



Informix 11.5 Bootcamp Application Development Overview

Information Management Partner Technologies

Agenda

- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- Informix Extensibility Features
- Appendix

Agenda

- IBM Common Client with DRDA Support
- Informix Application Programming and Web Interfaces
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- Informix Extensibility Features
- Appendix

IBM Common Client

- Set of development tools which use the *DRDA* protocol to communicate with IBM Data Servers
 - IBM Data Server Driver for JDBC and SQLJ
 - IBM Data Server Provider for .NET for Informix
 - IBM Data Server Client & Runtime Client
 - PHP: PDO_IBM
 - Ruby: Version IBM_DB-0.8
- Distributed Relational Database Architecture™ (DRDA®)
 - Set of protocols that coordinates communication between applications and database systems on disparate platforms
 - Based on a database interoperability standard from [The Open Group](#)

Informix Support for DRDA - API

- CSDK only supports the *SQLI* protocol*
 - Prior to Informix 10.x, this was the only supported way for a client to communicate with Informix
- DRDA support has been added
 - Using IBM Common Client
 - Installer automatically configures DRDA protocol

Features Supporting DRDA

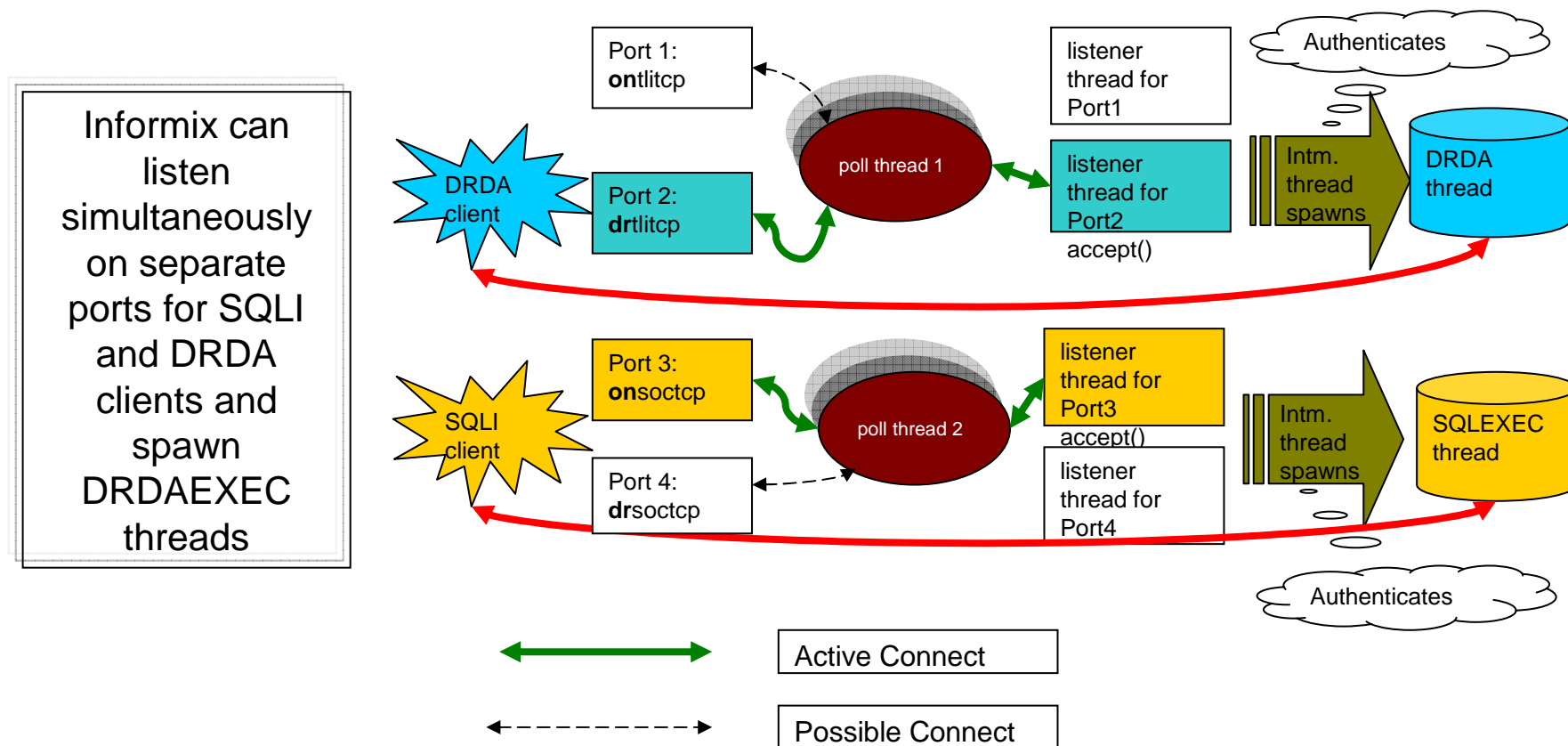
- Encryption using Secure Socket Layer
- Informix Connection Manager
 - However, Enterprise Replication (ER), High-Availability Data Replication (HDR), and certain utilities, such as DB-Access, still require *SQLI* connections

Benefits

- Supports a whole new class of tools and applications
- Common Clients reduces cost for development and faster deployment
- The Informix 11.5 installer allows for configuring a database server alias and a port for clients that use the DRDA protocol
- Allows access to distributed data

* starting CSDK 3.50.xC6 the DRDA driver is bundled with CSDK

Informix DRDA Support - Net Architecture



onstat -g ath

***41 f7d2920 ec166d8 1 cond wait netnorm 1cpu drdaexec**

Informix setup of DRDA is Simple!

DRDA configured as **DBSERVERALISES**
- separate protocol *drsoctcp* / *drtlitcp*

Example ONCONFIG file:

```
DBSERVERNAME          marsh_1110
DBSERVERALIASES       marsh_1110_drda
NETTYPE               onsoctcp,1,,NET # sqli
                     drsoctcp,1,,NET # drda
```

Example SQLHOSTS entry:

HOST	OPTIONS	PROTOCOL	PORT
marsh_1110	localhost	onsoctcp	1528
marsh_1110_drda	localhost	drsoctcp	1529

Agenda

- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- Informix Extensibility Features
- Appendix

Informix Application Development Technologies



- Key Database Technologies

- SQL / SQL Procedures
- XML
- SOA / Web Services



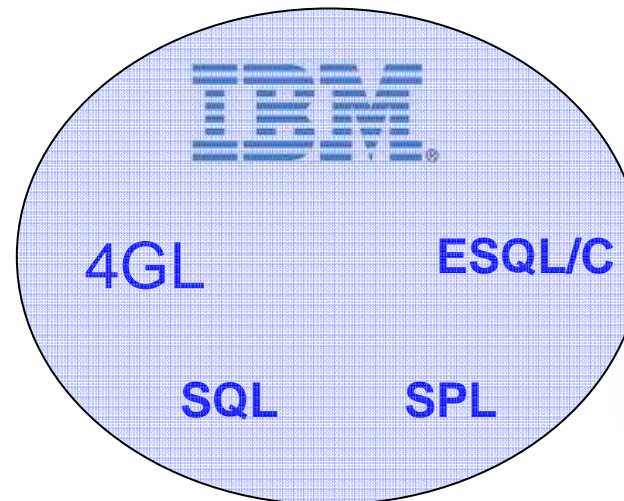
- Developer communities

- C/C++
- Java (JDBC / SQLJ)
- .NET (C#, VB .NET)
- EGL



- **Open Source**

- PHP/Zend FW
- Ruby/Rails
- Perl
- Python/Django



Structured Query Language (SQL)

- Widely used relational database computer language
- Great for managing data in relational database management systems (RDBMS)
 - Data query
 - Data update
 - Schema creation and modification
 - Data access control
- Standard language used to store, retrieve, and manipulate the data in relational databases, such as Informix
- Informix SQL is compatible with ANSI SQL
 - However, there are some Informix-specific extensions

Informix 4GL (I4GL)

- Informix 4GL is a well established structured development language ideal for most business requirements
- Native support for SQL statements
- Easy-to-use Data Entry and Reporting functions
- Can be deployed as either pseudo-code (P code) or C code applications
 - P code primarily is used for development
- Connectivity to Informix database servers only
 - Unless an Informix Gateway server is added
- Small footprint delivers great runtime performance
- No Windows support

New Investments in 4GL



4GL 7.50.xC1 June 2008

- Support System Curses Library compliance with UTF-8
- Client Required Priority Defect Fixes

4GL 7.50.xC2 December 2008

- SOA Web Services Support - Deployment of Web Services (Linux Only)
- Client Required Priority Defect Fixes

4GL 7.50.xC3 June 2009

- SOA Web Services Support - Consumption of Web Services (Linux Only)
- Client Required Priority Defect Fixes

4GL 7.x 2H 2009*

- SOA Web Services Support on All Major Platforms
- MAC OS Support*
- Client Required Priority Defect Fixes

4GL Interim Releases Every 12 – 18 Months 2010 and beyond *

- ISQL / Ace Limits
- 4GL GUI
- High value Feature Requests from Customers and Partners
- Enhanced Web Services functions
- Certification on new platform releases

* Subject to change

The information on the new product is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information on the new product is for informational purposes only and may not be incorporated into any contract. The information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at our sole discretion.

ESQL/C

- SQL application programming interface (API) that embeds Structured Query Language (SQL) statements directly into a C program
- Consists of both ESQL and C Preprocessor directives and Language statements
- Includes the following software components:
 - ESQL/C libraries of C functions which provide Informix access
 - ESQL/C header files, which provide definitions for the data structures, constants, and macros useful to the Informix ESQL/C program
 - The **esql** preprocessor, which processes the Informix ESQL/C source code and SQL statements to create a C source file that it passes to the C compiler
 - The **finderr** utility that enable you to obtain information about IBM Informix-specific error messages

PERL



- One of the most popular scripting languages available today
- Used widely in all forms of development
 - Web based applications
 - System administration automation on UNIX and Linux systems
 - Extraction and conversion of data due to its
- Provides a rich offering of modules for almost every task
- The Comprehensive Perl Archive (CPAN)
- The DBI (DataBase Interface) is the well established standard for accessing databases from Perl
 - Defines a set of abstract methods which are implemented by the underlying DataBase Driver (DBD)
 - Driver for accessing an Informix database is called **DBD::Informix**

JAVA & JDBC



- Very popular dynamic object oriented programming language originally developed by Sun Microsystems and released in 1995
 - Applications are compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture
- Prior to Informix 11.10, the **IBM Informix JDBC driver** provided Java application connectivity to Informix
 - Informix specific driver only
 - Limited integration with other IM products and tools
- Beginning with Informix 11.10, the new **IBM Data Server Driver for JDBC and SQLJ (JCC)** is available
 - Provides high performance Java connectivity to IBM servers including Informix
 - Provides JDBC (3 & 4) or SQLJ access to data
 - Compact (2.5MB footprint) with no installation required
 - Provides pure-Java or Type 4 connectivity to Informix
 - Supports many Informix features including MACH 11
 - All new enhancements for all databases will be worked into JCC

Applications can use *either* driver to connect to Informix

Informix and PHP



- Popular open source, platform independent scripting language for building dynamic data-driven Web applications
- PHP 5 now includes advanced features such as Web Services, XML and object-oriented constructs
- Coupling it with the reliability and performance of Informix 11 makes it an obvious data server choice for PHP Developers
- Informix support for PHP is available using four PHP drivers/extensions
 - Unified ODBC (ext/odbc)
 - PHP driver for Informix (based on esql/c) (PHP 5.0.3 or newer)
 - PHP Data Objects (Informix specific): **PDO_INFORMIX** (SQLI) + **PDO_ODBC**
 - Common client PHP driver (**PDO_IBM**) supported by Informix using DRDA

IBM Zend Core and PHP



- Zend Core for IBM (ZCI) is a seamless out-of-the-box, easy to install, and supported PHP development and production environment
- One stop shop for PHP installation, rapid development, deployment and production support
- Comes with all the necessary PHP drivers, third party libraries, samples, etc.
- ZCI 2.0 packages **PDO_IBM** and **PDO_INFORMIX** with all the other necessary components
- Zend Core for IBM is FREE!

www.ibm.com/software/data/info/zendcore

PYTHON



- Dynamic Object Oriented Programming language
- Scripts have a clean structure and are easy to read and maintain
- The standard for accessing databases from Python is the DB-API
- Common Python DBI driver supports both DB2 (LUW, zOS, i5) and Informix
 - Implementation of Python Database API Specification v2.0
 - **InformixDB** is a Python extension module that provides a way to connect to an Informix database via Python's Database API 2.0

Ruby on Rails



- Ruby
 - An object-oriented open source interpreted programming language
 - Inspired by Smalltalk, sharing features with Python, Lisp, Dylan and CLU
- Rails (a.k.a RoR)
 - A full stack Web framework written in Ruby for developing database-backed web applications
 - Web development made easy through “Convention over configuration” and “Don’t Repeat Yourself” principles
- The **IBM_DB** Ruby driver and Rails adapter allows Ruby applications to access Informix servers
- Connects to Informix 11.x via the DRDA protocol

