Create index …)

Updates Through Cursors

- Cursor = a pointer to a row in a results set
- Can update tuples fetched by cursor by declaring that the cursor is for update

  declare *c* **cursor for**
  select *
  from *instructor*
  where *dept_name* = 'Music'
  for update

- To update tuple at the current location of cursor *c*

  update *instructor*
  set *salary* = *salary* + 100
  where current of *c*

Procedural Extensions and Stored Procedures

- SQL provides a **module** language
  - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
- Stored Procedures
  - Can store procedures in the database
  - then execute them using the **call** statement
  - permit external applications to operate on the database without knowing about internal details

Functions and Procedures

- SQL:1999 supports functions and procedures
  - Functions/procedures can be written in SQL itself, or in an external programming language.
  - Functions are particularly useful with specialized data types such as images and geometric objects.
    - Example: functions to check if polygons overlap, or to compare images for similarity.
  - Some database systems support **table-valued functions**, which can return a relation as a result.
- SQL:1999 also supports a rich set of imperative constructs, including
  - Loops, if-then-else, assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999.

SQL Functions

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

  create function *dept_count* (*dept_name* **varchar**(20))
  returns integer
  begin
      declare *d_count* **integer;**
      select count (*) **into** *d_count*
      from *instructor*
      where *instructor.dept_name = dept_name*
      return *d_count;*
  end

- Find the department name and budget of all departments with more that 12 instructors.

  select *dept_name, budget*
  from *department*
  where *dept_*count (*dept_name*) > 1

Table Functions

- SQL:2003 added functions that return a relation as a result
- Example: Return all instructors from a given department

create function *instructors_of* (*dept_name* **char**(20)

    returns table (*ID* **varchar**(5),
        name **varchar**(20),
        dept_name **varchar**(20),
        salary **numeric**(8,2))

return table
    (**select** *ID, name, dept_name, salary*
    from *instructor*
    where *instructor.dept_name = instructors_of.dept_name*)

- Usage

    select *
    from table (*instructors_of* ('Music'))

SQL Procedures

- The *dept_count* function could instead be written as procedure:

  create procedure *dept_count_proc* (**in** *dept_name* **varchar**(20),
  out *d_count* **integer)**
  begin
  select count(*) **into** *d_count*
  from *instructor*
  where *instructor.dept_name = dept_count_proc.dept_name*
  end

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

  declare *d_count* **integer**;
  call *dept_count_proc*('Physics', *d_count*);

  Procedures and functions can be invoked also from dynamic SQL

- SQL:1999 allows more than one function/procedure of the same name (called name **overloading**), as long as the number of arguments differ, or at least the types of the arguments differ

Procedural Constructs

- Warning: most database systems implement their own variant of the standard syntax
  - read your system manual to see what works on your system
- Compound statement: **begin ... end**,
  - May contain multiple SQL statements between **begin** and **end**.
  - Local variables can be declared within a compound statements
- **While** and **repeat** statements:

```
declare n integer default 0;
while n < 10 do
    set n = n + 1
end while

repeat
    set n = n − 1
until n = 0
end repeat
```

Procedural Constructs (Cont.)

- **For** loop
  - Permits iteration over all results of a query
  - Example:

declare *n* **integer default** 0;
for *r* **as**
    select *budget* **from** *department*
    where *dept_name* = 'Music'
do
    set *n* = *n* - r.*budget*
end for

  - Statement **leave** is used to exit the loop
  - Statement **iterate** starts from the next tuple

Procedural Constructs (cont.)

- Conditional statements (**if-then-else**)
  SQL:1999 also supports a **case** statement similar to C case statement
- Example procedure: registers student after ensuring classroom capacity is not exceeded
  - Returns 0 on success and -1 if capacity is exceeded
  - See book for details
- Signaling of exception conditions, and declaring handlers for exceptions

  declare *out_of_classroom_seats* **condition**
  declare exit handler for *out_of_classroom_seats*
  begin
  …
  .. **signal** *out_of_classroom_seats*
  end

  - The handler here is **exit** -- causes enclosing **begin..end** to be exited
  - Other actions possible on exception

External Language Functions/Procedures

- SQL:1999 permits the use of functions and procedures written in other languages such as C or C++
- Declaring external language procedures and functions

create procedure dept_count_proc(**in** *dept_name* **varchar**(20),
out count **integer**)
language C
external name ' /usr/avi/bin/dept_count_proc'

create function dept_count(*dept_name* **varchar**(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'

External Language Routines (Cont.)

- Benefits of external language functions/procedures:
  - more efficient for many operations, and more expressive power.
- Drawbacks
  - Code to implement function may need to be loaded into database system and executed in the database system's address space.
    - risk of accidental corruption of database structures
    - security risk, allowing users access to unauthorized data
  - There are alternatives, which give good security at the cost of potentially worse performance.
  - Direct execution in the database system's space is used when efficiency is more important than security.

