SQL3 Standardization

Ki-Joon Han
Department of Computer Engineering
Kon-Kuk University
E-mail : kjhan@db.konkuk.ac.kr
URL : *http://db.konkuk.ac.kr*

SQL Standard

- Goal: enable the portability of SQL applications across conforming products
- Side effect: Increases and stabilizes the database market
- Joint efforts between vendors and users
  - IBM, Oracle, Informix, Microsoft, ...
- Joint effort among several countries
  - Australia, Belgium, Brazil, Finland, France, Germany, Italy, Japan, Korea, Netherlands, Spain, UK, USA, ...

JTC1/SC32:
Data Management and Interchange

- WG1: Open EDI
- WG2: Metadata
- WG3: Database Languages
- WG4: SQL Multimedia and Application Packages
- WG5: Remote Database Access(RDA)
- RG1: Reference Model for Data Management(Maintenance)
- RG2: Export/Import(Maintenance)

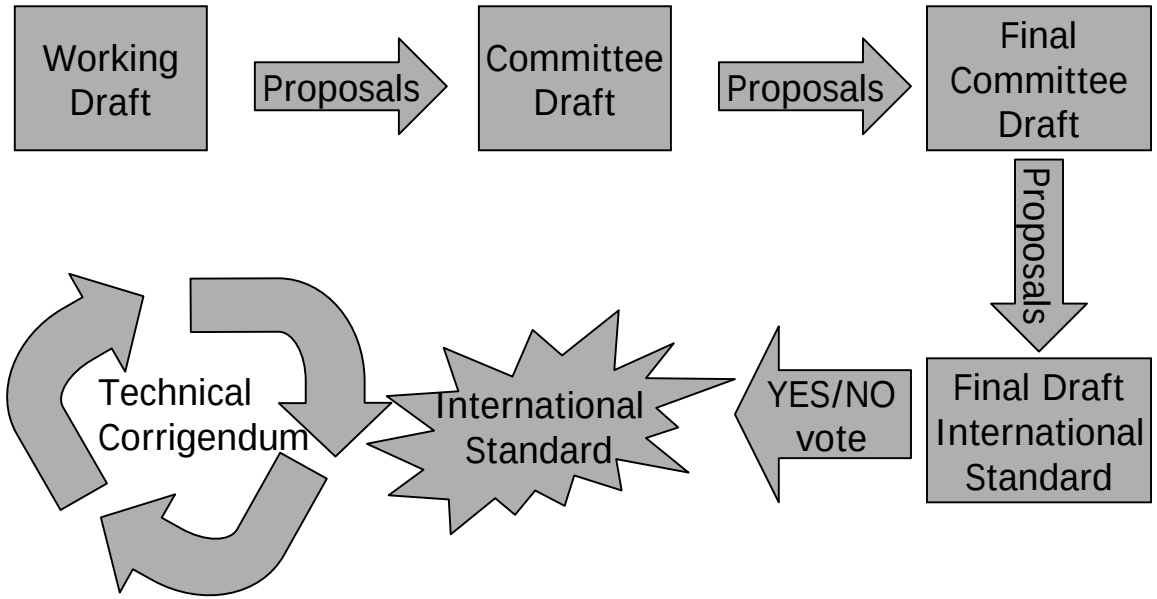
JTC1/SC32/WG3 Projects (SQL3)

- Part1: SQL/Framework 7/13/99
- Part2: SQL/Foundation 7/13/99
- Part3: SQL/CLI(Call-Level Interface) 9/1/99
- Part4: SQL/PSM(Persistent Stored Modules) 7/13/99
- Part5: SQL/Bindings withdrawal
- Part6: SQL/Transaction withdrawal
- Part7: SQL/Temporal 5/1/01
- Part9: SQL/MED(Management of External Data) 12/1/00
- Part10: SQL/OLB(Object Language Bindings) 8/1/00

- *Vendor extensions are allowed*

Database SQL Standard

- Process
  - Standards are produced by volunteers
  - Open process oriented towards achieving consensus
  - Proposals to change existing base document
- Life cycle of an ISO standard:



- Review every 5 years to reaffirm, replace, or withdraw

DBL Project History

Early 70's	Ted Codd's first papers on Relational Model
1975	CODASYL Database Specifications
1977	Database Project Initiated in U.S.
1978	ANSI Database Project Approved
1979	ISO Database Project Initiated
1982	ANSI Project Split into NDL and SQL
1983	ISO Project Split into NDL and SQL
1986	ANSI SQL Published – December
1987	ISO/IEC 9075:1986 (SQL86)
1989	ISO/IEC 9075:1989 (SQL89)
1992	ISO/IEC 9075:1992 (SQL92)
1995	ISO/IEC 9075-3:1995 (SQL/CLI for SQL92)
1996	ISO/IEC 9075-4:1996 (SQL/PSM for SQL92)

Progression of SQL Standards

- SQL/86
- SQL/89 (FIPS 127-1)
- SQL/89 with Integrity Enhancement
- SQL/92 — July 92
  - Entry Level (FIPS 127–2)
  - Intermediate Level
  - Full Level
- SQL CLI — Sept 95
- SQL PSM — Nov 96
- SQL/3(Work in Progress)
  - SQL Framework — July 99
  - SQL Foundation — July 99
  - SQL Call Level Interface(CLI) — Sept 99
  - SQL Persistent Stored Modules(PSM) — July 99
  - SQL Language Bindings — withdrawal
  - SQL Management of External Data — Dec 00
  - SQL Object Language Bindings — Aug 00
- SQL/4(Work to be defined soon)
  - All of the above, and ...
  - XA
  - SQL Temporal

SQL/86 (ISO/IEC 9075:1986)

- The starting point: IBM's SQL implementation
  - SQL/86 became a subset of IBM's SQL implementation
- Defined 3 ways to process DML
  - Direct processing
  - Module language
  - Embedded SQL
- Bindings to
  - Cobol
  - Fortran
  - Pascal
  - PL/1

SQL/89 (ISO/IEC 9075:1989)

- Superset of SQL/86
- Replaced SQL/86
- C and ADA were added to existing language bindings
- DDL in a separate “schema definition language”

  CREATE TABLE

  CREATE VIEW

  GRANT PRIVILEGES

  (No DROP, ALTER, OR REVOKE)

SQL/89 with Integrity Enhancement

- DEFAULT
  - Default value for a column when omitted at INSERT time
- UNIQUE (column-list)
- NOT NULL
- Views WITH CHECK OPTION
  - Insertions to view are rejected if they don't satisfy the view-definition
- PRIMARY KEYs
- CHECK constraint
  - Integrity constraint on values in a single row
- Referential Integrity

  CREATE TABLE T2

  …. FOREIGN KEY (COL3) REFERENCES T1 (COL2)
  - Any update that would violate referential integrity is rejected

SQL/92: Overview (1/2)

- Superset of SQL/89
- Not “ least-common-denominator”
- Significantly larger than SQL/89(579 versus 120 pages)
  - Data type extensions(varchar, bit, character sets, date, time & interval)
  - Multiple join operators
  - Catalogs
  - Domains
  - Derived tables in FROM clause
  - Assertions
  - Temporary tables
  - Referential actions
  - Schema manipulation language
  - Dynamic SQL
  - Scrollable cursors
  - Connections
  - Information schema tables

SQL/92: Overview (2/2)

- Many (but not all) features are available in exsiting products
- Divided into 3 levels
  - Entry level (much the same as SQL/89 with Integrity Enhancement)
  - Intermediate level
  - Full level
- Features are assigned to level
  - Full is a superset of Intermediate
  - Intermediate is a superset of Entry
- FIPS 127-2 defines a Transitional Level
  - Level between Entry and Intermediate
  - Subset of Intermediate
  - Superset of Entry

SQL/92: Entry Level

- SQL/89 plus a small set of new features
  - SQLSTATE
    - Carries more feedback information than SQLCODE
  - Delimited identifiers

    CREATE TABLE “SELECT”…
  - Named expressions in SELECT - list

    SELECT name, sal+comm AS pay
    FROM employee
    ORDER BY pay

SQL/92: Transitional Level (1/2)

- Defined by FIPS 127-2
- Subset of SQL/92: Intermediate Level
- Data types and operators

  DATE, TIME, TIMESTAMP, INTERVAL(with arithmetic)

  CHAR VARYING(n)

  LENGTH, SUBSTR, TRIM, and || (concatenate) operators
- Referential integrity with cascading delete
- New types of join
  - NATURAL JOIN
  - LEFT and RIGHT OUTER JOIN
- Dynamic SQL

  PREPARE

  EXECUTE

  DESCRIBE

SQL/92: Transitional Level (2/2)

- Schema evolution
  ALTER TABLE
  DROP TABLE
  REVOKE PRIVILEGE
- CAST(expression AS type)
  - Conversions among
    - Numeric types
    - Numeric <-> Character
    - Character <-> Date and Time
- Standard Catalogs
  - TABLES VIEWS COLUMNS
  - PRIVILEGE
- Views containing UNION
- Multiple schemas(collection of tables and other objects) per user
- Transaction isolation levels
  READ UNCOMMITTED
  READ COMMITTED
  REPEATABLE READ
  SERIALIZABLE

SQL/92: Intermediate Level (1/3)

- Scrollable cursors
- FULL OUTER JOIN
- Domains
  - Macro facility for data type, default, value, nullability, and CHECK constraint
  - No strong typing (type checking based on underlying data type)
  - Not the same as Codd's notion of domains
- Online DDL
- Implicit casting
  - Scalar-valued subquery can be used in place of any scalar

SQL/92: Intermediate Level (2/3)

- Set operations between query blocks
  - INTERSECT
  - EXCEPT
  - CORRESPONDING (allows operators to apply to like-named columns of tables)
- CASE expression

  SELECT CASE (sex)
  WHEN “F” THEN “female”
  WHEN “M” THEN “male”
  END
  …
- COALESCE
  - Returns the first non-null value

  COALESCE(EMP.AGE, “Age is null”)

SQL/92: Intermediate Level (3/3)

- UNIQUE predicate

  UNIQUE <subquery>

  - Returns true if the subquery returns no duplicates; otherwise, false
- 128-character identifiers
- Multiple character sets (including double-byte)
- SET statement to change authorization-ID
- More comprehensive catalog information
  - Constraints
  - Usage
  - Domains
  - Assertions
- Date and time arithmetic with time zones
- SQL FLAGGER
  - Extensions
  - Conforming language being processed in a non-conforming way