Microsoft .NET



- Software framework that runs on Microsoft Windows operating systems
 - Includes a large software library and a virtual machine that manages the execution of programs written specifically for the framework
- Two .NET providers from IBM that lets .NET applications access and manipulate data in IBM Informix databases
 - **IBM Informix .NET Provider** shipped with the Informix CSDK product
 - Provides access to Informix V7 and later servers
 - Supported, but no future enhancements
 - **IBM Common Informix .NET Provider** shipped with the IBM Data Server Driver set of products
 - Provides more functionality and better portability
 - Part of the next version of IBM Data Server .NET clients
 - Greater Visual Studio support
 - Utilizes the DRDA interface to communicate with Informix V11.10 later
 - Recommended more for new Informix deployments

Both providers
can co-exist in
a *single box*

Web 2.0 Applications

- Web-based software which is continually collaboratively updated
 - Software gets more useful the more people who consume and remix it
- Examples
 - RIA - Rich Internet Applications
 - SOA - Service Oriented Architecture , SaaS
 - Social Web
- Data Studio
 - Generate Web services which include SQL statements via Point-and-Click SOA
 - No Programming required
 - No Code Generation
 - Eclipse Platform
 - Support for Informix



WIKIPEDIA



Agenda

- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- Informix Extensibility Features
- Appendix

Triggers

- Database object that automatically performs one or more database operations when a certain database event occurs
- The database event can be an INSERT, UPDATE, DELETE or SELECT operation on a table or a view
- The operations that a trigger performs can be INSERT, UPDATE, DELETE, EXECUTE PROCEDURE or EXECUTE FUNCTION operations
- There can be multiple INSERT, UPDATE, SELECT or DELETE triggers created on the same table or view

Triggers – Components

- Trigger name

- Identifies the trigger
- Must be unique within the database

- Trigger event

- Database statement that activates the trigger
- Event can be INSERT, UPDATE, DELETE or SELECT statement
- For UPDATE or SELECT events, a single or multiple columns can be chosen on the table
- For INSERT or DELETE events, any insert or delete on the table activates the trigger
- A trigger is created on a local table or a view

- Trigger action

- SQL statements that are executed **when** the trigger event occurs
- 3 types of trigger actions:
 - BEFORE trigger action is executed before the triggering event occurs
 - AFTER trigger action is executed after the triggering event occurs
 - FOR EACH ROW trigger action is executed for each row that is affected by the triggering event
- Actions can be an INSERT, UPDATE, DELETE, EXECUTE PROCEDURE or EXECUTE FUNCTION statements

Triggers – Example

```
CREATE TRIGGER update_quantity  
  
UPDATE OF quantity ON items  
  
BEFORE(EXECUTE PROCEDURE upd_items_p1)
```

- The above example creates an UPDATE trigger with a name **update_quantity** on the **items** table
- The trigger will be activated when the column quantity gets updated
- When this trigger gets activated, the stored procedure **upd_items_p1** will be executed **before** the column quantity is updated

Triggers – REFERENCING Clause

- Can only be used with FOR EACH ROW trigger actions
- Allows the old and/or new values of the column to be accessed
- Example
 - Create a trigger to execute only if the condition in the WHEN clause is true

```
CREATE TRIGGER up_price
  UPDATE OF unit_price ON stock
  REFERENCING OLD AS pre NEW AS post
  FOR EACH ROW WHEN(post.unit_price > pre.unit_price * 2)
  (INSERT INTO warn_tab
    VALUES(pre.stock_num, pre.manu_code, pre.unit_price,
      post.unit_price, CURRENT));
```

Triggers - INSTEAD OF Clause

- Created on a **view** instead of a table
- Ignores the trigger event and executes the trigger action
- To insert, update, or delete rows in the base tables of a view, you can define an INSTEAD OF trigger

```
CREATE TRIGGER manager_info_update  
  
INSTEAD OF UPDATE ON manager_info_view  
  
REFERENCING NEW AS n  
  
FOR EACH ROW (EXECUTE PROCEDURE updtab (n.empno, n.empname, n.deptno,));
```

Triggers – Trigger Routines – Informix 11.50

- Special stored procedure for trigger usage only
- Can *only* be used with FOR EACH ROW trigger actions
- Allows the old and/or new values of the column to be accessed within the procedure body
- Uses Boolean operators that identifies the trigger type
- Can only be executed as a trigger action using “WITH TRIGGER REFERENCES” clause

```
CREATE TRIGGER update_quantity  
UPDATE OF quantity ON items  
FOR EACH ROW (  
EXECUTE PROCEDURE trig_prog() WITH TRIGGER REFERENCES)
```

```
CREATE PROCEDURE trig_proc()  
REFERENCING OLD as pre NEW as post for items;  
  
if (INSERTING) then  
    insert into log_records values(post.quantity,post.quantity);  
end if  
if (UPDATING) then  
    insert into log_records values(pre.quantity,post.quantity);  
end if  
if (SELECTING) then  
    insert into log_records values (pre.quantity,pre.quantity);  
end if  
if (DELETING) then  
    delete from log_records;  
end if  
END PROCEDURE;
```

Multiple INSERT, DELETE, UPDATE, and SELECT triggers on a Table – Informix 11.50

```
-- Multiple triggers on the same Insert event on manufact table
create trigger man_ins_t1 insert on manufact
before (insert into manu_operations_summary values ("New Manu"))
for each row(execute procedure man_proc() with trigger references);

create trigger man_ins_t2 insert on manufact referencing NEW as new
for each row (insert into manu_log values
(new.manu_code,new.manu_account,"INSERT"));

--- Multiple triggers on the same Update event on manufact table
create trigger man_upd_t1 update on manufact
Before (insert into manu_operations_summary values ("Manu update"))
for each row (execute procedure man_proc() with trigger references);

create trigger man_upd_t2 update on manufact
referencing OLD as old New as new
for each row (insert into manu_log values
(new.manu_code,new.manu_account,"UPDATE"));
```

- Multiple INSTEAD OF triggers on a view

For triggers of the same DML event type, the order of execution is:

1. BEFORE triggered actions
2. FOR EACH ROW actions
3. AFTER triggered actions

Informix Stored Procedure Language (SPL) Routines

- User-defined functions
- Includes SPL procedure and SPL function
 - Functions return values whereas procedures do not



Benefits

- Runs in database engine, less I/O
- Parsed and optimized when they are created rather than at runtime
- Stored in system catalog table in executable format
- Good for frequently repeated tasks (bypass parsing, validity checking, query optimization)
- Reduces application complexity

Flexible

- Callable in other SQL statements
- Useful in triggers
- Adds flow control to SQL
- Easy to maintain

SPL Routines

- An SPL routine consists of:
 - A CREATE statement
 - A statement block
 - An END statement

Within the statement block, you can use SQL or SPL statements

Procedure

```
CREATE PROCEDURE name (parameter list) SPECIFIC name1
    ... {statement block}
END PROCEDURE;
```

Function

```
CREATE FUNCTION name (parameter list)
    RETURNING list SPECIFIC name1
    ... {statement block}
END FUNCTION;
```

SPL Routines – Examples

```
CREATE PROCEDURE set_status (id INTEGER DEFAULT 0, val CHAR(25))  
    UPDATE invent SET invent.status = val WHERE invent.id = id;  
END PROCEDURE;
```

```
CREATE FUNCTION val_comp (val1 INTEGER, val2 INTEGER)  
    RETURNING INTEGER;  
  
    DEFINE res INTEGER;  
    IF (val1 = val2) THEN  
        res = 0;  
    ELSE  
        res = 1;  
    RETURN res;  
END FUNCTION;
```


User-Defined Routines (UDRs)

- Two kinds:
 - User Defined Functions (UDFs)
 - User Defined Procedures (UDPs)
- Can be internal or external
 - Internal are written in SPL
 - External can be written in C (shared object with 755 permissions) or Java (.JAR file stored in an sbpace)
- UDR Recommendations
 - Should be concise and precise (optimal resource utilization)
 - From a business perspective, return relevant values
 - Include error handling
 - Use a “specific” name or alias for effective administration
 - `drop specific function euro_to_ausdlr`

```
create function routine_name (param_list)  
  returns | returning } typename  
  [ specific specific_name ]  
  [ with (internal | handlesnulls / [ [not] variant]  
    | parallelizable | class="vp_class") ]  
  external name 'full_path_name_of_file'  
  language language [ [not] variant ]  
end function;
```

User-Defined Routines (UDRs) - *Overloading/Parallelize/UDVPs*

- Overloading

- Function overloading occurs when two or more functions have the same name but *different* signatures (# of parameters)

```
create function plus (in_1 dollar,      in_2 aus_dollar) ...  
create function plus (in_1 aus_dollar, in_2 euro) ...  
create function plus (in_1 euro,       in_2 aus_dollar) ...
```

- Parallelizable

- Default behavior of UDRs is single-threaded
- Executed in parallel if "**parallelizable**" keyword is used in the UDR creation statement AND several conditions are met (including PDQPRIORITY > 1)

- User Defined Virtual Processors (UDVPs)

- Recommendation is to use UDVPs for UDRs
- Prevents an "ill-tempered" UDR from affecting critical database operations
 - ONCONFIG file: `VPCLASS fence_vp ,num=2`
 - Dynamically: `onmode -p +2 fence_vp`

Agenda

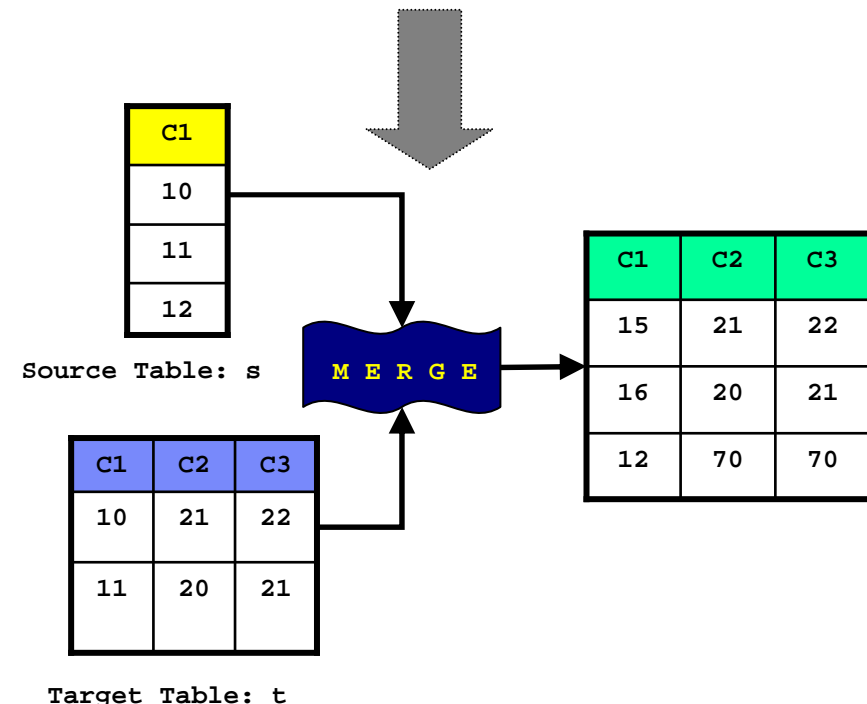
- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- **Informix 11.x Enhancements**
- Informix Extensibility Features
- Appendix

MERGE statement – Load data example

- Transfers data to target table using UPSERT
- Replaces current UPSERT logic implemented in client applications
- Key components of ETL (ELT)* in data warehouse environments
- Merges rows to a target table based on a condition
 - TRUE: Update the target table row
 - FALSE: Insert the target table row

MERGE

```
INTO TARGET t
USING SOURCE s
ON t.c1=s.c1
WHEN MATCHED THEN
    UPDATE SET t.c1=t.c1+5
WHEN NOT MATCHED
    THEN INSERT (t.c1, t.c2, t.c3) values
        (s.c1, 70, 70);
```

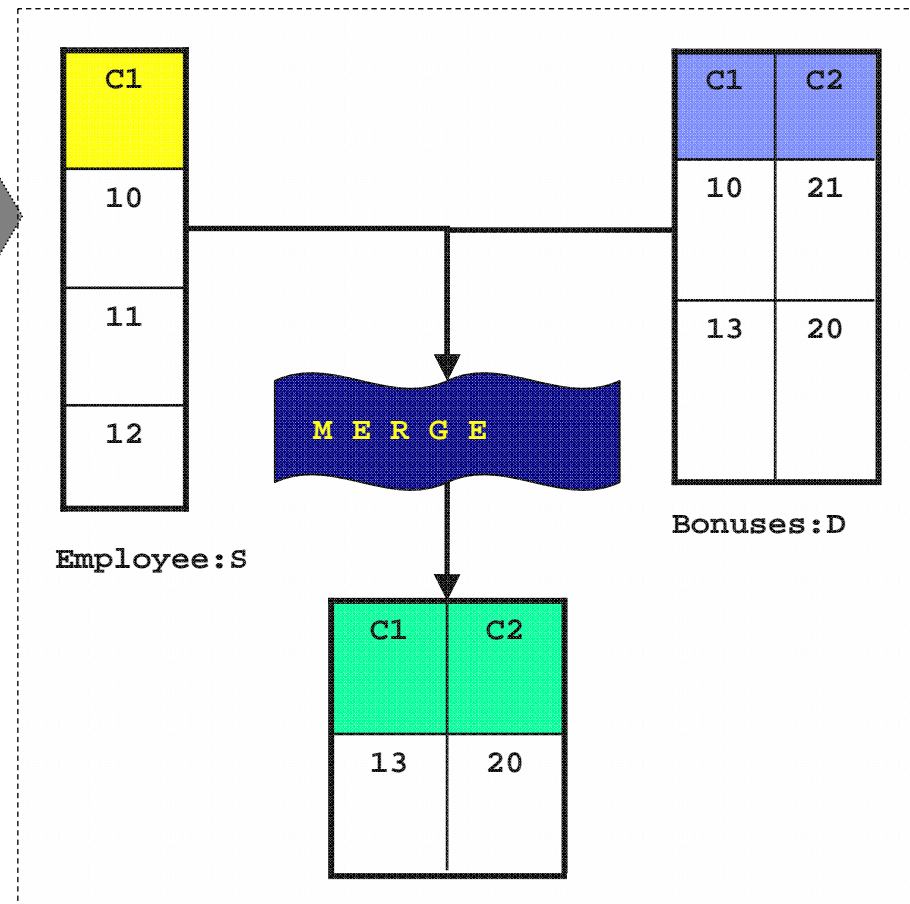


MERGE statement – Delete/Update Joins example

```
MERGE INTO bonuses D
USING employee S
ON D.employee_id = S.employee_id
WHEN MATCHED THEN DELETE;
```