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Syntax illustrated here may not work exactly on your database system; check the system manuals

Trigger Example

- E.g. *time_slot_id* is not a primary key of *timeslot,* so we cannot create a foreign key constraint from *section* to *timeslot.*
- Alternative: use triggers on *section* and *timeslot* to enforce integrity constraints

```
create trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when (nrow.time_slot_id not in (
            select time_slot_id
            from time_slot)) /* time_slot_id not present in time_slot */
begin
  rollback
end;
```

Trigger Example Cont.

```
create trigger timeslot_check2 after delete on timeslot
    referencing old row as orow
    for each row
    when (orow.time_slot_id not in (
                select time_slot_id
                from time_slot)
                /* last tuple for time slot id deleted from time slot */
            and orow.time_slot_id in (
                select time_slot_id
                from section))
                /* and time_slot_id still referenced from section*/
    begin
        rollback
    end;
```

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - **E.g., after update of** *takes* **on** *grade*
- Values of attributes before and after an update can be referenced
  - **referencing old row as :** for deletes and updates
  - **referencing new row as :** for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blank grades to null.

create trigger *setnull_trigger* **before update of** *takes*
referencing new row as *nrow*
for each row
when (*nrow.grade* = ‘ ‘)
 begin atomic
  set *nrow.grade* = **null;**
 end;

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **for each statement** instead of **for each row**
  - Use **referencing old table** or **referencing new table** to refer to temporary tables (called ***transition tables***) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

Ranking

- Ranking is done in conjunction with an order by specification.
- Suppose we are given a relation
  student_grades(ID, GPA)
  giving the grade-point average of each student
- Find the rank of each student.

  select *ID*, **rank**() **over** (**order by** *GPA* **desc) as** *s_rank*
  from *student_grades*

- An extra **order by** clause is needed to get them in sorted order

  select *ID*, **rank**() **over** (**order by** *GPA* **desc) as** *s_rank*
  from *student_grades*
  order by *s_rank*

- Ranking may leave gaps: e.g. if 2 students have the same top GPA, both have rank 1, and the next rank is 3
  - **dense_rank** does not leave gaps, so next dense rank would be 2

Ranking (Cont.)

- Ranking can be done using basic SQL aggregation, but resultant query is very inefficient

select *ID*, (1 + (**select count**(*)
from *student_grades B*
where *B.GPA* > *A.GPA*)) **as** *s_rank*
from *student_grades A*
order by *s_rank*;

Ranking (Cont.)

- Ranking can be done within partition of the data.
- "Find the rank of students within each department."

  select *ID*, *dept_name*,
  rank () **over** (**partition by** *dept_name* **order by** *GPA* **desc**)
  as *dept_rank*
  from *dept_grades*
  order by *dept_name*, *dept_rank*;

- Multiple **rank** clauses can occur in a single **select** clause.
- Ranking is done *after* applying **group by** clause/aggregation
- Can be used to find top-n results
  - More general than the **limit** *n* clause supported by many databases, since it allows top-n within each partition

Windowing

- Used to smooth out random variations.
- E.g., **moving average**: "Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day"
- **Window specification** in SQL:
  - Given relation *sales(date, value)*

    select *date*, ***sum***(*value*) **over**
        (**order by** *date* **between rows** 1 **preceding and** 1 **following**)
    from *sales*
- Examples of other window specifications:
  - **between rows unbounded preceding and current**
  - **rows unbounded preceding**
  - **range between** 10 **preceding and current row**
    - All rows with values between current row value –10 to current value
  - **range interval** 10 **day preceding**
    - Not including current row

Data Analysis and OLAP

- **Online Analytical Processing (OLAP)**
  - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.
  - **Measure attributes**
    - measure some value
    - can be aggregated upon
    - e.g., the attribute *quantity* of the *sales* relation
  - **Dimension attributes**
    - define the dimensions on which measure attributes (or aggregates thereof) are viewed
    - e.g., the attributes *item_name, color,* and *size* of the *sales* relation

Example sales relation

item_name	*color*	*clothes_size*	*quantity*
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
pant	dark	small	14
pant	dark	medium	6
pant	dark	large	0
pant	pastel	small	1
pant	pastel	medium	0
pant	pastel	large	1
pant	white	small	3
pant	white	medium	0
pant	white	large	2