SQL/92: Full Level (1/3)

- Derived tables
  - Table-expressions in FROM-clause
- Referential integrity with CASCADE UPDATE and SET NULL
- Integrity assertions
  - Stand-alone assertions that apply to entire tables or multiple tables
  - Subqueries in CHECK clause
  - Deferred checking of constraints (including assertions)
- Enhanced predicates
  - Multiple-column matching

    WHERE (X, Y) MATCH (SELECT A, B FROM T2)
  - Comparison by high-order and low-order columns

    WHERE (X, Y) > (A, B)

SQL/92: Full Level (2/3)

- More types of join
  - CROSS JOIN
  - UNION JOIN
- New data types
  - BIT (n)
  - BIT VARYING(n)
- Temporary tables (vanish at end of transaction or session)
- Implementation-defined collating sequences
- More character-string operators
  - UPPER
  - LOWER
  - POSITION
- INSERT privilege on individual columns

SQL/92: Full Level (3/3)

- Row and table constructors

```
(   ( 1, 'OPERATOR', 'JONES'),
    ( 2, 'PROGRAMMER', 'SMITH'),
    ( 3, 'MGR', 'MATTOS')
)
```

- Explicit Tables
  - TABLE EMP can be a subquery
- DISTINCT applies to expression

```
SELECT COUNT (DISTINCT SAL+COMM)
```

- Cursors declared SENSITIVE or INSENSITIVE
- Updates via scrollable or ordered cursors
- UPDATE and DELETE with subqueries on the same table

SQL99 Overview

- Superset of SQL/92
  - Completely upward compatible ("**object-oriented SQL**")
- Significantly larger than SQL/92
  - Object-Relational extensions
    - User-defined data types
    - Reference types
    - Collection types (e.g., arrays)
    - Large object support (LOBs)
    - Table hierarchies
  - Triggers
  - Stored procedures and user-defined functions
  - Recursive queries
  - OLAP extensions (CUBE and ROLLUP)
  - SQL procedural constructs
  - Expressions in ORDER BY
  - Savepoints
  - Update through unions and joins

SQL99 Multipart Standard

- SQL/Framework (Part 1)
  - Overview and conformance clause
- SQL/Foundation (Part 2)
  - The basics: types, schemas, tables, views, query and update statements, expressions, security model, predicates, assignment rules, transaction management, and so forth
- SQL/CLI (Call Level Interface) (Part 3)
  - No preprocessing of SQL statements necessary
- SQL/PSM (Persistent Stored Modules) (Part 4)
  - Extensions to SQL to make it procedural
- SQL/Bindings (Part 5)
  - Dynamic, embedded, direct invocation

SQL99 Framework Overview

- Overview
  - Provides an overview of the complete standard
- Conformance
  - Contains conformance requirements
  - Conformance model based on “**Core SQL**” and “**Packages**”

SQL99 Foundation Overview (1/7)

- All of SQL/92 functionality
  - Schemas
  - Different kinds of joins
  - Temporary tables
  - CASE expressions
  - Scrollable cursors
  - ...
- New built-in data types for increased modeling power
  - Boolean
  - Large objects (LOBs)
- Enhanced update capabilities
  - Update/delete through unions
  - Update/delete through joins
- Other relational extensions to increase modeling and expressive power
  - Additional predicates (FOR ALL, FOR SOME, SIMILAR TO)
  - Extensions to cursors (sensitive cursor, holdable cursor)
  - Extensions to referential integrity (RESTRICT)
  - Extensions to joins

SQL99 Foundation Overview (2/7)

- Triggers
  - Enhances integrity mechanism (active DBMS)
    - Different triggering events: update/delete/insert
    - Optional condition
    - Activation time: before or after
    - Multi-statement action
    - Several triggers per table
    - Condition and multi-statement action per each row or per statement
- Roles
  - Enhanced security mechanisms
    - GRANT/REVOKE privileges to roles
    - GRANT/REVOKE roles to users and other roles

SQL99 Foundation Overview (3/7)

- Recursion
  - Increase expressive power
  - Linear (both direct and mutual) recursion
  - Stop conditions
  - Different search strategies (depth first, breadth first)
- Savepoints
  - Enhances user-controlled integrity
  - Savepoint definition
  - Roll back to savepoint
  - Nesting
- OLAP extensions
  - Enhances query capabilities
    - CUBE
    - ROLLUP
    - Expressions in ORDER BY

SQL99 Foundation Overview (4/7)

- Object-relational extensions
  - Extensibility
  - Increases modeling power (complex objects)
  - Reusability
  - Integration
- User-defined types
  - Distinct types
    - Strong typing
    - Type-specific behavior
  - Structured types
    - Strong typing
    - Type-specific behaviors
    - Encapsulation
    - Value substitutability
    - Polymorphic routines
    - Dynamic binding (run-time function dispatch)
    - Compile-time type checking

SQL99 Foundation Overview (5/7)

- Collection types
  - Arrays
- Row types
  - Like record structures in programming languages
  - Type of rows in tables
  - Nesting (rows with row-valued fields)
- Reference types
  - Support “object identity”
  - Navigational access (path expressions)

SQL99 Foundation Overview (6/7)

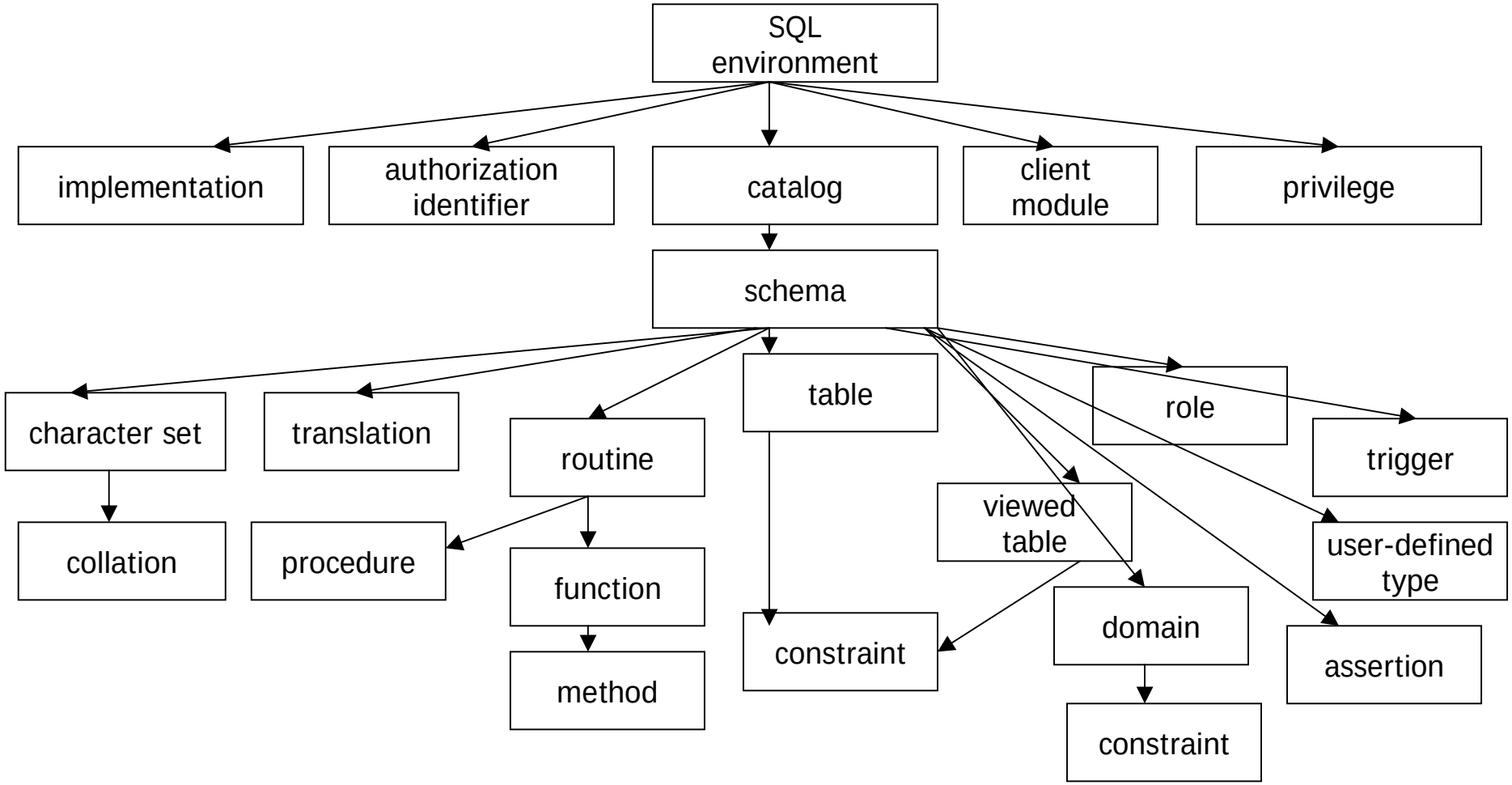
- User-defined functions
  - SQL and external functions
  - Overloaded functions
  - User-defined paths
  - Compile time type checking
  - Static binding
- User-defined procedures
  - SQL and external procedures
  - No overloading
  - Input and output parameters
  - Result sets
  - Static binding
- User-defined methods
  - Describe a user-defined type behavior
  - SQL and external methods
  - Overloading and overriding
  - Compile time checking
  - Late binding (dynamic dispatch)

SQL99 Foundation Overview (7/7)

- Subtables (table hierarchies)
  - Increase modeling power and expressive power of queries
  - Means to model collection hierarchies or object extents
    - CREATE/DROP subtable
    - CREATE/DROP subview
    - Object “identity” by means of references
    - Queries on a table operate on subtables as well
    - “Object-like” manipulation through references and path expressions
    - Extensions to authorization model to support “ object-like” manipulation

- View hierarchies (object hierarchies)

Database Objects



Catalogs and Schemas

- SQL objects (i.e., tables, views,...) are contained in schemas
- Schemas are contained in catalogs
- Each schema has a single owner
- Objects can be referenced with explicit or implicit catalog and schema name

```
FROM people                 --unqualified name
FROM sample.people          --partially qualified name
FROM cat1.sample.people     --fully qualified name
```