Optional: "WHEN NOT MATCHED"

Source table may be a table expression consisting of joins



* C1 -> employee_id, C2 -> salary

CONNECT BY Clause for Recursive Operations

- Hierarchical clause which sets the conditions for recursive queries on a table in which a hierarchy of parent-child dependencies exist

Seed of recursion

Specifies a search condition that the CONNECT BY clause uses for the first iteration of its recursive actions

```
SELECT name, empid, mgrid
FROM emp
START WITH name = 'Goyal'
CONNECT BY
    PRIOR empid = mgrid
```

Table with hierarchical data

Usually a self-referencing table in which one or more columns acts as a foreign key constraint for another column (or a subset of the columns) in the same table

Condition for recursive

Produces successive intermediate result sets by applying the CONNECT BY search condition until this recursive process terminates when an iteration yields an empty result set

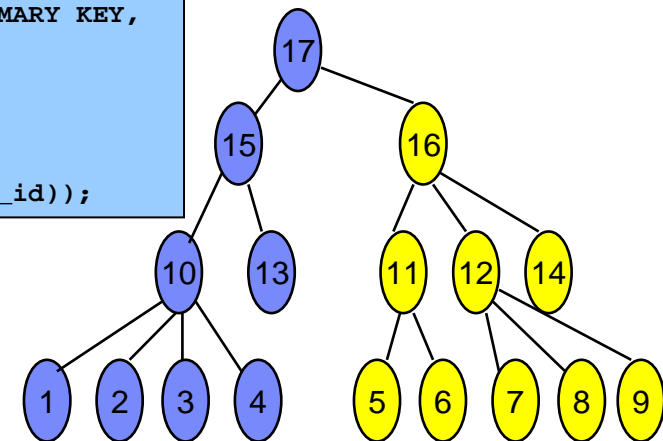
Example - Hierarchical View of the Query and Data

Name	Empid	Mgrid
Jones	1	10
Hall	2	10
Kim	3	10
Lindsay	4	10
McKeoug	5	11
Barnes	6	11
O'Neil	7	12
...		
Urbassek	17	NULL

```
CREATE TABLE emp1
(empid INTEGER NOT NULL PRIMARY KEY,
name VARCHAR(10),
salary DECIMAL(9, 2),
mgrid INTEGER
REFERENCES emp1 (emp_id));
```

Old Syntax

```
SELECT
  name,
  empid,
  mgrid
FROM emp
```



Using New Syntax

```
SELECT name, empid, mgrid
FROM emp
START WITH name = 'Goyal'
CONNECT BY PRIOR empid = mgrid
```

Name	Empid	Mgrid
Goyal	16	17
Zander	11	16
McKeough	5	11
Barnes	6	11
Henry	12	16
O'Neil	7	12
Smith	8	12
Shoeman	9	12
Scott	14	16

Dynamic SQL in Stored Procedures

- Statements can now be dynamically constructed and executed
- New Dynamic SQL Statement in SPL -
EXECUTE IMMEDIATE

```
CREATE PROCEDURE MYPROC()  
    RETURNING INT;  
    DEFINE A0 VARCHAR(30);  
    DEFINE A1 VARCHAR(5);  
    DEFINE A2 INT;  
    DEFINE A3 VARCHAR(60);  
    DEFINE A4 INT;  
    LET A0 = "INSERT INTO DYN_TAB VALUES ("  
    LET A1 = ")";  
    FOR A2 = 1 TO 100  
        LET A3 = A0 || A2 || A1;  
        EXECUTE IMMEDIATE A3 ;  
    END FOR;  
    SELECT COUNT(DISTINCT C1) INTO A4 FROM T1;  
    RETURN A4;  
END PROCEDURE;  
  
-- should return 100 as 100 unique values got  
-- inserted by the EXECUTE IMMEDIATE in loop  
EXECUTE PROCEDURE MYPROC();
```


XML Publishing Functions – Informix 11.10

- XML publishing functions produce SQL results as XML and support XPATH queries

- genxml() / genxmlob() - SQL results as XML elements
- genxmlem() / genxmlemlob() - column vals as XML elements
- genxmlschema() / genxmlschemalob() - schema as XML
- genxmlquery() / genxmlquerylob() - result set as XML
- genxmlqueryhdr() / genxmlqueryhdrlob() - result set as XML with header
- extract() / extractxmlob() - evaluate XPATH expression
- extractvalue() / extractxmlobvalue() - value of XML node
- existsnode() - verify whether a node exists in XML doc

XML EXTRACT() Example

`-extract--(--xml_string--,--xpath_expression--)-----><`

- This example evaluates the XML contained in column col2 of table tab and returns the given name for Jason Ma:

```
SELECT
extract(col2,'/personnel/person[@id="Jason.Ma"]/name/given')
FROM tab;

<given>Jason</given>
```

- Another example which returns the entire name:

```
SELECT extract(col2, '/personnel/person[@id="Jason.Ma"]/name')
FROM tab;

<name>
<family>Ma</family>
<given>Jason</given>
</name>
```

XML Validation Function

- **IDSXMLPARSE (lvarchar) , IDSXMLPARSE (CLOB)**
 - Arguments are XML documents
 - Returns the XML document if valid, generates an error if invalid
ex: (U0001) – IDSXMLPARSE : Error parsing the XML document string
- Some basic well-formed XML rules:

	Not well-formed	Well-formed
Has exactly one root element	bla <c>blub<a></c>	<a> bla
Each opening tag is matched by a closing tag	<a>bla	<a>bla
All elements are properly nested	<a>bla	<a>bla
Attribute values must be quoted		
Does not use disallowed characters in tags or values	<a> 3<5 	<a> 3<5

XSL Transformation Functions (XSLT) – Informix 11.50

- Extensible Stylesheet Language Transformation (XSLT) is a language for transforming XML documents into other XML documents
 - XML to XML
(transform to confirm different schema/standard)
 - XML to HTML
 - XML to PDF
- Based on the retired XSLT DataBlade
 - **xsltransform**(lvarchar, lvarchar) returns lvarchar(32739)
 - xsltransform(clob, lvarchar) returns lvarchar(32739)
 - xsltransform(clob, clob) returns lvarchar(32739)
 - xsltransform(blob, lvarchar) returns lvarchar(32739)
 - xsltransform(blob, informix.blob) returns lvarchar(32739)
 - **xsltransformAsClob**(lvarchar, lvarchar) returns clob
 - xsltransformAsClob(clob, lvarchar) returns clob
 - xsltransformAsClob(clob, clob) returns clob
 - **xsltransformAsBlob**(lvarchar, lvarchar) returns blob
 - xsltransformAsBlob(blob, lvarchar) returns blob
 - xsltransformAsBlob(blob, blob) returns blob
 - **ifx_checksum**(bigint, integer) returns informix.integer

www.alphaworks.ibm.com/tech/xsltblade

www.w3.org/TR/xslt

Extensible Stylesheet Language Transformation Example

- Info

```
<?xml version='1.0' encoding='ISO-8859-1' ?><doc>Hello world!</doc>
```

- Style

```
<?xml version='1.0'?>  
  <xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>  
    <xsl:output encoding='US-ASCII'/>  
    <xsl:template match='doc'>  
      <out><xsl:value-of select='.'/></out>  
    </xsl:template>  
  </xsl:stylesheet>
```

```
select xsltransform(info, style) from t
```

```
<?xml version="1.0" encoding="US-ASCII"?><out>Hello world!</out>
```

New ISOLATION Level – Informix 11.50

- Isolation levels define how a session's transactions are affected by other session transactions
- A new isolation level called **COMMITTED READ LAST COMMITTED**
- Similar to COMMITTED READ in that it only returns committed data
 - However, it ensures that reads will never block!
 - If a session encounters a locked resource, Informix will retrieve the value that was there before the current lock holder session made any change – last committed
- The last committed values is always available since it is used in case of ROLLBACK WORK
- Use:
 - **SESSION** Level:
SET ISOLATION TO COMMITTED READ LAST COMMITTED
 - **ONCONFIG**:
USELASTCOMMITTED [None, Committed Read, Dirty Read, All]

Automatic Re-Compilation of Prepared Statements – Informix 11.50

- In previous releases, when a session tried to open a statement that's using an altered object, Informix raised -710 error
 - The client program was forced to catch the exception, re-prepare the statement before proceeding
- In Informix 11.50, after a DDL operation modifies the schema of a database table, Informix now automatically performs the following actions:
 - Informix automatically issues the UPDATE STATISTICS statement to recalculate routine statistics for all SPL routines that reference the table
 - Informix automatically issues the PREPARE statement to update any prepared objects that reference the table
- For previous behavior, use [AUTO_REPREPARE](#) configuration parameter

Agenda

- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- **Informix Extensibility Features**
- Appendix

What is Database Extensibility?

- Ability to add business components in the database
- Tailor the database to the business environment
- Involves including non-standard data types, constraints, inheritance, functions and APIs in the database engine
- Used by developers and DBAs to create and manage data and applications according to the business use of the information
- It requires application developers and DBAs to think about information differently

Databases are not commodities!

Why Use Database Extensibility?

- Solve problems not easily possible before
- Reduce application complexity
 - Put the processing where it makes the most sense
 - Set processing provided by the database
- Higher performance
 - Less data movement, better data representation, better indexing
- Faster development, lower maintenance cost

Informix Extensibility Features

- Informix provides a complete set of features to extend the database server

Data types (Built-in and User-Defined)

Casts and Castings

Table and Type Inheritance

Indexing (R-Tree and Functional)

