



Information Management



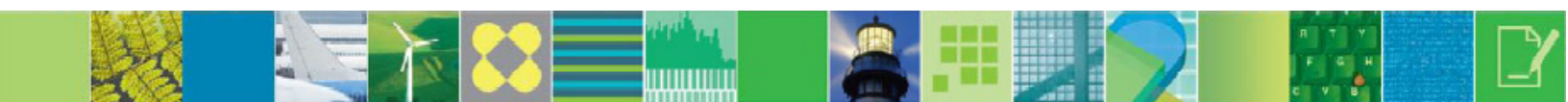
IBM® DB2® 9.7 Academic Workshop Course Workbook



IBM Canada Ltd.

Information Management
Ecosystem Partnerships

V-20100729



IBM[®] DB2[®] 9.7
Academic Workshop
Course Workbook

Information Management Ecosystem Partnerships
imschool@us.ibm.com

Contents

PREFACE	
WELCOME.....	3
RELATIONAL DATA MODEL.....	10
DB2 FUNDAMENTALS AND IBM DATA STUDIO.....	21
IBM DATA STUDIO LAB (HANDS-ON)	40
WORKING WITH DATABASES AND DATABASE OBJECTS	55
WORKING WITH DATABASES AND DATABASE OBJECTS LAB	74
INTRODUCTION TO SQL.....	104
UNDERSTANDING SQL LAB (HANDS-ON).....	128
DATA CONCURRENCY	162
DATA CONCURRENCY LAB (HANDS-ON)	176
DB2 DATABASE SECURITY	205
DB2 SECURITY LAB (HANDS-ON)	216
DB2 BACKUP AND RECOVERY	232
DB2 BACKUP AND RECOVERY LAB (HANDS-ON).....	243
DB2 PUREXML	257
DB2 PUREXML – STORING XML DATA MADE EASY LAB	275
DB2 PROGRAMMING FUNDAMENTALS	286
DB2 PROGRAMMING FUNDAMENTALS LAB (HANDS-ON)	303
APPENDIX I – VMWARE BASICS AND INTRODUCTION	327

Preface

Welcome to the **IBM DB2 9.7 Academic Workshop**! If you are reading this text, you are giving an important step towards building a successful career as an Information Technology professional.

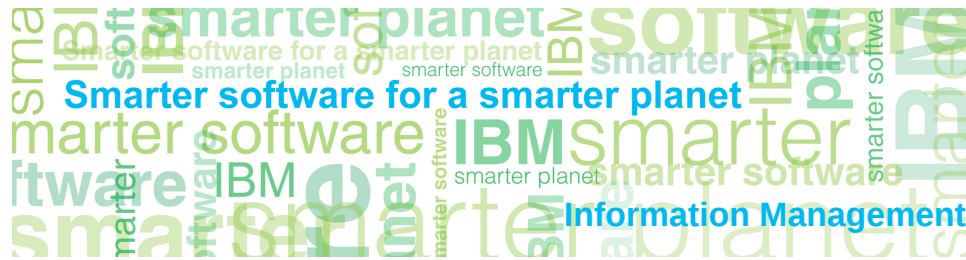
This course was specially designed for the academic community interested into expanding their skill set on the exciting field of relational databases. Either if you are a student taking your first steps into this area, or a member of the faculty, you are certain to learn something new from this material. Additionally, the provided hands-on laboratories, based on the latest version of **IBM® DB2® for Linux, UNIX and Windows**, close the gap between theory learned from the presentations, and real-world use of a Relational Database Management System (RDBMS).

IBM DB2 for Linux, UNIX and Windows is an industry-leading performance Relational Database Management System for mixed workloads that offers a wide range of advanced features. During this course you will be exposed to some of these features, understand the mechanics of relational databases and how DB2's features can be used to help increase productivity while lowering costs of maintaining databases.