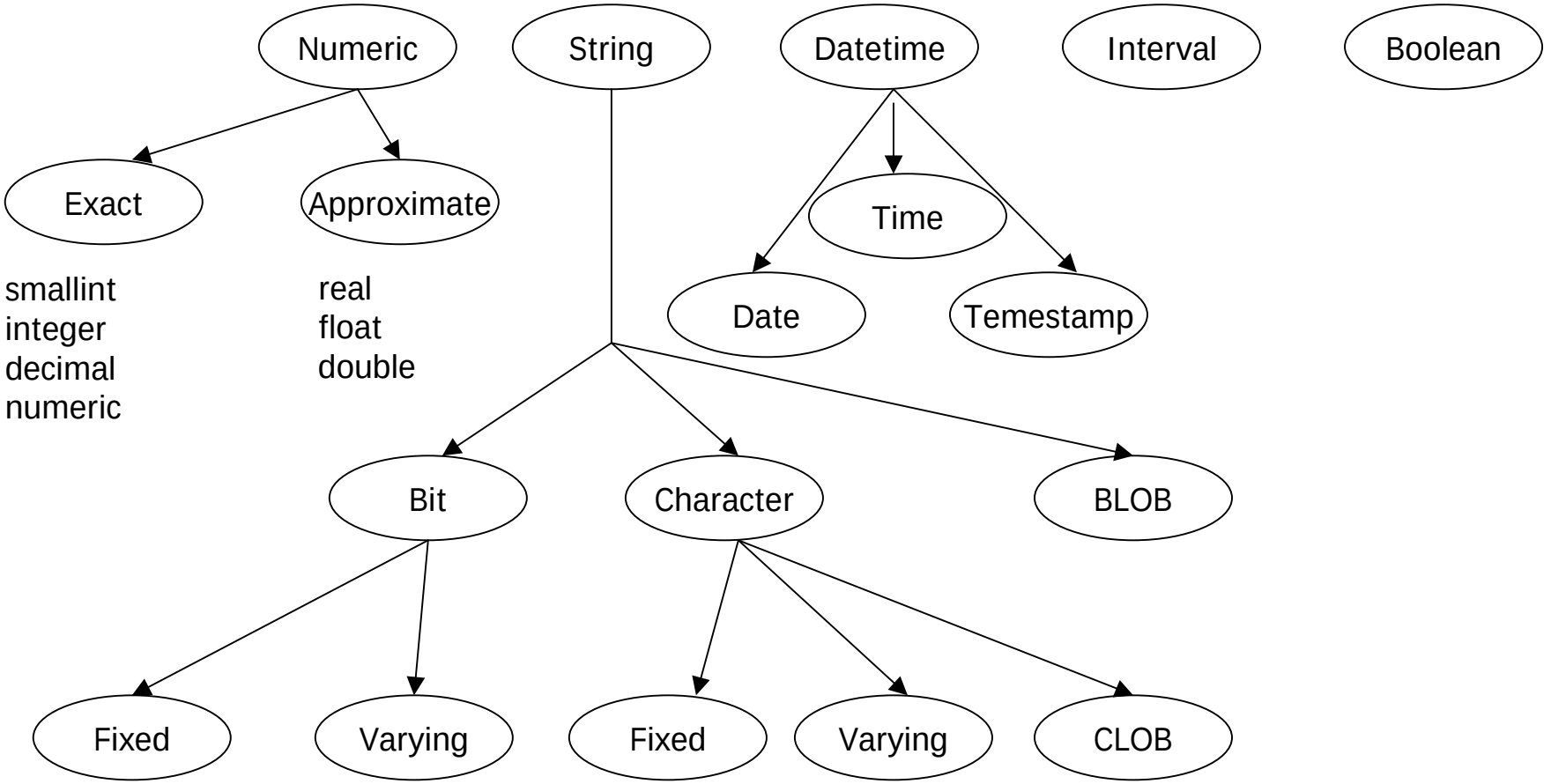
Schema Manipulation Language

- Syntax for creating objects
- Syntax for dropping or revoking with two behaviors
  - RESTRICT
  - CASCADE
- Syntax for altering objects
  - Table
    - Add/drop column
    - Alter column default and scope
    - Add/drop constraints
  - Domain
    - Set/drop default
    - Add/drop constraint
  - User-defined type
    - Add/drop attribute
    - Add/drop method
  - SQL-invoked routines
    - Alter routine characteristics

Data Types

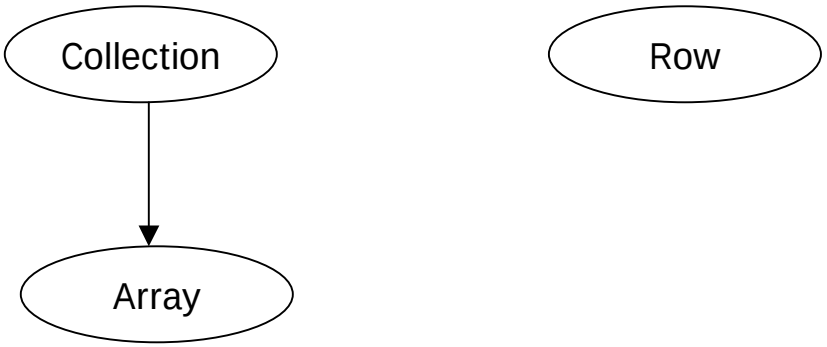
- Predefined types
  - Numeric
  - String
  - BLOB
  - Boolean
  - Datetime
  - Interval
- Constructed atomic types
  - Reference
- Constructed composite types
  - Collection: Array
  - Row
- User-defined types
  - Distinct type
  - Structured type

Predefined Types



Constructed Types

- Atomic
  - Currently, only one: ***reference type***
- Composite



More collection types likely in SQL4

Domains

- Persistent (named) definition of
  - A data type
  - An optional default value
  - An optional set of constraints
  - An optional collating sequence
- Used in place of a data type
- Do not provide strong typing
  - Not true " relational domains"

```
CREATE DOMIN money AS DECIMAL (7,2);

CREATE DOMAIN account_type AS CHAR (1)
DEFAULT 'C'
CONSTRAINT account_type_check CHECK (value IN ('C','S','M'));

CREATE TABLE accounts
(account_id    INTEGER,
 balance       money,
 type          account_type);
```

SQL-Invoked Routines (1/2)

- Named persistent code to be invoked from SQL
  - SQL-invoked **procedures**
  - SQK-invoked **functions**
  - SQL-invoked **methods**
- Created directly in a schema or in a SQL-server module
  - Schema-level routines
  - Module-level routines
- Have schema-qualified 3-part names
- Supported DDL
  - CREATE and DROP statements
  - ALTER statement – still limited in functionality
  - EXECUTE privilege controlled through GRANT and REVOKE statements
- Described by corresponding information schema views

SQL-Invoked Routines (2/2)

- Have a header and a body
  - Header consists of a name and a (possibly empty) list of parameters
- Parameters of procedures may specify parameter mode
  - IN
  - OUT
  - INOUT
- Parameters of functions are always IN
- Functions return a single value
  - Header must specify data type of return value via RETURNS clause
- SQL routines
  - Both header and body specified in SQL
- External routines
  - Header specified in SQL
  - Bodies written in a host programming language
    - May contain SQL by embedding SQL statements in host language programs or using CLI

SQL Routines (1/2)

- Parameters
  - Must have a name
  - Can be of any SQL data type
- Routine body
  - Consists of a single SQL statement
    - Can be a compound statement : BEGIN ...END
  - Not allowed to contain
    - DDL statement
    - CONNECT or DISCONNECT statement
    - Dynamic SQL
    - COMMIOT or ROLLBACK statement

```
CREATE PROCEDURE get_balance(IN acct_id INT, OUT bal
DECIMAL(15,2))
  BEGIN
      SELECT balance INTO bal
            FROM accounts WHERE account_id =acct_id;
      IF bal <100
            THEN SIGNAL low_balance
      END IF;
  END
```

SQL Routines (2/2)

- Routine body
  - RETURN statement allowed only inside the body of a function
    - Exception raised if function terminates not by a RETURN

```
CREATE FUNCTION get_balance(acct_id INT) RETURNS
DECIMAL(15,2)
    BEGIN
        DECLARE bal DECIMAL(15,2);
        SELECT balance INTO bal
            FROM accounts
            WHERE account_id = acct_id;
        IF bal <100 THEN SIGNAL low_balance
            END IF;
    RETURN bal;
    END
```

External Routines (1/2)

- Parameters
  - Names are optional
  - Cannot be of any SQL data type
  - Permissible data types depend on the host language of the body
- LANGUAGE clause
  - Identifies the **host language** in which the body is written
- NAME clause
  - Identifies the **host language code**, e.g., file path in Unix
  - If unspecified, it corresponds to the routine name

```
CREATE PROCEDURE get_balance (IN acct_id INT, OUT bal
DECIMAL(15,2))
LANGUAGE C
EXTERNAL NAME 'bank₩balance_proc'

CREATE  FUNCTION get_balance (IN INTEGER) RETURNS
DECIMAL(15,2)
LANGUAGE C
EXTERNAL NAME 'usr/han/banking/balance'
```

External Routines (2/2)

- RETURNS clause may specify CAST FROM clause

  CREATE FUNCTION get_balance(IN INT)
  RETURNS DECIMAL(15,2) CAST FROM REAL
  LANGUAGE C

  - C program returns a REAL value, which is then cast to DECIMAL(15,2) before returning to the caller

- Special provisions to handle null indicators and the status of execution (SQLSTATE)
  - PARAMETER STYLE SQL (is the default) : 2n + 4
  - PARAMETER STYLE GENERAL : 2n + 6

Routine Overloading

- Multiple routines with the same unqualified name

S1.F (p1 INT, p2 REAL)
S1.F (p1 REAL, p2 INT)
S2.F (p1 INT, p2 REAL)

- Within the same schema : must have a unique signature

S1.F (p1 INT, p2 REAL)
S1.F (p1 REAL, p2 INT)

- Across schemas : may have the same signature

S1.F (p1 INT, p2 REAL)
S2.F (p1 INT, p2 REAL)

- Only functions can be overloaded. Procedures cannot be overloaded.