User-defined Routines

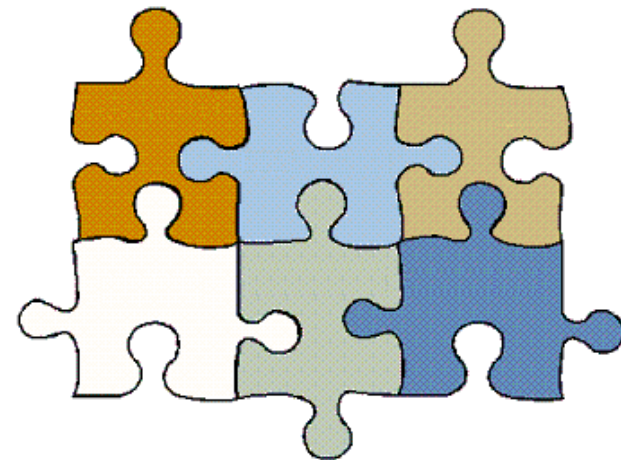
User-defined Aggregates

Access Methods

Virtual Tables and Indexes

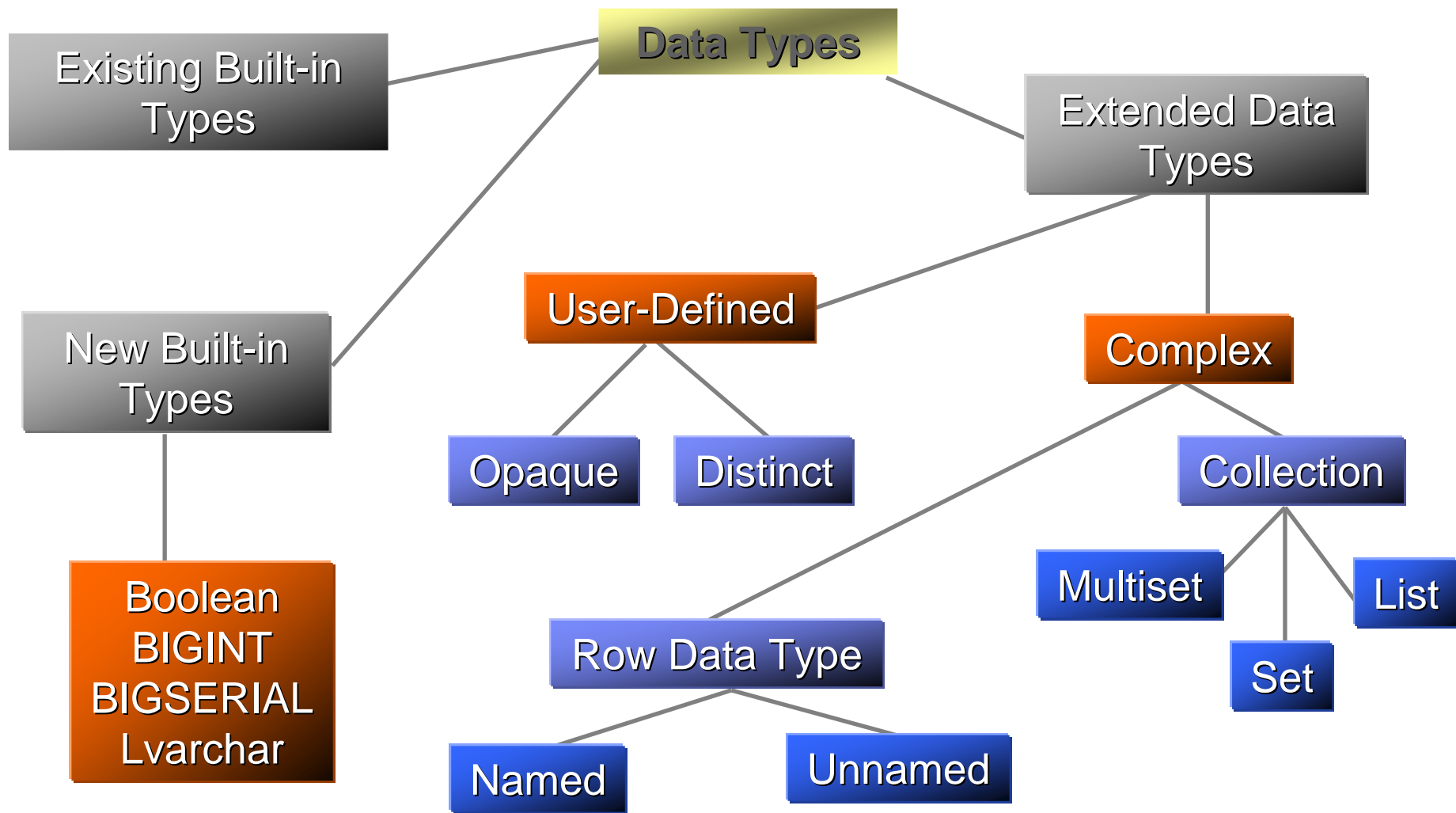
DataBlades

User Virtual Processors



Extensions can be written in:
SPL, C, Java

Complex and User-Defined Data Types



Complex Data Types - Row

- Analogous to C Structures
- Come in two “kinds”
 - **NAMED** - strongly typed, identified by name, has inheritance
 - **UNNAMED** - weakly typed, identified by structure, no inheritance

N
A
M
E
D

```
create row type name_t
(fname char(20),
 lname char(20));

create row type address_t
(street_1 char(20),
 street_2 char(20),
 city char(20),
 state char(2),
 zip char(9));

create table student
(student_id serial,
 name name_t,
 address address_t,
 company char(30));
```

U
N
N
A
M
E
D

```
ROW (a int, b char (10))
```

Note: is also equal to:
ROW(x int, y char(10))

```
create table part
(part_id serial,
 cost decimal,
 part_dimensions
 row ( length decimal,
        width decimal,
        height decimal,
        weight decimal));
```

Complex Data Types - Row

INSERT statement:

```
insert into student values
(
1234,
row ("John", "Doe" ::name_t,
row ("1234 Main Street",
"",
"Anytown", "TX", "75022" ::address_t
"Informix Software")
```

Use of datatype
keyword

Cast the row type!

Use of datatype
keyword

- Drop a named row type:
drop row type *address_t*
restrict;

SELECT statement:

```
select * from student where
name.lname matches "Doe";
```

Result set:

student_id	1234
name	ROW('John','Doe')
address	ROW('1234 Main Street', '', 'Anytown', 'TX','75022')
company	Informix Software

User-Defined Data Types - *Distinct*

- Modeled on an existing data type
- Has a unique name to distinguish it from other similar “types”
- Inherits the internal structure, operations and casts from it's source type
- Can define additional operations on distinct types

```
create distinct type dollar as decimal;
create distinct type aus_dollar as decimal;
create table sales
( sku int,
  sales_date date,
  us_sales dollar,
  aus_sales aus_dollar);
```

```
insert into sales values (1234, today,
  15.0::dollar, 0::aus_dollar);
insert into sales values (5678, today,
  0::dollar, 75.0::aus_dollar);
```

```
select sku,
  (sum(us_sales) + sum(aus_sales))
from sales where sales_date = today
group by 1;
```

error: 674 - routine (plus) can not be resolved

User-Defined Data Types - *Distinct*

- To avoid error 674, create two User Defined Functions (UDFs) that handle the value conversion (a.k.a. *Casting Functions*):

```
create function usdlr_to_ausdlr(parm1 dollar)
  returning aus_dollar
  specific usd_to_ausd;
  return (parm1::decimal * 1.8)::aus_dollar;
end function;

create function ausdlr_to_usdlr(parm1 aus_dollar)
  returning dollar
  specific ausd_to_usd;
  return (parm1::decimal / 1.8)::dollar;
end function;
```

- Rerun the query as follows:

```
select sku, (sum(us_sales)+ sum(ausdlr_to_usdlr(aus_sales))::dollar)
  from sales where sales_date = today
 group by 1;
```


Casts and Casting

- Casts allow comparisons between values of different data types
 - Informix substitute a value of one data type for a value of another data type
- Informix provides a number of “built-in” casts (*INT* to *DECIMAL*, *NUMERIC* to *CHAR*, etc.) for most built-in data types
 - Automatically casts types from one type to another when necessary (expressions, assignments, parameter passing)
- However, must create user-defined casts for *user-defined types*
 - Must be unique with respect to source and target data types
- Two kinds of Informix user-defined casts
 - **IMPLICIT** - Informix automatically invokes the cast when necessary
 - **EXPLICIT** - Informix requires the application/user to invoke the cast

Casts and Casting – *Implicit Casts*

```
select sum(us_sales) + sum(aus_sales) from sales;
```

674: Routine (plus) can not be resolved.

```
create implicit cast (aus_dollar as dollar);
```

```
select sum(us_sales) + sum(aus_sales) from sales;
```

(expression) 120.00 ← Wrong result!

```
drop cast (aus_dollar as dollar);
```

```
create implicit cast (aus_dollar as dollar with
```

ausdlr_to_usdlr);

```
select sum(us_sales) + sum(aus_sales) from sales;
```

(expression) 80.00 ← Right result!

Casts and Casting - *Explicit Casts*

```
select sum(us_sales) + sum(aus_sales) from sales;
```

674: Routine (plus) can not be resolved

```
create explicit cast (aus_dollar as dollar);
```

```
select sum(us_sales) + sum(aus_sales) from sales;
```

674: Routine (plus) can not be resolved

```
select sum(us_sales) + sum(aus_sales)::dollar from sales;
```

(expression) 120.00

```
drop cast (aus_dollar as dollar);
```

```
create explicit cast (aus_dollar as dollar with  
                                ausdlr_to_usdlr);
```

```
select sum(us_sales) + sum(aus_sales)::dollar from sales;
```

(expression) 80.00

DataBlade Modules

- DataBlade Modules extend the functionality of the engine by adding:
 - UDTs
 - Casts
 - Error messages
 - Aggregates
 - UDRs
 - Interfaces
 - client code (where necessary)
 - access methods to manipulate UDTs
- Functionality available through SQL, SPL, API calls to external functions
- Empowers customization
- Use the Blade Manager to manage blade registration
 - Graphical version available with Windows and ISA
 - Command line on UNIX / Linux

Many DataBlades – *Some Free!*

- BTS
- Geodetic
- Spatial
- TimeSeries
- NAG (National Algorithms Group)
- Image
- Binary
- MQ Series
- Spatial (ESRI, MapInfo, Geodetic)
- Text (Verity & Excalibur)
- Alerter
- Facial Recognition
- Fingerprint Recognition
- C-ISAM
- Node (hierarchical)
- XML
- Web

**See APPENDIX for
more information
about individual
DataBlades**

DataBlade Developers Kit (DBDK)

- DataBlade Developers Kit (DBDK) is a comprehensive graphical development environment for creating new data types and functions
- GUI tool only runs on Windows
- DBDK consists of three tools:
 - BladeSmith
 - Defines/build DataBlade modules
 - Generates functional tests
 - Generates script info for install
 - BladePack
 - Organize distribution
 - Generates Installshield script
 - Blade Manager
 - Stores info in the database
 - Install/uninstall DataBlades in a database



Questions

Agenda

- Informix and Application Development
- IBM Common Client with DRDA Support
- Informix Application Programming Technologies
- Triggers, Stored Procedures, UDFs
- Informix 11.x Enhancements
- Informix Extensibility Features
- Appendix

Appendix A – Application Development

- DRDA Enhancements in Informix 11.x
- Other Informix 11.x Enhancements
- Programming Language Examples
- Application Development Resources

Complete DRDA Enhancements in Informix 11

- **Support for connecting to a database with a name longer than 18 bytes**
- **Batching of multiple statements into one request**
- **Support for displaying DRDA session information**
 - `sys sesappinfo` table in `sysmaster`
 - `onstat -g ses` command
- **Support for using `IS NULL` or `IS NOT NULL` with expressions**
- **Secure Socket Layer (SSL) Support for DRDA Clients (JCC and CLI)**
 - IBM Global Security Kit (GSKit) is be used to provide SSL support
- **Support for DRDA connections between primary and shared disk secondary servers in high availability clusters**
- **Support for Callable statements**
- **Support for Variable FETCH buffer size**
`ResultSet.SetFetchSize()`
- **New Configuration Parameter to set the size of the DRDA communication Buffer - `DRDA_COMMBUFFSIZE`**
- **Support for BLOB/CLOB, INTERVAL and other built-in data types over the common API**
- **Support for built-in data types**
- **Xtrace support**
- **XA Support (tightly coupled transaction)**
- **Support for DRDA connections to ER nodes**
- **Support for FORWARD ONLY updateable cursors**
- **Support for new built-in functions –**
 - the `sysibm.Metadata` function provides database metadata information to Java Common Client (JCC) client applications
 - The `sysibm.SCLCMessage` function supports DRDA error handling

More Built-In Data Types Now Supported in Informix 11

- Additional built in data types supported for cross-database, cross-server queries
- Informix 11 enhances the data types support for cross-server (ISTAR) distributed queries.
 - Boolean
 - LVARCHAR
 - Distinct types of Boolean
 - Distinct types of LVARCHAR
 - DISTINCT TYPES of distinct types listed here
 - Distinct types of basic SQL types

Char	INT	SMALLINT	FLOAT
SMALLFLOAT	DECIMAL	MONEY	SERIAL
DATE	MONEY	DATETIME	INTERVAL
BYTE	TEXT	VARCHAR	NCHAR
NVARCHAR	INTS	SERIAL8	INT8



IDS 11 Application Development Enhancements				
Dynamic SQL in SPL	Multiple INSTEAD OF triggers on a view	Row versioning	Single sign-on	Derived table in FROM clause
IS [NOT] NULL predicate	Index Self-Join	Indexable binary type	SSL encryption	.Net 2.0 support
DRDA enhancements	sysdbopne / sysdbclose	Optimistic Concurrency	No temp tables in Union	Optimizer Directive for ANSI joined queries
New Built-In SQL Expressions and Functions	Auto Re-Prepare	SPL enhancements	Task Scheduling	Named parameters in JDBC callable statements
Create SPL procedures which can refer to applicable OLD and NEW trigger correlated values	DRDA support and session information	External directives control for a session	Explain output in XML format	Web feature services
Oracle compatibility functions	New Boolean operators for trigger procedures: DELETING, INSERTING, SELECTING, and UPDATING	Multiple INSERT, DELETE, UPDATE, and SELECT triggers on a table	Monitor and analyze recent SQL statements	XML publishing and manipulation
New SPL Looping Syntax	DBACCESS "C" style comments support	Basic text search Datablade	Inclusion in Zendcore (PHP)	Common drivers for Informix and DB2

New SQL Built-in Functions (11.10)

Additional Date, Numeric, Bit, String functions

ADD_MONTHS()
BITAND()
BITNOT()
BITXOR()
FLOOR()
LAST_DAY()
MONTHS_BETWEEN()
NULLIF()
ROUND()
SYSDATE()
TO_NUMBER()

ASCII()
BITANDNOT()
BITOR()
CEIL()
FORMAT_UNITS()
LTRIM()
NEXT_DAY()
POWER()
RTRIM()
TO_CHAR()
TRUNC()

Change Data Capture (CDC)

- Capturing Transactional Data with the Change Data Capture API
 - Client applications can use the Change Data Capture (CDC) API to capture transactional data from Informix
 - Includes a new system database called *syscdc*, which includes built-in SQL functions that control data capture
- Monitor Change Data Capture Sessions
 - Use the new `onstat -g cdc` command
 - Displays information about the captured tables, the buffers being used by sessions, the configuration of sessions, or the data capture activity
- View Change Data Capture Error Message Text
 - Corresponds to an error name
 - Viewed by using the new **`cdc_errortext()`** function

Rolling Back SQL Transactions to a Savepoint

- A savepoint identifies an arbitrary location within the statements of an SQL transaction
- Declare or reference savepoint objects in SQL statements
- Useful for client applications to rollback *only* the portion of the transaction that follows the specified savepoint

```
begin work;  
savepoint save_1;  
{execute some transactions}  
savepoint save_2;  
if some_condition then  
    rollback work to savepoint save_1  
    commit work;  
else  
    commit work;
```

- Rolls back to the specifically named savepoint, in this case: **save_1**

Derived table support

- Sub-select in the FROM clause now complies with ISO/IEC 9075:1992, the SQL-92 standard
- Examples:

```
select sum(vc1) as sum_vc1, vc2
from (SELECT c1, c2 FROM t1 ) AS vtab(vc1, vc2)
group by vc2;
```

```
select * from
( (SELECT c1,c2 FROM t3) AS vt3(v31,v32)
left outer join
    ( (SELECT c1,c2 FROM t1) AS vt1(vc1,vc2)
    left outer join
        (SELECT c1,c2 FROM t2) AS VT2(vc3,vc4)
        ON VT1.VC1 = vt2.vc3)
ON vt3.v31 = vt2.vc3);
```


More 11.x Application Enhancements

- Support for DELETE and UPDATE operations with subqueries that reference the same table object
- Support for the LVARCHAR data type with the CONCAT function and the || operator
- Setting the Frequency of Error Checking for Smart Large Object Transmission
 - Use the IFX_LOB_XFERSIZE environment variable to specify the number of bytes in a CLOB or BLOB to transfer from a client application to Informix before checking for errors
- DataBlade Module Registration through SQL
 - Use the built-in **SYSBldPrepare()** function to register one or more DataBlade modules or to unregister a DataBlade module
 - Alternative to using the BladeManager utility

More 11.x Application Enhancements (cont.)