Cross Tabulation of *sales* by *item_name* and *color*

clothes_size **all**

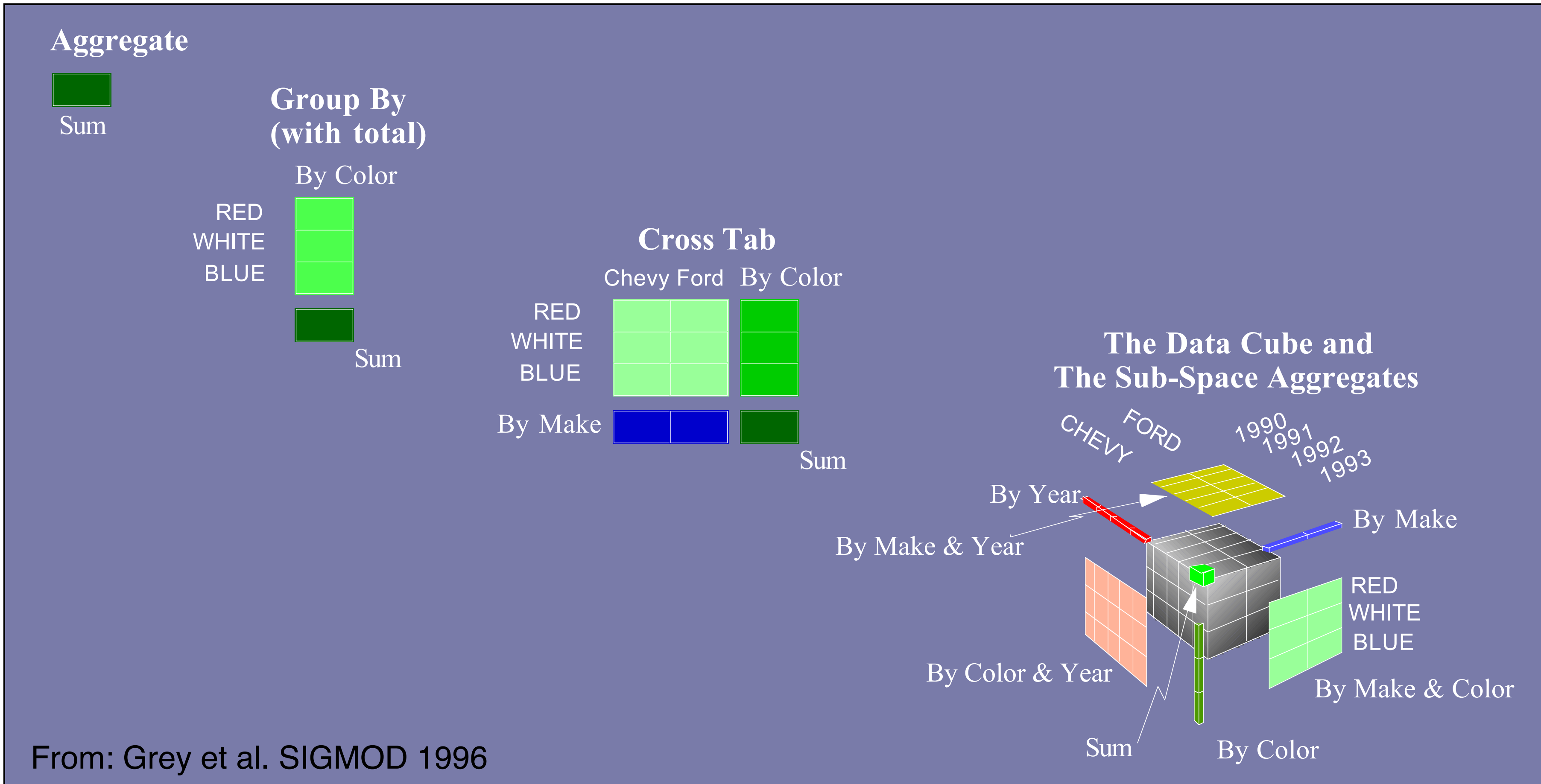
color

item_name	dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
  - Values for one of the dimension attributes form the row headers
  - Values for another dimension attribute form the column headers
  - Other dimension attributes are listed on top
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have *n* dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube



Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
  - Can drill down or roll up on a hierarchy

clothes_size: **all**

category	*item_name*	*color*				
		dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

Extended Aggregation to Support OLAP

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- Example relation:
  sales(*item_name, color, clothes_size, quantity*)
- E.g. consider the query

  select *item_name, color, size,* **sum**(*quantity*)
  from *sales*
  group by cube(*item_name, color, size*)

  This computes the union of eight different groupings of the *sales* relation:

  { (*item_name, color, size*), (*item_name, color*),
  (*item_name, size*), (*color, size*),
  (*item_name*), (*color*),
  (*size*), ( ) }

  where ( ) denotes an empty **group by** list.
- For each grouping, the result contains the null value for attributes not present in the grouping.

Online Analytical Processing Operations

- **Pivoting:** changing the dimensions used in a cross-tab
- **Slicing:** creating a cross-tab for fixed values only
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** moving from finer-granularity data to a coarser granularity
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data

OLAP Implementation

- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **relational OLAP (ROLAP)** systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **hybrid OLAP (HOLAP)** systems.