Specific Names

- Uniquely identifies each routine in the database
  - If unspecified, an implementation-dependent name is generated

```
CREATE FUNCTION get_balance(acct_id INTEGER)
RETURNS DECIMAL(15,2)
SPECIFIC func1
BEGIN
...
RETURN...;
END
```

- Can only be used to identify the routine in ALTER, DROP, GRANT, and REVOKE statements

```
DROP SPECIFIC FUNCTION fucn1 RESTRICT;
```

- DDL statements can also identify a routine by providing the name and the list of parameter types

```
DROP FUNCTION get_balance(INTEGER) CASCADE;
```

- Cannot be used to invoked a routine

Routine Invocation

- Procedure –invoked by a CALL statement

  CALL get_balance(100, bal);

- Function -- invoked as part of an expression

  SELECT account_id, get_balance (account_id)
  FROM accounts

- Requires the invoker to have EXECUTE privilege on the routine – Otherwise no routine will be found for the invocation

Object-Relational Support: Motivation

- Database systems provide
  - A set of types used to represent the data in the application domain
  - A set of operations (functions) to manipulate these types

TYPE	FUNCTION
INTEGER	+,-,/,*,...
CHAR	SUBSTRING, CONCAT,...
DATE	DAY,MOTH,YEAR,...

- Increasing need for extension
  - New types required to better represent the application domain
  - New operations (functions) required to better reflect the behavior of the types

TYPE	FUNCTION
MONEY	+,-,INTEREST,...
CHAR	CONTAINS,SPELLCHECKING,...
IMAGE	WIDTH,HEIGHT,THUMBNAIL, ...

Major Extensions in SQL99

- Mechanism for “objects”(specific types and their behavior - functions/methods)
  - User Defined Types (UDTs):Text,Image,CAD/CAM Drawing, Video…
  - User Defined Functions(UDFs):Contains,Display,Rotate,Play…
- Support for storage/manipulation of large data types
  - Large Objects (LOBs):Binary, Character
- Mechanism to improve the DB integrity and to allow checking of business rules inside the DBMS
  - Triggers: Auditing, Cross-Referencing, Alerts ...
- Means to express complex data relationships such as hierarchies, bills-of material, travel planning ...
  - Recursion
  - Update through UNION and JOIN
  - Common Table Expressions

Upward compatible extension of SQL to guarantee application portability and database independence!

Object-Relational Support

- Large Objects (LOBs)
  - Binary
  - Character
- User-Defined Data Types
  - Distinct types
  - Structured types
- Type Constructors
  - Row types
  - Reference types
- Collection Types
  - Arrays
- User-Defined Methods, Functions, and Procedures
- Typed tables and views
  - Table hierarchies
  - View hierarchies (object views)

What Are Large Objects (LOBs)?

- LOBs are a new set of data types
  - LOBs store strings of up to gigabytes
- There are 2 new data types
  - **BLOB** - Binary Large Object
    - Useful for Audio, Image data
  - **CLOB** - Character Large Object
    - Useful for character data (text)

Large Object Data Types

- Maintained directly in the database
- Not in “external files”
- LOB size can be specified at column definition time
  (in terms of KB, MB, or GB)

```
CREATE TABLE  BookTable
(title        VARCHAR(200),
book_id       INTEGER,
summary       CLOB(32K),
book_text     CLOB(20M),
movie         BLOB(2G))
```

How Do You Use LOBS?

- LOBs may be retrieved, inserted, updated like any other type
  - You must acquire **buffers** large enough to store the LOBs
    - This may be difficult for very large LOBs
    - SQL99 provides **locators** to make LOB access manageable

```
EXEC SQL
   SELECT summary, book_text, movie
   INTO :bigbuf,:biggerbuf,:massivebuf
   FROM  BookTable
   WHERE title="Moby Dick';
```

```
BookTable :
  title          VARCHAR(200)
  book_id        INTEGER
  summary        CLOB(32K)
  book_text      CLOB(20M)
  movie          BLOB(2G)
```

LOB Functions

- Functions that support LOBs
  - CONCATENATION ***string1***|| ***string2***
  - SUBSTRING(***string FROM start FOR length***)
  - LENGTH(***expression***)
  - POSITION(***search-string IN source-string***)
  - NULLIF/COALESCE
  - TRIM
  - OVERLAY
  - ***Cast***
  - ***User-defined functions***
  - **LIKE predicate**

```
EXEC SQL
   SELECT POSITION('Chapter 1' IN book_text)
   INTO :int_variable
   FROM  BookTable
   WHERE title='Moby Dick';
```

Locators (1/2)

- *Locator*: 4-byte value stored in a host variable that a program can use to refer to a LOB value
  - Application declares ***locator variable***, and then may set it to refer to the current value of a particular LOB
  - A locator may be used anywhere a LOB value can be used

```
EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB_LOCATOR
        movie_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL
    SELECT movie
    INTO :movie_loc
    FROM BookTable
    WHERE title ='Moby Dick'
```

Locators (2/2)

- HOLD locator
  - Maintains the LOB value and locator after the commit of a transaction
- FREE locator
  - Frees a locator and its LOB value

```
SELECT book_text
   INTO :LOB_locator
   FROM BookTable
   WHERE title ='Moby Dick';

HOLD LOCATOR :LOB_locator;

COMMIT;

INSERT INTO my_favor_books
   VALUES (...,:LOB_locator,...)

FREE LOCATOR :LOB_locator;
```

User-Defined Types

- User-defined data types
  - User-defined, named type representing entities
  - employee, project, money, polygon, image, text, language, format, ...

  (1) **Distinct types**
    - based on a predefined types
    - no inheritance

  (2) **Structured types**
    - one or more attributes
    - type hierarchy supported
- User-defined methods and functions (operators)
  - User-defined operation representing the behavior of entities in the application domain
  - hire, appraisal, convert, area, length, contains, ranking, ...

User-Defined Distinct Types (1/2)

```
CREATE TABLE  RoomTable (
RoomID            CHAR(10),
RoomLength        INTEGER,
RoomWidth         INTEGER,
RoomPerimeter     INTEGER,
RoomArea          INTEGER);
```

```
UPDATE RoomTable
SET RoomArea = RoomLength;
```

No Error Results !!!

- Before SQL99, columns could only be defined with the existing built-in data types
  - There was no strong typing
  - Logically incompatible variables could be assigned to each other

User-Defined Distinct Types (2/2)

```
CREATE TYPE plan.roomtype
AS CHAR(10) FINAL;

CREATE TYPE plan.meters
AS INTEGER FINAL;

CREATE TYPE plan.squaremeters
AS INTEGER FINAL;

CREATE TABLE RoomTable (
RoomID          plan.roomtype,
RoomLength      plan.meters,
RoomWidth       plan.meters,
RoomPerimeter   plan.meters,
RoomArea        plan.squaremeters);
```

```
UPDATE RoomTable
SET RoomArea =
RoomLength;
```

ERROR

```
UPDATE RoomTable
SET RoomLength =
RoomWidth;
```

NO ERROR RESULTS

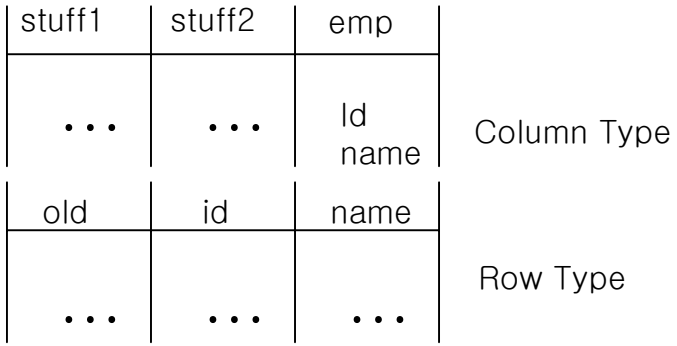
"No inheritance or subtyping"

Each UDT is logically incompatible with all other types

User-Defined Structured Types

- Column Types
  - E.g.,text, image, audio, video, time series, point, line,...
  - For modeling new kinds of *facts* about enterprise entities
  - Enhanced infrastructure for SQL/MM
- Row Types
  - Types and functions for rows of tables
  - E.g., employees, departments, universities, students, ...
  - For modeling e*ntities* with *relationships* & *behavior*
  - Enhanced infrastructure for business objects

```
CREATE TYPE employee
AS
(id       INTEGER,
name     VARCHAR(20))
```



Structured Types: Examples

```
CREATE TYPE address AS
(street              CHAR(30),
city                 CHAR(20),
state                CHAR(2),
zip                  INTEGER) NOT FINAL

CREATE TYPE bitmap AS BLOB FINAL

CREATE TYPE real_estate AS
(owner                       REF (person),
price                        money,
rooms                        INTEGER,
size                         DECIMAL(8,2),
location                     address,
text_description             text,
front_view_image             bitmap,
document                     doc) NOT FINAL
```

Use of Structured Types

- Wherever other (predefined data) types can be used in SQL
  - Type of attributes of other structured types
  - Type of parameters of functions, methods, and procedures
  - Type of SQL variables
  - Type of domains or columns in tables

  CREATE TYPE address AS (street CHAR (30), **...**) NOT FINAL
  CREATE TYPE real_estate AS (**...** location address, **...**) NOT FINAL

- To define tables and views

  CREATE TABLE properties OF real_estate ...