- Use SQL expressions as operands of the IS NULL and IS NOT NULL predicate
 - Previously, only column names were allowed
 - Provides a value for entries that otherwise are not computable
- Add a version column to a table to contain both a checksum and a version number
 - Use a version column to detect if a row has been updated since it was originally queried
 - Version numbers help detect differences if a row is deleted and another row is re-inserted into a table
 - Use the `ALTER TABLE tablename ADD VERCOLS` statement
 - Beneficial to applications requiring latest information

Named Parameters in a JDBC CallableStatement

- A CallableStatement provides a way to call a stored procedure on the server from a Java™ program
- Introduced in the JDBC 3.0 specification
- Adds the convenience of being able to identify parameters by name instead of by ordinal position
- If the stored procedure is unique, you can omit parameters that have default values and you can enter the parameters in any order
- Named parameters are especially useful for calling stored procedures that have many arguments and some of those arguments have default values

Index Binary Data Types

- The new Binary UDT DataBlade module provides two new data types allowing you to store binary-encoded strings, which can be indexed for quick retrieval
- The *binaryvar* data type is a variable-length opaque type with a maximum length of 255 bytes
- The *binary18* data type is the same as the *binaryvar* data type except it holds a fixed value of 18 bytes
- As part of a new DataBlade module, these data types come with string manipulation functions to validate the data types and bitwise operation functions that allow you to perform bitwise logical AND, OR, XOR, and NOT comparisons.

Structured Query Language (SQL)

- Good resources for Developers (PDF documents)
 - IBM Informix Guide to SQL: Reference
 - IBM Informix Guide to SQL: Syntax
 - IBM Informix Guide to SQL: Tutorial

www.ibm.com/software/data/informix/pubs/library

SPL Routines – Parameters

- Formal argument in the declaration of a UDR
- Declare a *name* and *data type* for each parameter passed to UDR
- Pass as many as needed
 - Limit 64K for total size of parameters
- Can be any SQL, complex or user-defined data type except:
 - BIGSERIAL, BLOB, BYTE, CLOB, SERIAL, SERIAL8, or TEXT
- Can specify DEFAULT value for a parameter
- OUT keyword
 - Corresponds to a value the routine returns indirectly, through a pointer
 - Extra value in addition to any values that it returns explicitly
- INOUT keyword
 - Supported for UDRs written in the SPL, C, or Java™ languages
 - A value is passed by reference and any modified value is returned

SPL Routines – REFERENCING/FOR Clause

- Can include REFERENCING and FOR *table_object* clauses immediately after the CREATE PROCEDURE parameter list
 - Also known as a *trigger UDR* or *trigger routine*
- FOR clause
 - Specifies the table/view whose triggers can invoke the routine from the FOR EACH ROW section of their Triggered Action list
- REFERENCING clause
 - Declares correlation names for the *original* value (OLD clause) and for the *updated* value (NEW clause) in columns of the *table_object* that the FOR clause specifies
- Use of correlations names to reference OLD and/or NEW values depends on triggered action

SPL Routines – RETURNS/RETURNING Clause

- Specifies the data type of a value or values that a user-defined function returns
- Use with a list of data types
- Can be any SQL data type, except:
 - SERIAL, SERIAL8, TEXT, or BYTE
- Must specify a RETURN statement in procedure body
- Can specify more than one data type in the Return clause

```
CREATE FUNCTION val_comp (val1 INTEGER, val2 INTEGER)  
    RETURNING INTEGER AS comp_res
```


SPL Routines – Statement Block

- A group of SPL and SQL statements
- Define the scope of a variable or of the ON EXCEPTION statement
- Between CREATE PROCEDURE/FUNCTION and END PROCEDURE/FUNCTION is an implicit statement block
- Use BEGIN and END to specify an explicit statement block nested within another statement block
- ROLLBACK WORK SQL statement is *not* valid in an SPL statement block

Some Valid SPL Statements

<<Label >>
CALL
CONTINUE
EXIT
FOR
FOREACH
GOTO
IF
LET
LOOP
RAISE
EXCEPTION
RETURN
SYSTEM
TRACE
WHILE

SPL Routines – Statement Block – Other Statements

- **DEFINE**
 - Declares local variables that an SPL routine uses, or to declare global variables that can be shared by several SPL routines
 - Variables are held in memory, not in database
 - Variables can be any SQL data type and extended data types, except:
 - Serial, Serial8, Text or Byte
 - Example: DEFINE x, y INT
- **ON EXCEPTION**
 - Specifies actions to be taken for any error, or for a list of one or more specified errors, during execution of a statement block
- **EXECUTE FUNCTION/PROCEDURE**
 - Invokes a user-defined function or procedure
 - Function returns a value

SPL Routines – Local Variables

- **Local Variable**
 - Valid only for the duration of the SPL routine
 - Reset each time the SPL routine is executed
 - Cannot have default value
- **Scope**
 - Available in the statement block in which it is defined, and within any nested statement block
 - Can be redefined in a statement block

```
CREATE PROCEDURE scope()
  DEFINE x,y,z INT;
  LET x = 5;
  LET y = 10;
  LET z = x + y;  --z is 15
  BEGIN
    DEFINE x, q INT;    -- x is redefined
    DEFINE z CHAR(5);  -- z is redefined
    LET x = 100;
    LET q = x + y;      -- q = 110
    LET z = 'silly';    -- z receives a character value
  END
  LET y = x;    -- y is now 5
  LET x = z;    -- z is now 15, not 'silly'
END PROCEDURE;
```

SPL Routines – Global Variables

- Available to other SPL routines that are run by the same user session on the same database
- Requires a default value
- Must be defined in any SPL routine in which it is used
- Carries its value from one SPL routine to another until the session ends
- Cannot be a collection variable

```
CREATE FUNCTION func1() RETURNING INT;  
  DEFINE GLOBAL gvar INT DEFAULT 2;  
    LET gvar = gvar + 1;  
    RETURN gvar;  
END FUNCTION;
```

```
CREATE FUNCTION func2() RETURNING INT;  
  DEFINE GLOBAL gvar INT DEFAULT 5;  
    LET gvar = gvar + 1;  
    RETURN gvar;  
END FUNCTION;
```

```
EXECUTE FUNCTION func1();  
EXECUTE FUNCTION func2();
```

gvar: 3
gvar: 4

```
EXECUTE FUNCTION func2();  
EXECUTE FUNCTION func1();
```

gvar: 6
gvar: 7

SPL Routines – RETURN Statement

- Use RETURN statement to pass values back
- Return values must match the type and order defined in RETURNING clause
- WITH RESUME keyword
 - The routine resumes at the statement after the RETURN WITH RESUME statement

```
CREATE FUNCTION foo(a int) RETURNING CHAR(20)
  IF ( a = 5 ) THEN
    RETURN "HELLO" WITH RESUME;
  ELSE
    RETURN "HELLO";
  END IF;

  RETURN "world";
END FUNCTION;
```

EXECUTE FUNCTION foo(1);

HELLO

EXECUTE FUNCTION foo(5);

HELLO
world

SPL Routines – Exception Handling – ON EXCEPTION

- ON EXCEPTION statement
 - Provides error-trapping and error-recovery mechanism
 - Specify errors to trap with IN clause
 - Specify actions to take if the error occurs
 - Allows multiple ON EXCEPTION statements in one statement block
 - Must be defined after DEFINE statements and *before* any executable statement in a statement block

```
CREATE PROCEDURE ex_test()
  DEFINE sql_err INTEGER;
  DEFINE isam_err INTEGER;
  DEFINE err_txt CHAR(200);

  ON EXCEPTION IN (-206) SET sql_err, isam_err, err_txt

    CREATE TABLE tab1 ( col1 INT, col2 INT); -- creates tab1 and return

  END EXCEPTION WITH RESUME

  INSERT INTO tab1 VALUES (1, 2); -- tab1 doesn't exist, go to exception handling
  INSERT INTO tab1 VALUES (2, 3); -- resume here after exception handling
```

SPL Routines – Exception Handling – RAISE EXCEPTION

- Generates error
- Can specify SQL error and, optionally, ISAM error and error message
- The generated error can be caught by ON EXCEPTION
- Use special error number - 746 to produce a customized message

```
CREATE PROCEDURE ex_test ( a INT )  
  
ON EXCEPTION IN (-206) SET sql_err, isam_err, err_txt  
  
    CREATE TABLE tab1 ( col1 INT, col2 INT);  
  
    RAISE EXCEPTION sql_err, isam_err, err_txt;  
  
END EXCEPTION;  
  
IF (a < 1) THEN  
    RAISE EXCEPTION -746, 0, "Insert value must be greater than 0";  
END IF;  
  
INSERT INTO tab1 VALUES (1, 2); -- if tab1 doesn't exist, go to exception handling  
  
END PROCEDURE;
```

SPL Routines – Tracing

- Use SET DEBUG FILE statement to identify the trace file that receives the runtime trace output of a SPL routine
- Use TRACE statement to trace the execution of the routine

```
CREATE FUNCTION test_trace ()  
  
    RETURNING INT  
  
    DEFINE count, a INT;  
  
    LET count = 0;  
  
    SET DEBUG FILE TO "/tmp/trace_out";  
  
    TRACE ON;  
  
    LET count = (SELECT count(*) FROM tab1);  
  
    RETURN count;  
  
END FUNCTION;
```

Trace File output:

```
trace on  
expression:  
  (select (count *) from tab1)  
evaluates to 3 ;  
let count = 3  
expression:count  
evaluates to 3  
procedure test_trace returns 3  
  
iteration of cursory procedure  
test_trace
```

- TRACE ON: trace everything (statements, function calls, variables, returned values)
- TRACE OFF: turn all tracing off
- TRACE PROCEDURE: trace routine calls
- TRACE <expression>: output the expression to trace file

SPL Routines – Other Actions

- Execute SPL Routine
 - EXECUTE PROCEDURE and EXECUTE FUNCTION
 - CALL statement to execute a SPL routine from another SPL routine
 - Use routine name with an expression in an SQL statement
- Drop SPL Routine
 - DROP PROCEDURE to drop SPL procedures
 - DROP FUNCTION to drop SPL functions
 - DROP ROUTINE to drop both SPL procedures and functions
- Re-optimize SPL Routine
 - UPDATE STATISTICS FOR PROCEDURE my_proc;
 - UPDATE STATISTICS FOR FUNCTION my_func;
 - UPDATE STATISTICS FOR ROUTINE my_routine;

I4GL Code Example

```
DATABASE stats
GLOBALS
  DEFINE      player_bats RECORD LIKE informix.atbats.*
  DEFINE      batting_ave RECORD LIKE informix.battingaverage.*
END GLOBALS
-----
MAIN
-----
  DEFINE BA DECIMAL (4,3)
  DECLARE curs_at_bats cursor for select * from informix.atbats
  OPEN curs_at_bats
  WHILE (SQLCA.SQLCODE = 0)
    FETCH curs_at_bats into player_bats.*
    IF (SQLCA.SQLCODE = 100) THEN
      EXIT WHILE
    END IF
    CALL compute_BA(player_bats.atbats, player_bats.hits) RETURNING BA
    INSERT INTO informix.battingaverage VALUES (player_bats.lname, BA);
  END WHILE
  COMMIT WORK
  CLOSE curs_at_bats
END MAIN
```

```
FUNCTION compute_BA(at_bats, hits)
  DEFINE at_bats INTEGER
  DEFINE hits INTEGER
  DEFINE BA DECIMAL (4,3)

  LET BA = hits/at_bats
  RETURN BA
END FUNCTION
```

Informix + ESQL/C Example

```
#include <stdio.h>
EXEC SQL define FNAME_LEN      15;
EXEC SQL define LNAME_LEN      15;
main()
{
EXEC SQL BEGIN DECLARE SECTION;
    char fname[ FNAME_LEN + 1 ];
    char lname[ LNAME_LEN + 1 ];
EXEC SQL END DECLARE SECTION;
    printf( "DEMO1 Sample ESQL Program running.\n\n");
    EXEC SQL WHENEVER ERROR STOP;
    EXEC SQL connect to 'stores_demo';
    EXEC SQL declare democursor cursor for
        select fname, lname into :fname, :lname from customer where lname < "C";
    EXEC SQL open democursor;
    for (;;)
    {
        EXEC SQL fetch democursor;
        if (strcmp(SQLSTATE, "00", 2) != 0)
            break;
        printf("%s %s\n",fname, lname);
    }
    if (strcmp(SQLSTATE, "02", 2) != 0)
        printf("SQLSTATE after fetch is %s\n", SQLSTATE);
    EXEC SQL close democursor;
    EXEC SQL free democursor;
    EXEC SQL disconnect current;
    exit(0);
}
```

Informix + PERL Example

```
#!/usr/bin/perl
use strict;
use DBI;
my $dbh = DBI->connect (
    "DBI:Informix:stores",
    '', '',
    { PrintError => 0, RaiseError => 1, AutoCommit => 0 } );
$dbh->do("begin");
my $sth1 = $dbh->prepare("select code, sname from state");
$sth1->execute();
my $sth2 = $dbh->prepare("delete from state where code = ?");
while (my $row = $sth1->fetchrow_hashref())
{
    if ($row->{code} =~ /^C/)
    {
        $sth2->bind_param(1, $row->{code});
        $sth2->execute();
        print sprintf("DELETED: %-2s %-15s\n", $row->{code}, $row->{sname});
        next;
    }
    print sprintf("%-2s %-15s\n", $row->{code}, $row->{sname});
}
$dbh->do("commit");
$dbh->disconnect();
exit 0;
```

Informix + JAVA Example

```
import java.sql.*;

public class jcctest {
    public static void main() {
        try {
            // Initialize JCC driver
            Class.forName("com.ibm.db2.jcc.DB2Driver");
        }
        catch (Exception e) {
            System.out.println("driver err: " + e.getMessage());
        }
        String url = "jdbc:Informix://localhost:1529/stores_demo";
        Connection conn = null;
        try {
            // get database connection
            conn = DriverManager.getConnection(url,"user","pass");
            ...
            conn.close();
        }
        catch(SQLException e) {
            System.out.println("SQL Error: " + e.getMessage());
        }
    }
}
```

JDBC application using IBM Informix driver

```
import java.sql.*;
public class TestInformixConnection
{
    public static void main (String[] args) throws
    Exception
    {
        Connection conn1;
        String URL = "jdbc:informix-sqli:" +
            "//myhost:1242/test:INFORMIXSERVER=testserver";

        try {
            Class.forName
                ("com.informix.jdbc.IfxDriver").newInstance();
        }
        catch (Exception e) { // Handle exception }
        try {
            conn1 = DriverManager.getConnection (URL,
                "testuser", "testpass");
        }
        catch (SQLException se) { // Handle exception }

        System.out.println("Established connection");
        conn1.close();
    }
}
```

JDBC application using Data Server driver (JCC)

```
import java.sql.*;
public class TestJccConnection
{
    public static void main (String[] args) throws
    Exception
    {
        Connection conn1;
        String URL = "jdbc:db2://myhost:1243/test:";

        // Load the JDBC driver ..
        try {
            Class.forName
                ("com.ibm.db2.jcc.DB2Driver").newInstance();
        }
        catch (Exception e) { // Handle Exception }
        try {
            conn1 = DriverManager.getConnection (URL,
                "testuser", "testpass");
        }
        catch (SQLException se) { // Handle Exception }
        System.out.println("Established connection");
        conn1.close();
    }
}
```

Informix & JAVA - Two JAR files



JAR file	Driver version	Level of JDBC support	Minimum level of Java
db2jcc.jar	3.50+	JDBC 3.0 and earlier	1.4
db2jcc4.jar	4.0+	JDBC 4.0 and earlier	6.0

- Informix 11.10 shipped with JCC 3.50 but also certified with JCC 4.0.
- Informix 11.5 requires JCC 3.52 and JCC 4.2 to fully exploit MACH11 and other Informix features

www.ibm.com/software/data/informix/ids/ad/java.html

Informix + Python Example

```
#!/usr/bin/python
import sys
import ibm_db # import the ibm_db module

conn = informixdb.connect("stores")

cursor1 = conn.cursor(rowformat = informixdb.ROW_AS_DICT)
cursor1.execute('select code, sname from state')
cursor2 = conn.cursor()

for row in cursor1:

    if row['code'][0] == 'C':
        cursor2.execute('delete from state where code = ?', (row['code'],))
        print "DELETED: %-2s %-15s" % (row['code'], row['sname'])
        continue

print "%-2s %-15s" % (row['code'], row['sname'])

conn.commit()
conn.close()

sys.exit(0);
```


Informix + PHP Example

```
<?php

$dbh = new PDO("informix:host=starship2; service=1753; database=stores; server=starship2;
               protocol=onsoctcp");

$dbh->beginTransaction();
$stmt1 = $dbh->prepare("select code, sname from state");
$stmt1->execute();
$stmt2 = $dbh->prepare("delete from state where code = ?");

while( $row = $stmt1->fetch() )
{
    if (strcmp($row['CODE'], "C", 1) == 0)
    {
        $stmt2->bindParam(1, $row['CODE']);
        $stmt2->execute();
        printf("DELETED: %-2s %-15s\n", $row['CODE'], $row['SNAME']);
        continue;
    }
    printf(" %-2s %-15s\n", $row['CODE'], $row['SNAME']);
}
$dbh->commit();
$dbh = null;
?>
```

Informix + ADO.NET Example

```
using System;
using IBM.Data.Informix;
namespace IfxAdoPres.Basics {
    public class BasicConnection {
        const string HOST = "huskie";
        const string SERVICENUM = "1601";
        const string SERVER = "cic_ids";
        const string DATABASE = "cic";
        const string USER = "infx";
        const string PASSWORD = "morteste";
        public IfxConnection conn = new IfxConnection();
        public BasicConnection() {}
        public void MakeConnection() {
            string ConnectionString = "Host = " + HOST + "; " +
                "Service=" + SERVICENUM + "; " +
                "Server=" + SERVER + "; " +
                "Database=" + DATABASE + "; " +
                "User Id=" + USER + "; " +
                "Password=" + PASSWORD + "; ";
            conn.ConnectionString = ConnectionString;
            try {
                conn.Open();
                Console.WriteLine("made connection!");
                Console.ReadLine();
            } catch (IfxException ex) {
                Console.WriteLine("Problem with connection attempt: "+ex.Message);
            }
        }
        public void CloseConnection() {
            conn.Close();
        }
    }
}
```

Informix + RUBY Example

```
#!/usr/local/bin/ruby -w

require 'informix'
db = Informix.connect('stores')

db.transaction {

  cursor = db.cursor("select code, sname from state")

  stmt_delete = db.prepare("delete from state where code = ?")

  cursor.open.each_hash {|row|
    if row['code'][0].chr == 'C'
      stmt_delete.execute(row['code'])
      printf("DELETED: %-2s %-15s\n", row['code'], row['sname'])
    next
  end

  printf("%-2s %-15s\n", row['code'], row['sname'])
}.close

}
```

db.close

Resources

- developerWorks Forum:
 - Unleash the new Java, .NET, PHP and Ruby client and developer solutions for Informix
www.ibm.com/developerworks/forums/dw_thread.jsp?forum=1137&thread=170325&cat=19
- Python Driver
 - For the latest driver go to: pypi.python.org/pypi/ibm_db
- PHP Resources
 - Developing PHP applications:
www.ibm.com/software/data/informix/ids/ad/php.html
 - Developing PHP Applications for IBM Data Servers: A Redbook
www.redbooks.ibm.com/abstracts/sg247218.html
 - PHP Official Site: www.php.net
 - PHP Extensions: www.pecl.php.net

Resources

- RUBY Developer Resources:
 - rubyforge.org/projects/ruby-informix
 - rails-informix.rubyforge.org
 - rubyforge.org/projects/rubyibm
- Connects to Informix 11.x via DRDA protocol
 - rubyforge.org/projects/rubyibm
- The open source ruby-informix driver developed by Gerardo Santana is also available and supports recent Informix versions using the SQLI protocol
 - rubyforge.org/projects/ruby-informix
- .NET and Visual Studio
 - www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=swg-vsai
 - Provided with Client SDK and includes the Visual Studio Add-Ins.
 - www.ibm.com/software/data/informix/ids/ad/dotnet.html
- Developer Resources:
 - IBM Web 2.0 Developer Kit
 - www.ibm.com/developerworks/lotus/kits/d-ls-web20kit
 - IBM Web 2.0 Portal
 - www.ibm.com/software/info/web20

Resources

- The Online Informix Information Center
 - publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp
- IBM Informix DeveloperWorks Technical Articles
 - www.ibm.com/developerworks/db2/products/informix/index.html
- IBM DeveloperWorks Informix Blogs
 - www.ibm.com/developerworks/blogs/page/roundrep (Informix Replication)
 - www.ibm.com/developerworks/blogs/page/gbowerman (Informix Application Development)
 - www.ibm.com/developerworks/blogs/page/idsteam (Informix Experts Blog)

Appendix B - Extensibility

- New Functionality in Informix 11
- Informix DataBlades
- Informix Bladelets
- Web Feature Service
- Example – Making QUARTER datablade available
- Informix Extensibility Development Considerations
- Resources

New Functionality in Informix 11

- Extends support for UDRs in cross-database and cross-server distributed operations to most contexts where a UDR is valid in the local database
- Extends the data types that are valid as parameters or return values of cross-server UDRs, which were formerly restricted to non-opaque built-in SQL data types, by supporting these additional data types:
- These data types can be returned by SPL, C, or Java language UDRs that use these data types as parameters or as return values, if the UDRs are defined in all the participating databases
- Any implicit or explicit casts defined over these data types must be duplicated across all the participating Informix instances
- The DISTINCT data types must have exactly the same data type hierarchy defined in all databases that participate in the distributed query.

User-Defined Data Types - *Opaque*

1. Create the C / Java data structure to represent the internal data structure
2. Write the support functions in C / Java
3. Register the opaque data type with the "create opaque type" statement

```
create opaque type type_name (  
    internallength = length,  
    alignment = num_bytes);
```

Note: length is in bytes, alignment = 1,2,4,8 bytes (default = 4)

```
create opaque type type_name (  
    internallength = variable,  
    maxlen = length);
```

Note: default length = 2 KB, max value = 32 kb

4. Register the support functions with the "create function" statement

```
create opaque type  
    my_type(internallength=8,  
            alignment=4);  
create function  
    support_in(lvarchar)  
    returning my_type with (not  
        variant);  
    external name  
        "/funcs/my_type.so"  
    language C  
end function;
```

```
create implicit cast (lvarchar as  
    my_type with support_in);
```

5. Grant access to the opaque data type and support functions
6. Write any user-defined functions needed to support the opaque data type
7. Provide any customized secondary-access methods for creating indexes

Informix DataBlades

- Basic Text Search (BTS) DataBlade
- Spatial DataBlade
- Geodetic DataBlade
- TimeSeries DataBlade
- Excalibur DataBlade
- Binary DataBlade
- MQ Series DataBlade
- Node DataBlade
- NAG DataBlade
- Large Object Locator DataBlade
- C-ISAM DataBlade
- Image Foundation DataBlade
- Video Foundation DataBlade
- Web DataBlade

Basic Text Search (BTS) DataBlade



- Basic Text Search (BTS) is a DataBlade module which supports searching of words and phrases within unstructured text in a column of a table
- Basic Text Search is FREE - supplied with Informix 11
- The BTS module uses the open source CLucene text search package
 - CLucene is a C++ port of Apache Lucene: the high-performance text search engine written in Java
 - A [case insensitive](#) search syntax common across all Lucene open source implementations
- Simple to setup and use
- BTS employs various methods as search criteria including wildcard matching, Proximity searches, Fuzzy logic, Range searches, Rating Score produced for ordering of results

Basic Text Search DataBlade - Components

- Three main components:
 - The **bts_contains()** search predicate for queries
 - The **BTS DataBlade functions** for compacting indexes and tracing
 - New **bts_index_fields() function** returns a list of elements that can be searched on
- New Stopword support
 - Stopwords are a list of words that are not indexed
 - There is a default list of stopwords (in lowercase) are based on English:
 - a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, s, such, t, that, the, their, then, there, these, they, this, to, was, will, with
 - The stopwords parameter specifies a different list of stopwords to be used instead of the default list

Basic Text Search DataBlade - Setting up BTS

- Register the blade for your database using BladeManager

```
bash-2.03$ blademgr  
myserver>register bts.1.00 mydb  
Register module bts.1.00 into database mydb? [Y/n]y  
Registering DataBlade module... (may take a while).  
DataBlade bts.1.00 was successfully registered in database mydb.  
myserver>quit  
Disconnecting...
```

- **NOTE:** Run as **informix**, or set **IFX_EXTEND_ROLE** to 0
- **NOTE:** On Windows use Blade Manager supplied with DBDK

Basic Text Search DataBlade - Preparing BTS

- Define a bts Extension Virtual Processor Class
 - `VPCCLASS bts,noyield,num=1`
 - Only 1 bts EVP supported
- Create an **extspace** for the bts Index

```
mkdir /work/bts_extspace  
onspaces -c -x bts_extspace -l /work/bts_extspace
```

- Create an index by specifying the btsaccess method.

Basic Text Search DataBlade – Creating a BTS Index

- Example BTS CREATE INDEX statement

```
CREATE INDEX cust_bts  
ON work_items(notes bts_varchar_ops)  
USING bts (delete='immediate')  
IN bts_extspace;
```

- **bts_varchar_ops** is an "Operator Class" to support the VARCHAR data type
- *delete='immediate'* - optional parameter indicating delete index information when row is deleted. The default is *'deferred'* which marks docs as deleted until the **bts_index_compact()** UDR is run.