Methods (1/2)

- What are methods?
  - SQL-invoked functions “attached” to **user-defined types**
- How are they different from functions?
  - Implicit SELF parameter (called subject parameter)
  - Two-step creation process: **signature** and **body** specified separately
  - Must be created in the type’s schema
  - Different style of invocation (UDT value.method(...))

```
CREATE TYPE  employee AS
(name              CHAR(40),
base_salary        DECIMAL(9,2),
bonus              DECIMAL(9,2))
INSTANTIABLE NOT FINAL
METHOD salary() RETURNS DECIMAL(9,2);

CREATE METHOD salary() FOR employee
BEGIN
.....
END;
```

Methods (2/2)

- Original methods: methods attached to super type
- Overriding methods: methods attached to subtypes

```
CREATE TYPE  employee AS
(name                CHAR(40)
base_salary          DECIMAL(9,2),
bonus                DECIMAL(9,2))
INSTANTIABLE NOT FINAL
METHOD salary() RETURNS DECIMAL(9,2);

CREATE TYPE manager UNDER employee AS
(stock_option INTEGER)
INSTANTIABLE NOT FINAL
OVERRIDING METHOD salary() RETURNS DECIAML(9,2), -- overriding
METHOD vested() RETURNS INTEGER;                 -- original
```

- Invoked using dot syntax (assume dept table has mgr column)

  SELECT mgr.salary() FROM dept;

Creating Structured Types

- System-supplied constructor function
  - address() -> address or real_estate() -> real_estate
    - Returns new instance with attributes initialized to their default
- NEW operator
  - NEW <method name> <list of parameters>
    - invokes constructor function before invoking method
- INSERT statement against a typed table

CREATE TABLE properties OF real_estate **...**

INSERT INTO properties VALUES (:owner, money (350000),15, 4500,
NEW address (‘1543 3rd Ave. North, Sacramento, CA 93523’)**...**)

SELECT owner, price FROM properties
WHERE address = gen_address (address(), ‘1543 3rd Ave. North. Sacramento, CA 93523’)

Uninstantiable Types

- Structured types can be uninstantiable
  - Like abstract classes in OO language
    - ***No system-supplied constructor function is generated***
    - ***Types does not have instances of its own***
  - Instances can be defined on subtypes

- By default, structured types are instantiable

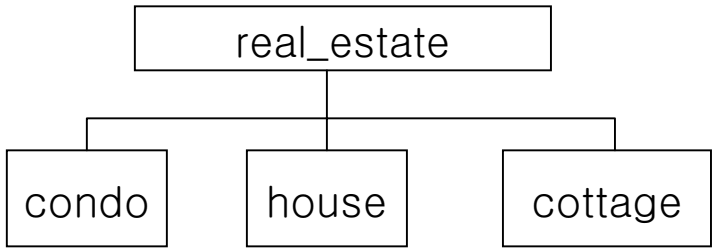
- Distinct types are always instantiable

```
CREATE TYPE person AS
(name        VARCHAR(30),
 address     address,
 sex         CHAR(1)) NOT INSTANTIABLE NOT FINAL
```

Subtyping and Inheritance (1/2)

- Structured types can be a subtype of another UDT
- UDTs **inherit** structure (attributes) and behavior (methods) from their supertypes (*single inheritance*)
- FINAL and NOT FINAL
  - ***FINAL types may not have subtypes***
  - In SQL99, structured types must be NOT FINAL and distinct types must be FINAL
  - In SQL4, both options will be allowed

```
CREATE TYPE real_estate ... NOT FINAL
CREATE TYPE condo UNDER real_estate ... NOT FINAL
CREATE TYPE house UNDER real_estate ... NOT FINAL
```



Subtyping and Inheritance (2/2)

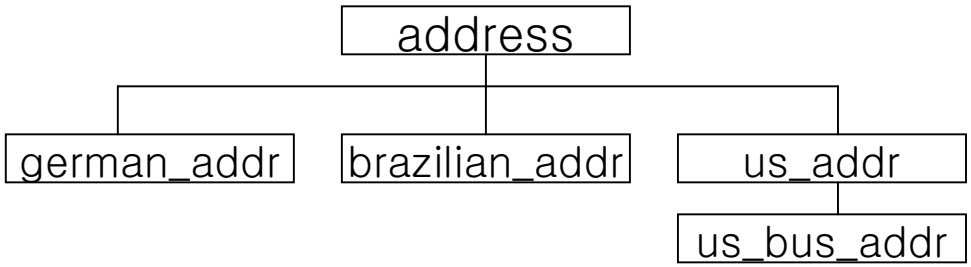
CREATE TYPE address AS
(street CHAR(30), city CHAR(20), state CHAR(2), zip INTEGER) NOT FINAL

CREATE TYPE german_addr UNDER address
(family_name VARCHAR(30)) NOT FINAL

CREATE TYPE brazilian_addr UNDER address
(neighborhood VARCHAR(30)) NOT FINAL

CREATE TYPE us_addr UNDER address
(area_code INTEGER, phone INTEGER) NOT FINAL

CREATE TYPE us_bus_addr UNDER us_addr
(bus_area_code INTEGER, bus_phone INTEGER) NOT FINAL



Value Substitutability (1/2)

- Each row can have a value a difference subtype

INSERT INTO properties (price, owner, location)
VALUES (US_dollar (100000), REF('Mr.S.White'), New us_addr ('1654 Health Road', 'Health', 'OH', 45394, ...))

INSERT INTO properties (price, owner, location)
VALUES (real (400000), REF('Mr.W.Green'), NEW brazilian_addr('245 Cons. Xavier da Costa', 'Rio de Janeiro', 'Copacabana'))

INSERT into PROPERTIES (price, owner, location)
VALUES(german_mark (150000), REF('Mrs.D.Black'), NEW german_addr('305 Kurt-Schumacher Strasse', 'Kaiserslautern', 'Prof. Dr. Heuser'))

type tag

price	owner	Location
<us_dollar> amount 100,000	'Mr.S.White'	*<us_addr>* '1654 Health ...'
<real> amount 400,000	'Mr.W.Green'	*<brazilian_addr>* '245 Cons.Xavier...'
<german_mark> amount 150,000	'Mrs. D. Black'	*<german_addr>* '305 Kurt-Schumacher...'

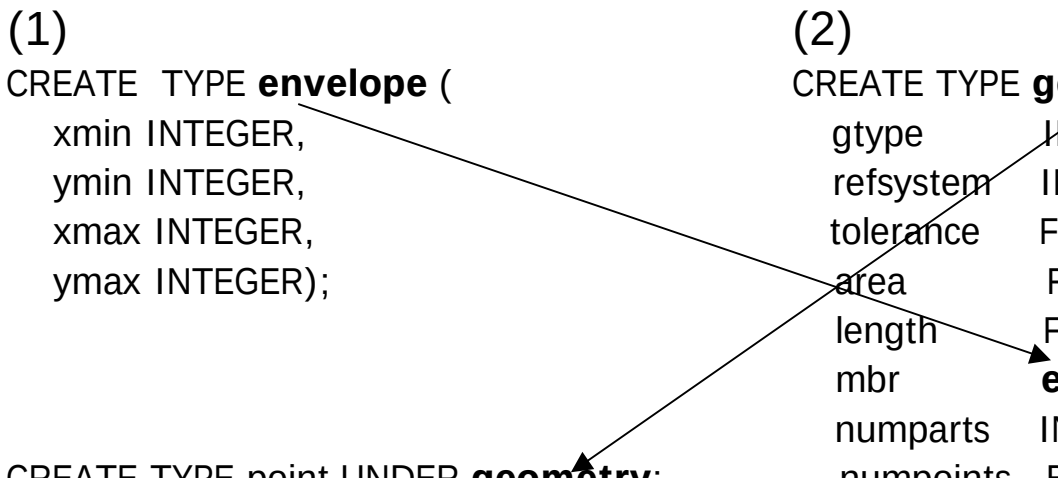
Value Substitutability (2/2)

- An instance of a subtype can be found at runtime
  (requires dynamic dispatch - late binding)

  SELECT owner, **price.dollar_amount ( )**
  FROM properties
  WHERE price.dollar_amount ( ) < US_dollar (500000)

  - Will cause the invocation of a different method, depending on the type of money stored in the column PRICE (i.e., US_dollar, CDN_dollar, D_mark, S_frank, real, ...)
  - Only methods are dynamically dispatched
    - Functions are statically selected

Structured Types as Column Types (1/3)



(1)

```
CREATE  TYPE envelope (
    xmin INTEGER,
    ymin INTEGER,
    xmax INTEGER,
    ymax INTEGER);
```

(2)

```
CREATE TYPE geometry (
    gtype        INTEGER,
    refsystem    INTEGER,
    tolerance    FLOAT,
    area         FLOAT,
    length       FLOAT,
    mbr          envelope,
    numparts     INTEGER,
    numpoints    BLOB(1m),
    points       BLOB(1m),
    zvalue       BLOB(500k),
    measure      BLOB(500k));
```

```
CREATE TYPE point UNDER geometry;
CREATE TYPE line UNDER geometry;
CREATE TYPE polygon UNDER geometry;
```

(3)

```
CREATE FUNCTION distance
(s1 geometry, s2 geometry)
RETURNS BOOLEAN
EXTERNAL NAME
'/usr/lpp/db2se/gis!shapedist'
  ...
```

(4)

```
CREATE FUNCTION within
(s1 geometry, s2 geometry)
 RETURNS BOOLEAN
EXTERNAL NAME
 '/usr/lpp/db2se/gis!shapewithin'
    ...
```

Structured Types as Column Types (2/3)

(5)

```
CREATE TABLE customers (
   cid        INTEGER,
   name       VARCHAR(20),
   income     INTEGER,
   addr       CHAR(20),
   loc        point);

CREATE TABLE stores (
   sid        INTEGER,
   name       VARCHAR(20),
   addr       CHAR(20),
   loc        point,
   zone       polygon);

CREATE TABLE sales (
   sid        INTEGER,
   cid        INTEGER,
   amount     INTEGER);
```

CUSTOMERS

CID	NAME	INCOME	ADDR	LOC

STORES

SID	NAME	ADDR	LOC	ZONE

SALES

SID	CID	AMOUNT

Structured Types as Column Types (3/3)

(6)

“Tell me the all the information I have about each customer who either lives within a stores’ zone or within 100 miles of the store.”

```
SELECT * FROM stores s, customers c
WHERE within(c.loc, s.zone) = 1
      OR distance(c.loc, s.loc) < 100
ORDER BY s.name, c.name;
```

Structured Types as Row Types: Typed Tables

- Structured types can be used to define typed tables
  - Attributes of type become columns of table
  - Plus one column to define ***REF value*** for the row (**object id**)

```
CREATE TYPE real_estate AS
(owner                REF (person),
 price                money,
 rooms                INTEGER,
 size                 DECIMAL(8,2),
 location             address,
 text_description     text,
 front_view_image     bitmap,
 document             doc) NOT FINAL

CREATE TABLE properties OF real_estate
(REF IS oid USER GERNERATED)
```

Reference Types

- Structured types have a corresponding reference type
  - Can be used wherever other types can be used

- Representation
  - **User generated** (REF USING <predefined type>)
  - **System generated** (REF IS SYSTEM GENERATED) : *default*
  - **Derived** from a list of attributes (REF (<list of attributes>)

CREATE TYPE real_estate AS (owner REF (person),...)
NOT FINAL **REF USING INTEGER**