Basic Text Search DataBlade – Example BTS Query

```
SELECT work_idx, notes  
FROM work_items  
WHERE  
bts_contains(notes,"(win* OR nt) AND hang~")
```

- In this example *win** means any word beginning with "win"
- *AND/OR* are logical predicates (not search words)
- *hang~* means a fuzzy search for words like "hang"

Basic Text DataBlade (BTS) Module Enhancements

- Obtain a list of fields in an index to search on with the new `bts_index_fields()` function
- Specify a custom stopwords list with a new index parameter instead of using the default stopwords list
- Index XML documents with new index parameters
- Supports searches on XML attributes in a document repository
- Perform searches on high-availability cluster servers by creating indexes in sbspaces
- Specify the degree of similarity of search results in fuzzy searches (number between 0 and 1)
- Map characters in data to other characters during indexing
- Change the default Boolean operator between search terms from OR to AND
- Specify that temporary files are to be stored in a separate sbpace from the one used to store the bts index
- Track what queries are run against a bts index by including the `query_log` parameter
- Fragment bts indexes by expressions into multiple sbspaces instead of a single sbpace

Spatial DataBlade

- Brings all the significant features and benefits of the Informix to location-based data
- Many key business decisions involve location and proximity. The Spatial DataBlade enables organizations to transform both traditional and location-based data into important information to help gain a competitive advantage.
- Expands Informix object-relational data server to provide industry leading SQL-based spatial data types and functions, that can be used directly through standard SQL queries or with client-side Geographic Information Systems (GIS) software (such as that from ESRI and MapInfo).
- Enables organizations to intelligently manage complex geospatial information alongside traditional data without sacrificing the efficiency of the relational database model.
- Built-in Informix R-tree multidimensional index to provide industry-leading spatial query performance.

Spatial DataBlade – Why and How

- Why Spatial?
 - 80% of corporate data has a geographic component
 - Good way to understand patterns, deliver services, manage assets, report results
- Integrated R-tree index
- New data types:
 - ST_Point, ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon
- New functions
 - ST_Equals, ST_Disjoint, ST_Intersects, ST_Touches, ST_Within, SE_Nearest, etc.
- Simple SQL:

"List the parcels that overlap a flood plaine area"

```
SELECT a.member, b.member FROM parcel a, flood100 b  
WHERE st_overlap(a.geometry, b.geometry);
```



Spatial DataBlade – Spatial is Everywhere!

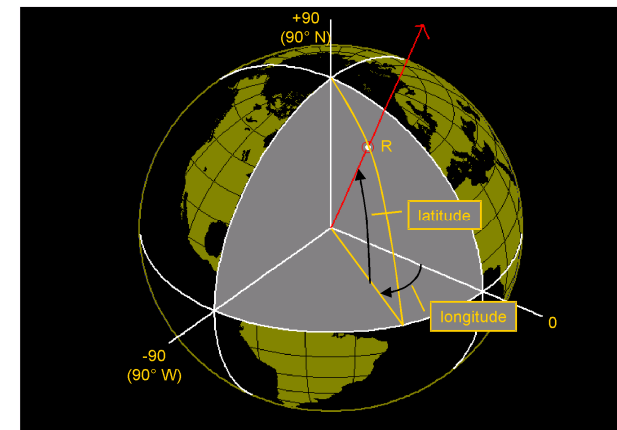
- Where are my stores located related to my distributors?
- How can I efficiently route my delivery trucks?
- How can I micromarket to customers fitting a particular profile near my worst performing store?
- How can I set insurance rates near to flood plain?
- Where are the parcels in the city that are impacted by a zoning change?
- Which bank branches do I keep after the merger based on my customers locations (among other things)?

Geodetic DataBlade

- Provides the ability to manage geospatial information referenced by latitude-longitude coordinates -- supporting global space- and time-based queries without limitations inherent in map projections
- Manages spatial data using Geographic Information Systems (GIS) technologies
- Ensures precision and accuracy -- engineered (from inception) to treat Earth as a globe, not a flat plane
- Uses the R-tree index on integrated space, time and numeric dimensions for lightning performance
- Includes database replication of geospatial data
- Provides a robust, well-crafted C-language application programming interface (API), so you can build new functions that use the same data structures and internal interfaces used by SQL functions already provided

Geodetic DataBlade - Global

- Specialized spatial datablade module:
 - "GeoSpatial objects consist of data that is referenced to a precise location on the surface of the earth"
- Based on the Hipparchus library
 - Geodyssey Ltd., Calgary, AB
- Altitude and time range dimensions
- Applications:
 - **Global, Polar, Trans-Pacific, High Accuracy**
- Basic computations: Complex, Expensive
- Example:



Find all the atmospheric elements that were present in the Denver area during the last thunderstorm

Comparison of Geodetic and Spatial DataBlades

- **Geodetic DataBlade**

- Treats the Earth as a globe
- Uses a latitude and longitude coordinate system on an ellipsoidal Earth model
 - Geometric operations are precise regardless of location
- Includes Server-Side SQL API & Client-Side Java API
- Based on Hipparchus library by Geodyssey Limited
<http://www.geodyssey.com>
- Best used for global datasets and applications, such as satellite imagery repositories

- **Spatial DataBlade**

- Treats the Earth as a flat map
- Uses planimetric (flat-plane) geometry
 - Approximates the round surface of the Earth by projecting it onto flat planes using various transformations
- Integrated with ESRI GIS software programs: ArcView GIS, MapObjects , and ArcInfo
- Implements the Open GIS Consortium, Inc. (OpenGIS, or OGC) SQL3 specification of abstract data types (ADTs)
- Best used for regional datasets and applications

TimeSeries DataBlade



- Time series information is data collected as it varies over time
- TimeSeries DataBlade Module enables for storage and manipulation of this time stamped data
- Data is stored in a user defined row type which can include additional data in addition to the timestamp
- Includes a rich set of analysis routines built into the server
- Real Time Loader (RTL) is very high performance, in memory, TimeSeries data loading tool
 - Leverages the Informix-NAG Financial DataBlade for superior analysis of real-time data. Makes time-stamped data available in real time
 - Provides instant visibility to real-time data
 - Analyzes complex data to facilitate well-informed business decision
 - RTL designed to cope with the first 15 minutes on Wall Street

TimeSeries DataBlade – Storage and Performance

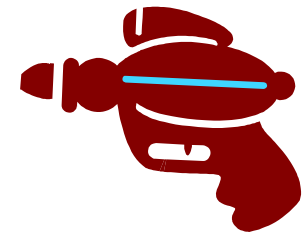
- TimeSeries DataBlade optimizes storage usage
 - 50% Savings not uncommon
- Optimized Access Time
 - 10 times performance improvement typical
- Calendars
 - Defines period of time that data may or may not be collected
- SQL, Java, and C interfaces
- VTI Interface
 - Makes a time series look like a regular table
- Office Connect Interface
 - Web enables Microsoft Excel plug-in

TimeSeries DataBlade - Who's interested?

- Capital Markets
 - Arbitrage opportunities, breakout signals, risk/return optimization, portfolio management, VaR calculations, simulations, backtesting...
- Telecommunications:
 - Network monitoring, load prediction, blocked calls (lost revenue) from load, phone usage, fraud detection and analysis...
- Manufacturing:
 - Machinery going out of spec; process sampling and analysis
- Logistics:
 - Location of a fleet (e.g. GPS); route analysis
- Scientific research:
 - Temperature over time...

Excalibur DataBlade

- Text search engine specifically designed to perform sophisticated and fast text searches
 - Full text indexing
 - Fuzzy logic searches
 - Multiple stop word lists
 - Proximity searches
 - Synonym lists
- Support for documents in ASCII, Microsoft Word, Excel, Powerpoint, HTML, PDF, Word Perfect, and other formats
- Adaptative pattern recognition process engine for maximum precision, recal, and relevancy
- Multi stop word lists, proximity searching, synonym lists
- New Installer provided by InstallAnywhere 8

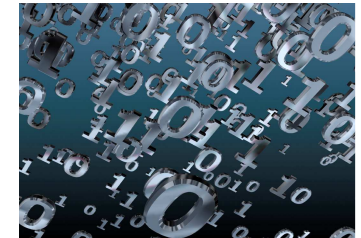


Excalibur DataBlade – Vector Indexing

- Feature-vector extraction search techniques
- Multiple image file formats are supported
- Image-based searching of visual data by color, shape or texture
- Indexing capabilities Excalibur's Visual Retrieval Ware
- Ability to read index with dirty read isolation level
 - Writers to ETX index no longer block readers!
 - Reads the “SIM” in memory
- Ability to perform ETX queries on HDR/SDS/RSS nodes
 - Required Informix to instantiate a new row type during the query, so this involves a new datatype cache
- Reduced memory footprint during transactions

Binary DataBlade

- Includes the *binary18* and *binaryvar* data types that allow for storage of binary-encoded strings, which can be indexed for quick retrieval
- Includes string manipulation functions to validate the data types
- Logical operators also supported
 - AND (bit_and), OR (bit_or), XOR (bit_xor) & NOT (bit_compliment)
- Binary data types are unstructured types
 - Can store many different types of information including IP addresses, MAC addresses, or device identification numbers from RFID tags
- Implicit casting of ASCII HEX representation
 - Ex. IP addresses like xxx.xxx.xxx.xxx as a CHAR(15) data type can be stored as a *binaryvar* data type, which uses only 6 bytes
 - Saves disk space!
- Maximum length 255 bytes



MQ Series DataBlade

- Interface between the Informix and applications using IBM WebSphere MQSeries
 - Seamless communication between MQ Series and Informix
- Enables communications between applications or between users and a set of applications on dissimilar systems
- Supported on over 30 platforms
- Comprised of several user-defined routines (UDRs) that communicate using the WebSphere MQ integration software
 - The DataBlade provides access to MQSeries queues from SQL functions or tables
 - Using SQL to interface to these queues results in faster application development and integration times
 - MQSubscribe(), MQUnsubscribe()
 - MQSend(), MQRead()
 - MQReceive()
 - MQPublish()



Node DataBlade

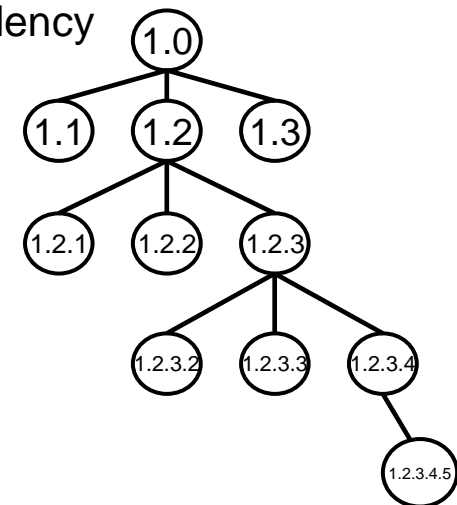
- Used to model hierarchies and networks
- Typical industry applications: Manufacturing and Process Control
- Improves query performance for many recursive queries
- Resolves a traditional RDBMS problem - transitive closure
- Eases the burden of transitive dependency in the relational database model
 - *Transitive dependency* occurs when a non-key attribute is dependent on another non-key attribute – as seen in manufacturing and process controls
 - This relationship frequently has multiple levels of attribute dependency
- Example :

```
CREATE TABLE nodetab1 (col1 node);  
INSERT INTO nodetab1 VALUES ('1.0');  
INSERT INTO nodetab1 VALUES ('1.1');  
INSERT INTO nodetab1 VALUES ('1.1.1');
```

```
SELECT col1 FROM nodetab1 WHERE isAncestor(col1, '1.1.2');
```

col1 1.0

col1 1.1



NAG DataBlade

- NAG: Numerical Analysis Group
 - Over 30 years experience
 - Experts in numerical and statistical computation
 - Reputation for accuracy and performance
- Business functions: convert Timeseries data to vectors and vectors of returns; present value calculations; ErlangB (network) calculations; data export functions
- NAG Fortran functions: correlation and regression analysis; variance-covariance matrix generation; optimization; eigenvectors, eigenvalues; ...
- Examples:
 - Oil industry: temperature variability down an oil well
 - Finance: calculate volume weighted average price and volatility
 - Spatial routing: Find the shortest path between two points

Large Object Locator DataBlade

- Manages large objects inside or outside the database
- Uses two new types
 - lld_locator: row type identifying the object and its location
 - lld_lob: binary or character large object
- SQL interface functions
- API interface
- ESQL/C interface

C-ISAM DataBlade

- A library of C-language functions that manage indexed sequential access method (ISAM) files efficiently
- SQL Access
 - Provides an fast SQL interface to the C-ISAM data
 - Uses the Virtual Table Interface (VTI)
- Server Storage
 - Provides the capability to store ISAM data directly in the database server while allowing C-ISAM programs to continue accessing this data.
 - Requires recompilation of C-ISAM applications with a new client library

Image Foundation DataBlade

- Provides a base on which new or specialized image types and image processing technologies can be quickly added or changed
- Store and retrieve images in the database or on remote computers and storage servers
- Store and retrieve image metadata
- Transform images using the industry-standard CVT command set
- Formats supported:
 - TIFF, FlashPix, JPEG, PNG, GIF, PhotoShop

Video Foundation DataBlade

- Incorporates specific video technologies into complete database management applications
- Manage video content and metadata
- Actual video content can be maintained on disk, video tape, video server, or other external storage devices
- Extends the capabilities of Informix database server to manage video content and metadata, or information about the content
- Platforms: HP-UX, SunOS

Web DataBlade

- Collection of tools, functions, and examples that ease development of "intelligent", interactive, Web-enabled database applications.
- Supports most Web Server Application Programming Interfaces (APIs), and enables a truly interactive Web site.
- Webdriver database client application build SQL queries that execute the WebExplode function for retrieving AppPages from your database.
- Enables you to customize Web applications using information from its configuration file, or stored in the database-- without gateway programming.
- Allows you to track persistent session variables between AppPages

Web Feature Service (WFS)



- Provides a generic way to access raw geographic data over the web
- Based on the Transactional WFS (WFS-T) specification from the Open Geospatial Consortium (OGC)

www.opengeospatial.org

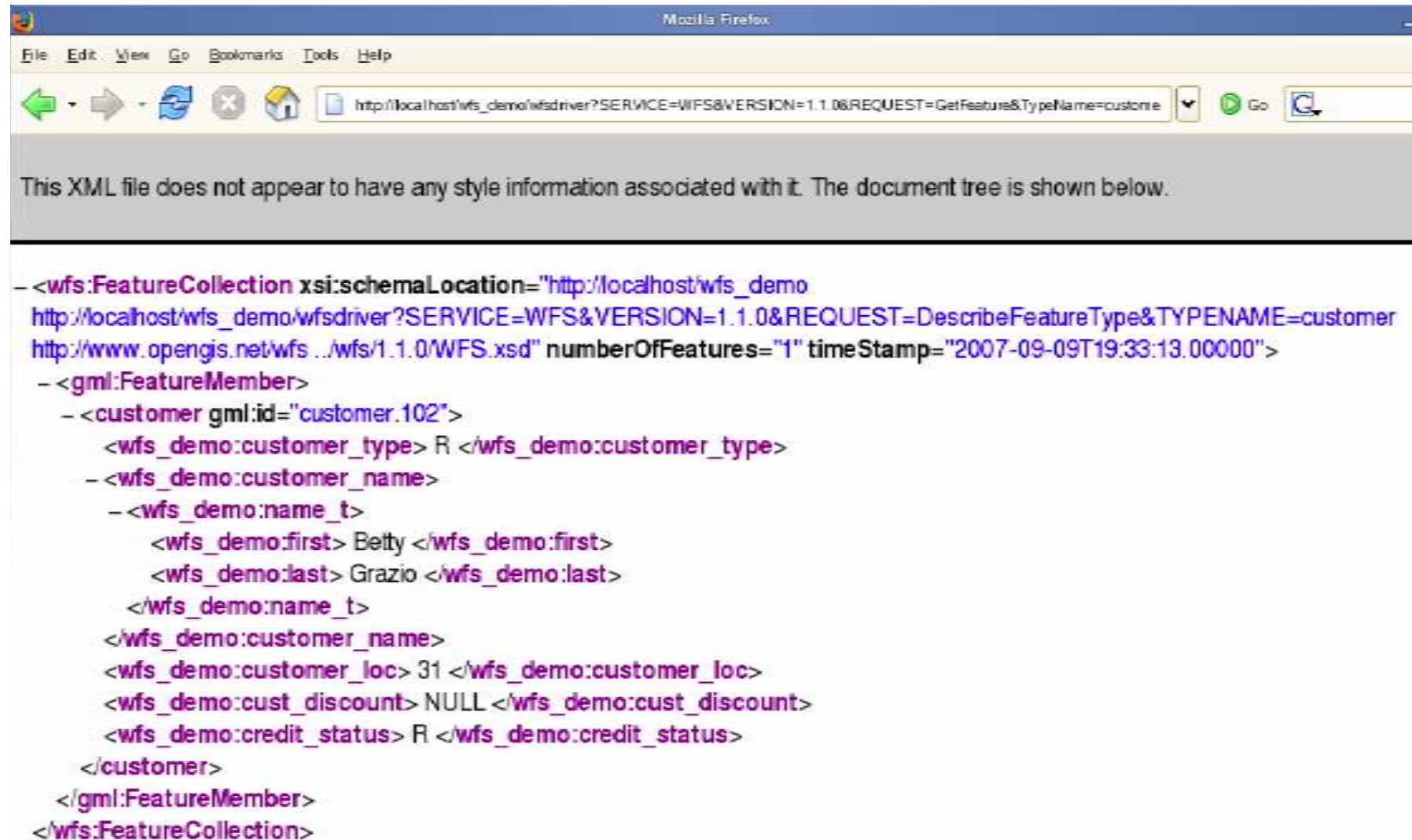
- An interface for web-based requests using platform-independent calls
- Uses HTTP GET or POST methods encoded as key-value-pairs (KVP) or XML for requests and responses
- Geographical features are encoded in the platform-independent, XML-based geography markup language (GML)
- Works with the Spatial DataBlade or Geodetic DataBlade modules to enable Informix to manage geographical features

Existing Bladelets – FREE!

Bladelet	Description
Exec	Provides dynamic SQL functionality with an SPL procedure
JPEG Image Bladelet	Provides a UDT for JPEG images so that you can manipulate images and extract and search on image properties
Flat File Access Method	A complete access method that lets you build virtual tables based on operating system files
Regex	Creates routines so that you can manipulate character and CLOB data by using regular expressions
Shapes	Creates several opaque types for managing simple spatial data, including R-tree index support.
XSLT	Creates new SQL functions that allow transformation of XML documents from one format to another using XSLT style sheets.
Multirepresentational varchar Opaque Type	Creates the ids_mrLvarchar opaque type, which stores character data up to 2 GB
SqlLib	SqlLib is a Bladelet that adds several other database compatibility functions to Informix. Versions are implemented in both Java and C languages

www.ibm.com/developerworks/db2/zones/informix/library/samples/db_downloads.html

Web Feature Service (WFS) – Web Tags



```

- <wfs:FeatureCollection xsi:schemaLocation="http://localhost/wfs_demo
http://localhost/wfs_demo/wfsdriver?SERVICE=WFS&VERSION=1.1.0&REQUEST=DescribeFeatureType&TYPENAME=customer
http://www.opengis.net/wfs/1.1.0/WFS.xsd" numberOfFeatures="1" timeStamp="2007-09-09T19:33:13.000000">
  - <gml:FeatureMember>
    - <customer gml:id="customer.102">
      <wfs_demo:customer_type> R </wfs_demo:customer_type>
      - <wfs_demo:customer_name>
        - <wfs_demo:name_t>
          <wfs_demo:first> Betty </wfs_demo:first>
          <wfs_demo:last> Grazio </wfs_demo:last>
        </wfs_demo:name_t>
      </wfs_demo:customer_name>
      <wfs_demo:customer_loc> 31 </wfs_demo:customer_loc>
      <wfs_demo:cust_discount> NULL </wfs_demo:cust_discount>
      <wfs_demo:credit_status> R </wfs_demo:credit_status>
    </customer>
  </gml:FeatureMember>
</wfs:FeatureCollection>
  
```


Example – Making QUARTER available

- Write the code
- Create the shared library
- Register the Function
- Install in the \$INFORMIXDIR directory (not shown)
- Use the quarter function

QUARTER Function code

```
#include <mi.h>

mi_lvarchar *quarter(mi_date date, MI_FPARAM *fparam)
{
    mi_lvarchar *RetVal;          /* The return value. */
    short      mdy[3];
    mi_integer  ret, qt;
    mi_char     buffer[10];

    ret = rjulmdy(date, mdy); /* Extract month, day, and year from the
                               date */

    /* calculate the quarter */
    qt = (mdy[0] - 1) / 3;
    qt++;
    sprintf(buffer, "%4dQ%d", mdy[2], qt);
    RetVal = mi_string_to_lvarchar(buffer);

    /* Return the function's return value. */
    return RetVal;
}
```

Create the Shared Library

- Compilation:

```
cc -xs -I$(INFORMIXDIR)/incl/public -c -o quarter.o quarter.c
```

- Linking into the target directory:

```
ld -G -o $(INFORMIXDIR)/extend/quarter/quarter.bld quarter.o
```

```
chmod a+x $(INFORMIXDIR)/extend/quarter/quarter.bld
```

- On Windows, the commands become:

```
link ... /out:... /def:... . . .
```

```
attrib +R $(INFORMIXDIR)/extend/quarter/quarter.bld
```

Register the QUARTER Function

```
CREATE FUNCTION quarter(date)
RETURNING varchar(10)
WITH (NOT VARIANT, PARALLELIZABLE)
EXTERNAL NAME
"$INFORMIXDIR/extend/quarter/quarter.bld(quarter)"
LANGUAGE C

END FUNCTION;

DROP FUNCTION quarter(date);
```

Use of the QUARTER Function

```
EXECUTE FUNCTION quarter("09/02/2002"::DATE);  
2002Q2
```

```
SELECT quarter(my_date) FROM tab  
WHERE id = 7;
```

Informix Extensibility Development Considerations

- Development Methodology
- Development Steps
- Development Environment

Development Methodology

- Work with the strength of the relational model
 - Set processing
 - Comparing, grouping, sorting
- Analyze the problem domain
 - Document high-level structures
 - List data types: state and behavior
- Design and implement
 - Define types, functions
 - Combine objects into schema table/views

Development Steps

- Use DBDK to generate a skeleton of the DataBlade
- Decide where to run each function: CPU VP or EVP
- Testing/Debugging

Development Environment

- SPL
 - Standard SPL environment
- C
 - Include directory: `$INFORMIXDIR/incl/public`
 - Makefile generated by DBDK
 - DBDK Include file for Makefile:
`$INFORMIX/incl/dbdk/makeinc.<platform>`
- Java
 - Server configuration: onconfig parameters (see release notice)
 - `$(INFORMIXDIR)/extend/krakatoa/krakatoa.jar`

Complex Data Types - *Collections*

- Grouping of elements of the same data type (max size = 32)
- Used when the data is meaningless without the context of the other members in the collection (golf scores, to-do list)
- Three kinds of collections:

SET - unordered, no duplicates allowed

set {"apple", "orange", "grapefruit", "plum"}

MULTISET - unordered, duplicates allowed

multiset {"apple", "orange", "grapefruit", "apple", "plum",
"grapefruit"}

LIST - ordered, duplicates allowed

list {"apple", "orange", "grapefruit", "apple", "plum",
"grapefruit"}

Complex Data Types - *Collections*

- Create a table to work with:

```
create table class
(
  class_id serial,
  class_name varchar(60),
  description lvarchar,
  prereqs set(char(20)
              not null)
)
```

- Insert syntax similar to Row:

```
insert into class values
(
  300,
  "Performance and Tuning",
  "Covers advanced information
  on tuning Informix",
  (SET{"RDD","BSQL"})
)
```

- Use the */N* keyword to query values in a collection

```
select * from class where ("ASQL") in prereqs;
```

- Cannot update one element in a collection, must replace whole collection:

```
update class set prereqs = (set{"RDD","ASQL","BSQL"}) where id = 300;
```

User-Defined Data Types - *Opaque*

- Data type that is unknown to the database server
- User must define the internal structure, functions, and operations (C, C++, or Java)
- Stored in its entirety by the engine without any interpretation of the contents or its structure
- Access to an opaque type is through user defined functions
- User defines the storage size of the data type and input and output routines
- **Must register Opaque data type and supporting functions**

Virtual Table/Index Interface (VTI/VII)

- Virtual Table Interface (VTI)
 - Interface to provide a primary access method for developing gateways to data that is not stored in Informix tables
 - Data may be in flat files, in competitors' databases, cached in memory, or out on the net
 - To the end user, it looks exactly as if it's stored in Informix tables whereas it's in "virtual" tables
 - Allows Informix to define access methods for searching, reading, inserting, updating, and deleting data from these 'tables'
- Virtual Index Interface (VII)
 - Interface for implementing a secondary access method in an external or specialized data source
 - Set of definitions and function hooks that allow for the definition of new index methods

Functional Indexes

- All keys derive from the results of a function done on the columns of the rows
- Result is not actually stored in the table, but it is pre-computed and used to build an index
- Result can be a compressed version of the original columns or some calculation based on them
- Can be a B-tree index, an R-tree index, or a user-defined index type

R-Tree Indexes

- For columns that contain spatial data such as maps and diagrams
- Uses a tree structure whose leaf nodes refer to the bounding boxes of the elements
- Elements can be spatial data, time periods, etc
- At the leaves of the R-tree are a collection of data pages that store n -dimensional shapes

Bladelets – *All Free!*

- Small, informal modules meant to be useful "out of the box"
- Offered complete with source code at **no cost** -- and no support or warranty
- Many bladelets are available, such as:
 - An EXEC datablade that provides dynamic SQL functionality within a SPL procedure
 - A flat file access method datablade which allows for building virtual tables based on operating system files
 - A datablade that allows character and clob data manipulation using regular expressions
 - An image datablade that allows manipulation of JPEG images and extraction and search on image properties

Resources

- Informix DataBlade Modules

www.ibm.com/software/data/informix/blades

- Extensibility example - Date Processing in Informix

www.ibm.com/developerworks/db2/library/techarticle/dm-0510roy

- Extensibility example - Generating XML from Informix 9.x

www.ibm.com/developerworks/db2/zones/informix/library/techarticle/0302roy/0302roy2.html

- Extensibility example - Event-driven fine-grained auditing for Informix

www.ibm.com/developerworks/db2/library/techarticle/dm-0410roy

- Extensibility example - Using GUIDs with Informix 9.x

www.ibm.com/developerworks/db2/library/techarticle/dm-0401roy/index.html



Informix 11.5 Bootcamp Application Development Overview

Information Management Partner Technologies