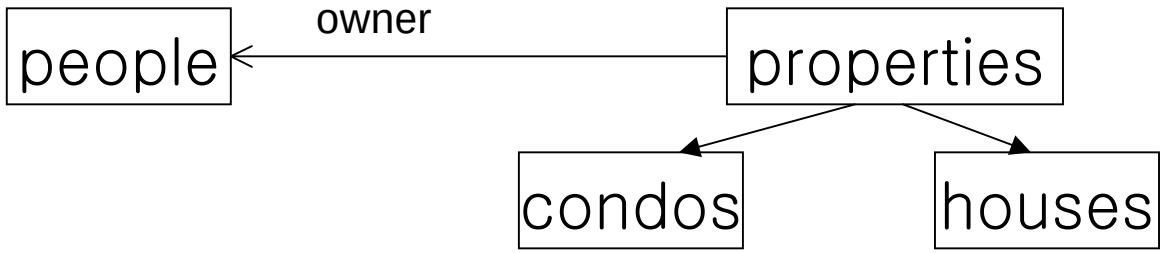
CREATE TYPE person AS (ssn INTEGER, name CHAR(30), ...)
NOT FINAL **REF (ssn)**

Subtables: Table Hierarchies

- Typed tables can have subtables
  - **Inherit** columns, constraints, triggers, ... of the supertable

```
CREATE TYPE person... NOT FINAL
CREATE TYPE real_estate ... NOT FINAL
CREATE TYPE condo UNDER real_estate ... NOT FINAL
CREATE TYPE house UNDER real_estate ... NOT FINAL

CREATE TABLE people OF person (...)
CREATE TABLE properties OF real_estate
CREATE TABLE condos OF condo UNDER properties
CREATE TABLE houses OF house UNDER properties
```

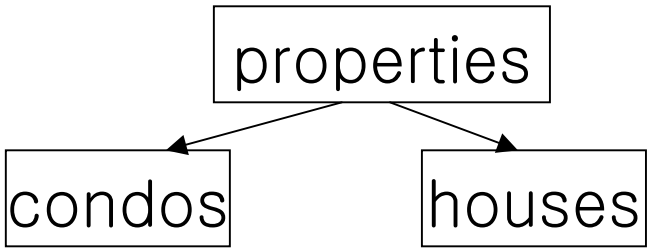


Substitutability

- Queries on table hierarchies range over the rows of every subtable

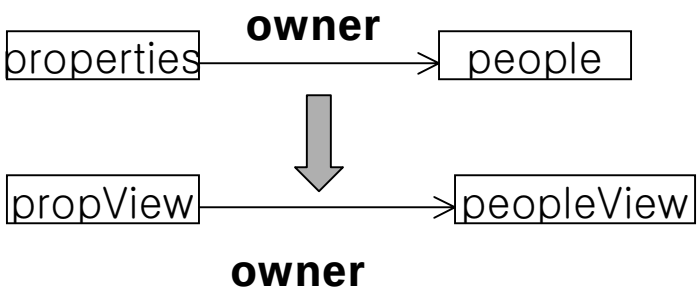
SELECT price, location.city, location.state FROM properties
WHERE contains (text_description, ‘excellent school district’)

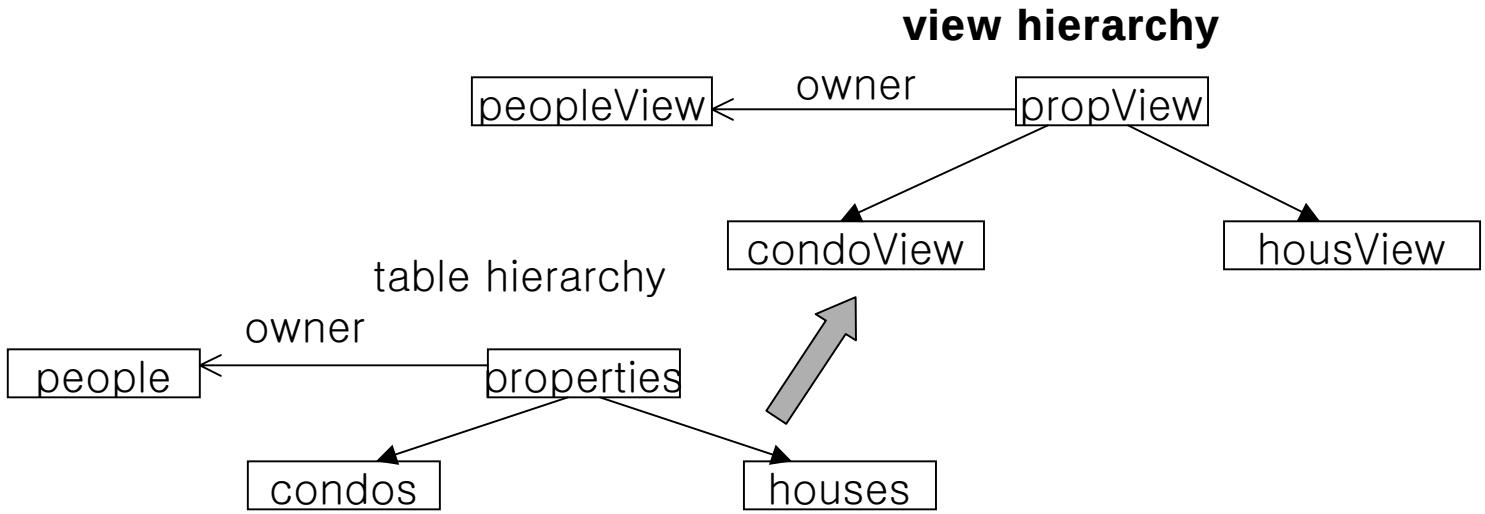
  - Returns properties, condos, and houses



Object Views (1/2)

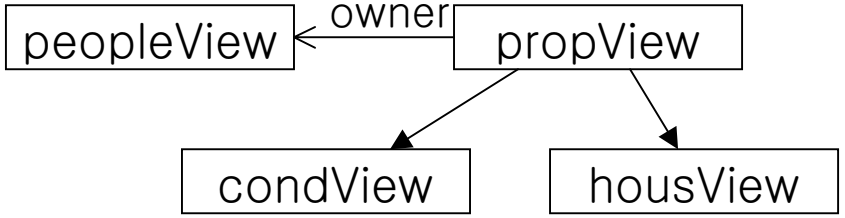
- Views have been extended to support
  - Typed views
  - View hierarchies
  - References on base tables can be mapped to references on views





Object Views (2/2)

```
CREATE TYPE propViewType AS
(owner     REF (person),
 location   address ) NOT FIANL
```




```
CREATE TYPE condViewType UNDER propViewType...
CREATE TYPE housViewType UNDER propViewType ...

CREATE VIEW propView OF propViewType
REF IS propID USER GENERATED
(owner WITH OPTIONS SCOPE peopleView)
AS (SELECT owner, location FROM ONLY (properties))

CREATE VIEW housView OF housViewType UNDER propView
AS (SELECT owner, location FROM ONLY (houses))

CREATE VIEW condView OF condViewType UNDER propView
AS (SELECT owner, location FROM ONLY (condos))
```

Arrays (1/3)

- The only collection type of SQL99
- Why arrays?
  - Tables with collection-valued columns
    - “repeating groups”
    - n1NF tables
  - Heavily used in Standard Type Libraries
    - SQL/MM Full-Text
    - SQL/MM Spatial
- Array characteristics
  - Maximal length vs actual length (like CHARACTER VARYING)
  - Any element type admissible (except array types)
  - Substitutability applies at element level
  - “Arrays anywhere”

Arrays (2/3)

- Tables with array-valued columns

```
CREATE TABLE reports
(id          INTEGER,
 authors     VARCHAR(15) ARRAY[20],
 title       VARCHAR(100),
 abstract    FullText)
```

- Appropriate DML operations

```
INSERT INTO reports(id, authors, title)
VALUES (10, ARRAY ['Date', 'Darwen'], 'A Guide to the SQL Standard')
```

Arrays (3/3)

```
SELECT id, authors[1] AS name FROM reports

SELECT r.id, a.name
FROM reports AS r, UNNEST(r.authors) AS a(name)

SQL TYPE IS point AS LOCATOR pointvar;

EXEC SQL SELECT center INTO :pointvar
FROM circles WHERE ...

EXEC SQL UPDATE circles
SET center = :pointvar
WHERE ...
```

New and Extended Predicates

- Extensions
  - BETWEEN predicate (syntactic sugar)
  - LIKE predicate (BLOB support)
  - Matching rows : SIMPLE match (syntactic salt)

- New predicates
  - DISTINCT predicate (no simple match)
  - SIMILAR predicate (GREP facilities)
  - Type predicate (test dynamic types)

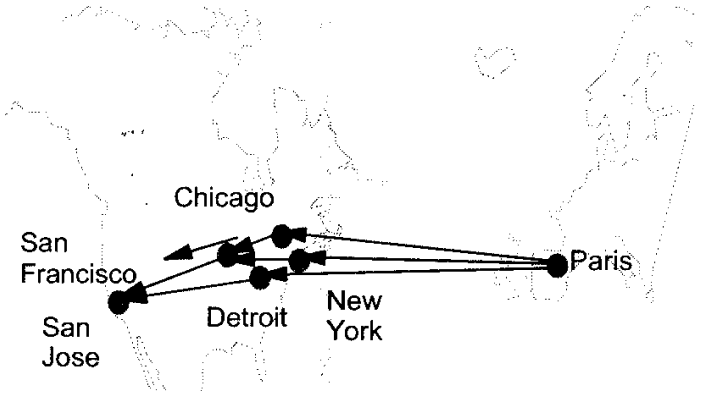
Recursive SQL

"Find the cheapest flight from Paris to San Jose or San Francisco."

```
WITH RECURSIVE Reachable_From (Source, Destin, Total_Cost) AS
 ( SELECT Source, Destination, Cost
   FROM Flights
   WHERE Source = 'Paris'
 UNION
   SELECT  in.Source, out.Destination, in.Total_Cost+out.Cost
   FROM Reachable_From in, Flights out
   WHERE in.Destin = out.Source
 )
SELECT Source, Destin, MIN(Total_Cost)
FROM Reachable_From
WHERE Destin in ('San Jose', 'San Francisco')
GROUP BY Source, Destin
```

Flights

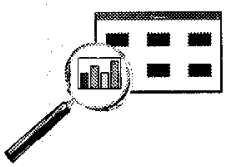
Source	Destination	Carrier	Cost
Paris	Detroit	KAL	7
Paris	New York	KAL	6
Paris	Boston	AA	8
New York	Chicago	AA	2
Boston	Chicago	AA	6
Detroit	San Jose	AA	4
Chicago	San Jose	AA	2

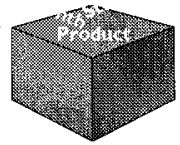


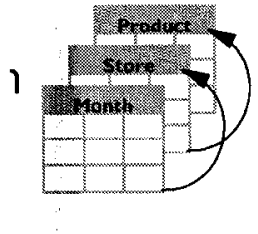
SOURCE	DESTIN	MIN (Total_ Cost)
Paris	San Fran	14
Paris	San Jose	10

SQL99 OLAP SQL Extensions

- Extension to GROUP BY clause
- Produces “super aggregate” rows
- ROLLUP equivalent to “control breaks”
- CUBE equivalent to “cross tabulation”
- GROUPING SETS equivalent to multiple GROUP BYs
- Provides “data cube” collection capability
  - Often used with data visualization tool

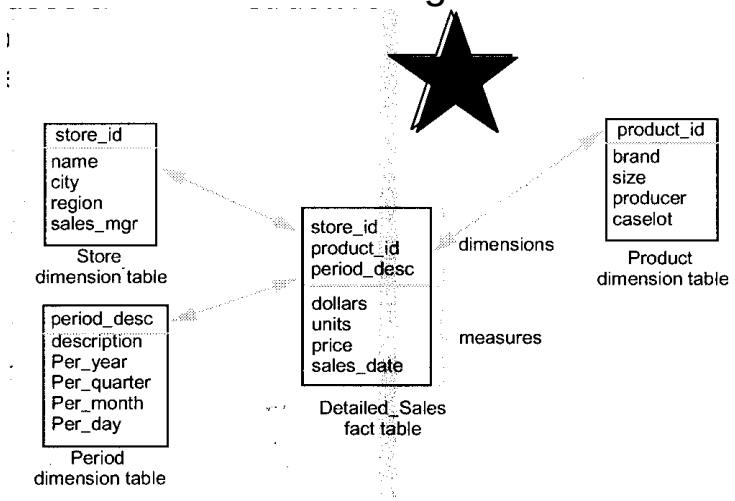






OLAP Schema

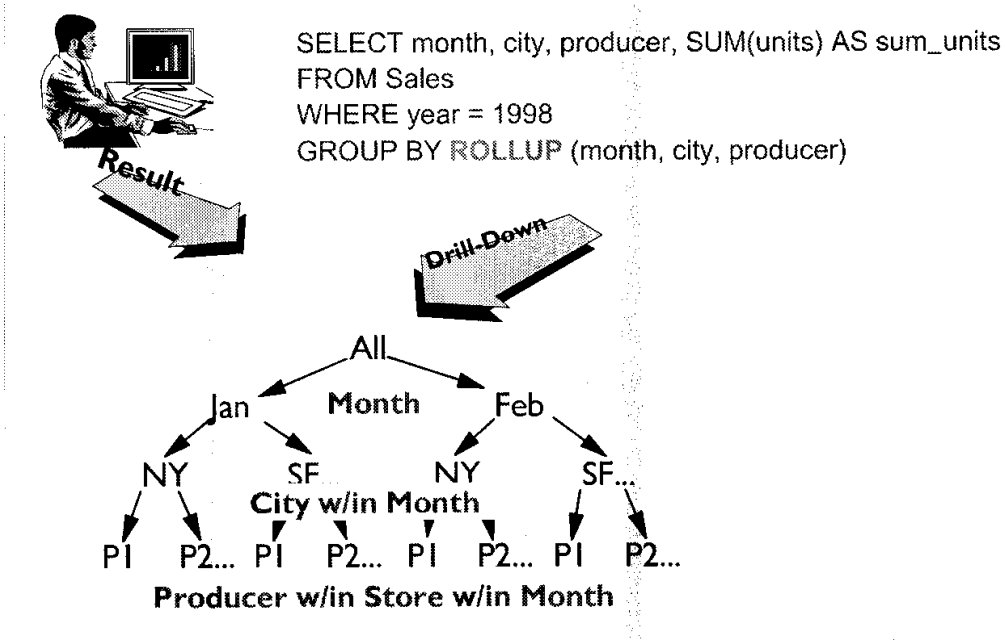
- Typically used a “STAR” structure
  - Dimension tables tend to be small
  - Fact table tends to be huge



CREATE VIEW Sales AS
(SELECT ds.*, YEAR(sales_date) AS year, MONTH(sales_date) AS month, DAY(sales_date) AS day
FROM(Detailed_Sales NATURAL JOIN Store NATURAL JOIN Product NATURAL JOIN Period) ds

ROLLUP (1/2)

- Extends grouping semantics to produce “subtotal” rows
  - Produces “regular” grouped rows
  - Produces same groupings reapplied down to grand total



ROLLUP (2/2)

- Find the total sales per region and sales manager during each month of 1996, with subtotals for each month, and concluding with the grand total:

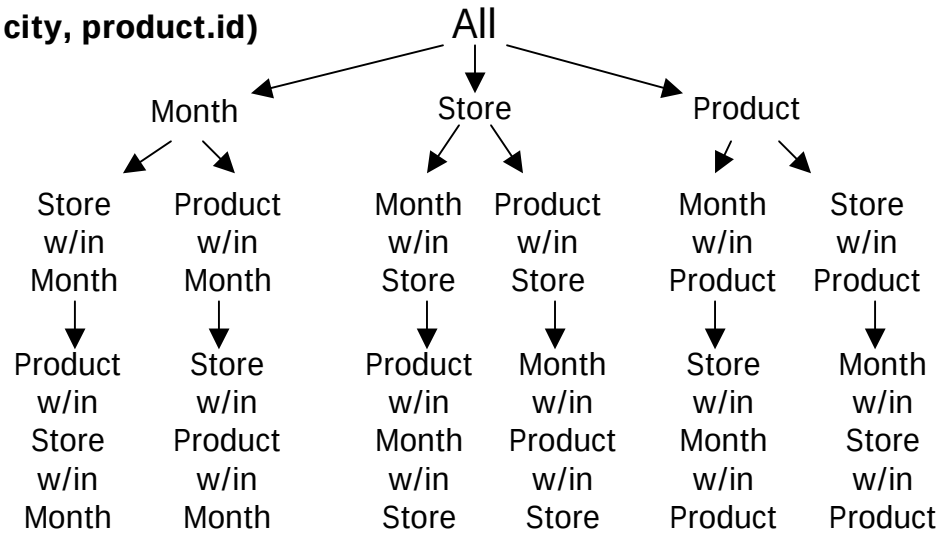
SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year=1996
GROUP BY ROLLUP(month, region, sales_mgr)

MONTH	REGION	SALES_MGR	SUM(price)
April	Central	Chow	25000
April	Central	Smith	15000
April	Central	-	40000
April	NorthWest	Smith	15000
April	NorthWest	-	15000
April	-	-	55000
May	Central	Chow	25000
May	Central	-	25000
May	NorthWest	Smith	15000
May	NorthWest	-	15000
May	-	-	40000
-	-	-	95000

CUBE

- Further extends grouping semantics to produce multidimensional grouping and "subtotal" rows
  - Superset of ROLLUP
  - Produces "regular" grouped rows
  - Produces same groupings reapplied down to grand total
  - Produces additional groupings on all variants of the CUBE clause

```
SELECT month, city, product_id, SUM(units)
FROM Sales
WHERE year=1998
GROUP BY CUBE (month, city, product.id)
```



SELECT...GROUP BY CUBE

SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year=1996
GROUP BY CUBE(month, region, sales_mgr)

MONTH	REGION	SALES_MGR	SUM(price)
April	Central	Chow	25000
April	Central	Smith	15000
April	Central	-	40000
April	NorthWest	Smith	15000
April	NorthWest	-	15000
April	-	Chow	25000
April	-	Smith	30000
April	-	-	55000
May	Central	Chow	25000
May	Central	-	25000
May	NorthWest	Smith	15000
May	NorthWest	-	15000
May	-	Chow	25000
May	-	Smith	15000
May	-	-	40000
-	Central	Chow	50000
-	Central	Smith	15000
-	Central	-	65000
-	NorthWest	Smith	30000
-	NorthWest	-	30000
-	-	Chow	50000
-	-	Smith	45000
-	-	-	95000

GROUPING SETS

- Multiple “groupings” in a single pass
  - Used in conjunction with usual aggregation(MAX, MIN, SUM, AVG, COUNT, ...)
  - Allows multiple groups e.g. (month, region) and (month, sales_mgr)
  - Result can be further restricted via HAVING clause

Find the total sales during each month of 1996, per region and per sales manager:

```
SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year = 1996
GROUP BY GROUPING SETS((month, region),(month, sales_mgr))
```

MONTH	REGION	SALES_MPR	SUM(SALES)
April	Central	-	40000
April	NorthWest	-	15000
April	-	Chow	25000
April	-	Smith	30000
May	Central	-	25000
May	NorthWest	-	15000
May	-	Chow	25000
May	-	Smith	15000

Generating Grand Total Rows

- Special syntax available to include a “grand total” row in the result
  - Grand totals are generated implicitly with ROLLUP and CUBE operations
  - Syntax allows grand totals to be generated without additional aggregates

Get total sales by month, region, and sales manager and also the overall total sales:

```
SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year=1996
GROUP BY GROUPING SETS((month, region), ())
```

MONTH	REGION	SALES_MPR	SUM(SALES)
April	Central	Chow	25000
April	Central	Smith	15000
April	NorthWest	Smith	15000
May	Central	Chow	25000
May	NorthWest	Smith	15000
-	-	-	95000

The GROUPING Function

- New column function
  - Allows detection of rows that were generated during the execution of CUBE and ROLLUP.

    i.e., generated nulls to be distinguished from naturally occurring ones

- Run a rollup, and flag the generated rows...

```
SELECT month, region, sales_mgr, SUM(price), GROUPING(sales_mgr)
FROM Sales
WHERE year=1996
GROUP BY ROLLUP (month, region, sales_mgr)
```

Result...

SELECT month, region, sales_mgr, SUM(price), GROUPING(sales_mgr) AS GROUPED
FROM Sales
WHERE year=1996
GROUP BY ROLLUP (month, region, sales_mgr)

MONTH	REGION	SALES_MPR	SUM(SALES)	GROUPED
April	Central	Chow	25000	0
April	Central	Smith	15000	0
April	Central	-	40000	1
April	NorthWest	Smith	15000	0
April	NorthWest	-	15000	1
April	-	-	55000	1
May	Central	Chow	25000	0
May	Central	-	25000	1
May	NorthWest	Smith	15000	0
May	NorthWest	-	15000	1
May	-	-	40000	1
-	-	-	95000	1

Selecting Nongrouped Columns

- Nongrouped columns can sometimes to selected based on functional dependencies:

```
SELECT e.deptno, d.location, AVG(e.salary) AS average
        FROM Emp e, Dept d
        WHERE e.deptno=d.deptno
        GROUP BY e.deptno
```

Legal

e.deptno determines d.deptno (equals in WHERE clause),
And d.deptno determines d.location (deptno is PK of Dept);
Therefore, d.deptno and d.location are consistent within any group. This is functional dependency analysis in action.

ILLEGAL

```
SELECT e.deptno, e.name, AVG(e.salary) AS Average
  FROM Emp e, Dept d
  WHERE e.deptno=d.deptno
  GROUP BY e.deptno
```

Cursors (1/2)

- In SQL89, FETCH always retrieves “next” row
- Scrollable cursors (in SQL92)
  - Allows both forward and backward movement of the cursor
  - Allows skipping of rows

```
EXEC SLQ DECLARE c SCROLL CURSOR FOR SELECT…;
EXEC SQL OPEN c;
EXEC SLQ FETCH ABSOLUTE 10 FROM c INTO …;
EXEC SQL FETCH RELATIVE 32 FROM c INTO …;
EXEC SQL FETCH PRIOR FROM c INTO …;
```

- FETCH options are:
  - FIRST
  - LAST
  - NEXT
  - PRIOR
  - ABSOLUTE n
  - RELATIVE n

Cursors (2/2)

- Cursor sensitivity (in SQL99)
  - SENSITIVITY : changes are visible
  - INSENSITIVITY : changes are invisible

EXEC SQL DECLARE CURSOR SENSITIVITY FOR
SELECT * FROM People;

- Holdable cursors (in SQL99)
  - remain open when a transaction is committed
  - closed and destroyed when
    - transaction is rolled back
    - session is terminated

Transaction Management

- New statements for
  - Explicitly starting TXs
    - Also sets TX characteristics
  - Establishing savepoints
  - Destroying savepoints

```
INSERT INTO People (Lname, Fname, Nick)
    VALUES ('Doe', 'John', 'Hans');
SAVEPOINT SP1

UPDATE People SET Nick='Jean'
WHERE Lname='Doe'

ROLLBACK TO SAVEPOINT SP1
```

Connections

- Associations between an SQL-client and an SQL-server
- There is an SQL-session associated with each connection

```
env="IBMSYS";
connect="STLconnection";
user="Todd";
EXEC SQL CONNECT TO :env AS :connect USER :user
   ...
EXEC SQL COMMIT;
env="IBMSYS2";
EXEC SQL SET CONNECTION :env;
```

- Transactions that affect more than one SQL-environment do not have to be supported.

Module Language

- **Module definition**

```
module read
Language C
Authorization reader
DECLARE people CURSOR FOR
        SELECT last, first
        FROM   hobbies
        WHERE hobbies=:h
PROCEDURE open_people (SQLSTATE, :h CHAR(5));
  OPEN people;
PROCEDURE fetch_people (SQLSTATE, :last CHAR(20), :first CHAR(20));
   FETCH people INTO :last,  :first;
PROCEDURE close_people SQLSTATE;
   CLOSE people;
```

- **Application program**

```
main()
{
char SQLSTATE[6];
char last[21], first[21];
OPEN_PEOPLE(SQLSTATE, “travel”);
    while…
    FETCH_PEOPLE(SQLSTATE, last, first);
}
```

SQL99 PSM (1/3)

- Procedural Extensions
  - Improve performance in a centralized and client/server environments
    - Multiple SQL statements in a single EXEC SQL
    - Multi-statement procedures, functions, and methods
  - Gives great power to DBMS
    - Several, new control statements (procedural language extension) (begin/end block, assignment, call, case, if, loop, for, signal/resignal, variables, exception handling)
  - SQL-only implementation of complex functions
    - Without worrying about security (“firewall”)
    - Without worrying about performance (“local call”)
  - SQL-only implementation of class libraries

SQL99 PSM (2/3)

- Includes two major aspects
  - Procedural extensions (control statements) - feature from block-structured languages, including exception handling
  - SQL-server modules - groups of SQL-invoked routines managed as named, persistent objects
- C program with embedded SQL statements

```
void main
{
 EXEC SQL INSERT INTO employee
VALUES (…);
EXEC SQL INSERT INTO department
VALUES (…);
}
```

- Using PSM-96 procedural extensions

```
void main
{
EXEC SQL
BEGIN
 INSERT INTO employee VALUSE (…);
 INSERT INTO department VALUSE (…);
END;
}
```

SQL99 PSM (3/3)

- If we create a SQL procedure first:

```
CREATE PROCEDURE proc1 ()
{
BEGIN
INSERT INTO employee VALUSE (…);
INSERT INTO department VALUSE (…);
END;
}
```

- Then the embedded program can be written as

```
void main
{
EXEC SQL CALL proc1 ();
}
```

SQL Procedural Language Extensions

Compound statement	**BEGIN … END;**
SQL variable declaration	**DECLARE var CHAR(6);**
If statement	**If subject (var< >'urgent') THEN … ELSE …;**
Case statement	**Case subject (var) WHEN 'SQL' THEN … WHEN …;**
Loop statement	**Loop <SQL statement list> END LOOP;**
While statement	**While i<100 DO … END WHILE;**
Repeat statement	**REPEAT … UNTIL i<100 END REPEAT;**
For statement	**For result AS … DO … END FOR;**
Leave statement	**LEAVE …;**
Return statement	**RETURN 'urgent';**
Call statement	**CALL procedure_x (1,3,5);**
Assignment statement	**SET x='abc';**
Signal/resignal statement	**SIGNAL division_by_zero**

SQL99 Bindings

- Embedded SQL
  - An embedded host language program is transformed into a pure host language program and an “abstract” SQL module

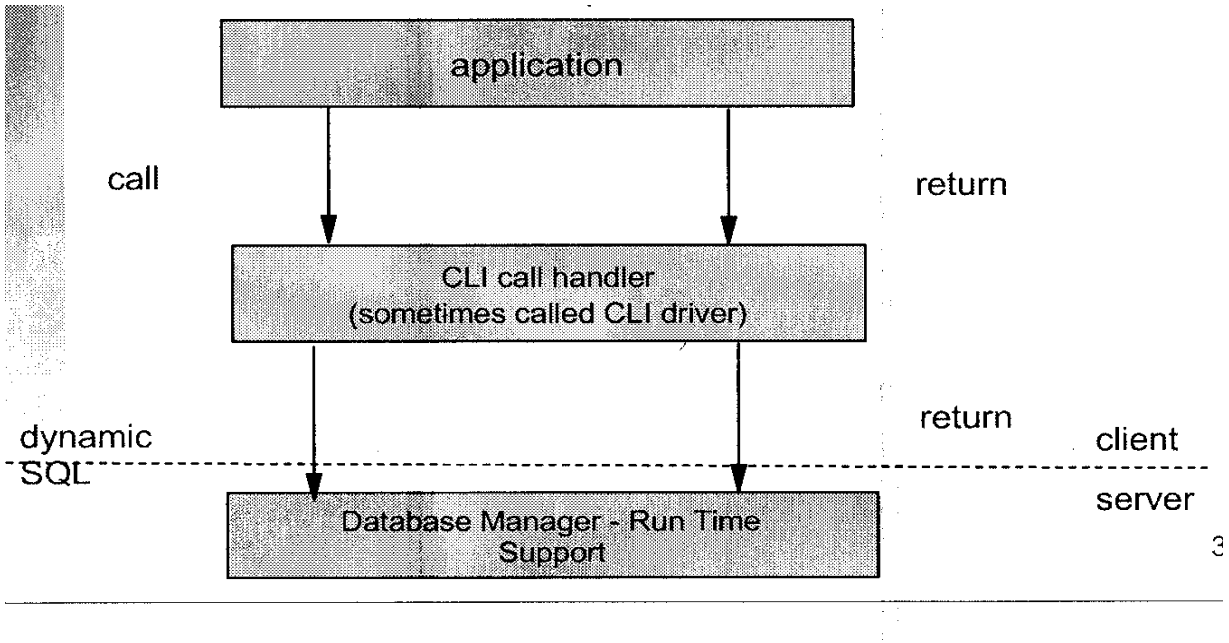
- Dynamic SQL
  - When the tables, columns, or predicates are not known when the application is written

```
EXEC SQL PREPARE stmt FROM ...;
EXEC SQL EXECUTE stmt ...;
    ...
```

- Direct SQL
  - Implementation-defined mechanism for executing direct SQL statements

Call Level Interface (CLI)

- Functional interface to database
- Consists of over 40 routine specifications
  - Control connections to SQL-servers
  - Allocate and deallocate resources
  - Execute SQL statements
  - Control transaction termination
  - Obtain information about the implementation



SQL99 CLI (1/2)

- SQL99 data type support
  - BOOLEAN
  - LOBs with optional locators and helper routines (GetLength, GetPosition, GetSubstr)
  - UDTs with locators and transformation functions
  - Arrays with locators only
  - Reference types with table scope
  - Can retrieve/store unnamed ROW types

SQL99 CLI (2/2)

- CLI descriptor model aligned with ODBC 3.x (defaults, Get/Set restrictions, etc.)
- JDBC 2.0 support for user-defined types
- Multi-row fetch ODBC
- Catalog routines aligned with SQL99 and ODBC
- Parallel result set processing after CALL statement
- SQL99 savepoints
- General SQL99 alignment(roles, user-defined casts, SQLSTATEs, etc.)

Web References

- ISO (International Organization for Standardization)
  - **http://www.iso.ch**
- JTC1 SC32 - Data Management and Interchange
  - **http://bwonotes5.wdc.pnl.gov/SC32/JTC1SC32.nsf**
- ANSI (American National Standards Institute)
  - **http://web.ansi.org**
- NCITS (National Committee for Informational Technology Standards)
  - **http://www.ncits.org/**
- KISI (Korean Industrial Standards Institute)
  - **http://www.kisi.or.kr**
- KDPC (Korea Database Promotion Center)
  - **http://www.dpc.or.kr**