The course starts with a general overview of databases and the relational model and moves on to introducing the DB2 environment and the ease of use of its tools. Next, you will learn about the various objects that are part of a relational databases and how to interact with them using SQL. Once you are comfortable with the basic of RDBMS, we explore more advanced features such as DB2 pureXML, which allows supports storage of XML documents and use of XML technologies such as XQuery and XML Schema. Afterwards, topics that every Database Administrator (DBA) should know are presented such as implementing security policy for access to data, backing up your database and understanding data concurrency. The course finishes by briefly exploring how applications can store and retrieve data from a DB2 server.

Welcome

DB2 9.7 Academic Workshop



© 2010 IBM Corporation



Disclaimer

© Copyright IBM Corporation 2010. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

IBM, the IBM logo, ibm.com, and DB2 are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

Agenda – Day 1

1.0 Welcome

1.1 The Relational Data Model

< Break >

1.2 DB2 Fundamentals and Data Studio

1.3 IBM Data Studio Lab (Hands On)

Lunch

1.4 Working with Databases and Database Objects

1.5 Working with Databases and Database Objects (Hands On)

< Break >

1.6 Introduction to SQL

1.7 Understanding SQL Lab (Hands On)

Summary

3

© 2010 IBM Corporation

Agenda – Day 2

2.1 Data Concurrency

2.2 Data Concurrency Lab (Hands On)

< Break >

2.3 Database Security in DB2

2.4 DB2 Security Lab (Hands On)

Lunch

2.5 DB2 Backup and Recovery

2.6 DB2 Backup and Recovery Lab (Hands On)

< Break >

2.7 DB2 pureXML

2.8 DB2 pureXML – Storing XML Data Made Easy Lab (Hands On)

Summary

4

© 2010 IBM Corporation

Agenda – Day 3

3.1 DB2 Programming Fundamentals

< Break >

3.2 Accessing DB2 Databases From Applications Lab (Hands On)

Summary

Lunch

3.3 Course Review

3.4 IBM Certified Academic Associate - DB2 9 Database and Application Fundamentals

5

© 2010 IBM Corporation

IBM Certified Academic Associate - 302 - DB2 9 Database and Application Fundamentals

Test : IBM Certified Academic Associate - DB2 9 Database and Application Fundamentals

- This is an entry level academic course for the DB2 9 products The certified individual is knowledgeable about the fundamental administration and development concepts of DB2 9.7.
- Exam breakdown:
 - Relational Data Model (15%)
 - DB2 Fundamentals (15%)
 - DB2 pureXML (10%)
 - Transactions on DB2 (5%)
 - Data Concurrency (5%)
 - Working with Database Objects (10%)
 - Database Security in DB2 (10%)
 - Backup and Recovery (5%)
 - DB2 Programming Fundamentals (10%)
 - Working with SQL (15%)

6

© 2010 IBM Corporation

DB2 Certification & Free Tutorials

- **Information Management Certification website:**

- www.ibm.com/software/data/education



- Step 1

- **IBM Certified Database Associate for DB2 9 Fundamentals, Exam 730**

- Exam info: <http://www-03.ibm.com/certify/tests/obj730.shtml>
- Free tutorial: <http://www.ibm.com/developerworks/offers/lp/db2cert/db2-cert730.html>

- Step 2 (for DBAs)

- **IBM Certified Database Administrator for DB2 9.7 DBA for LUW, Exam 541**

- Exam info: <http://www-03.ibm.com/certify/tests/obj541.shtml>
- Free tutorial (DB2 9): <http://www.ibm.com/developerworks/offers/lp/db2cert/db2-cert731.html>

- Step 2 (for Developers)

- **IBM Certified Application Developer for DB2 9.7, Exam 543**

- Exam info: <http://www-03.ibm.com/certify/tests/obj543.shtml>

- Step 2 (for z/OS)

- **IBM Certified Database Administrator for DB2 9 DBA for z/OS, Exam 732**

- Exam info: <http://www-03.ibm.com/certify/tests/obj732.shtml>

- Step 3

- **IBM Certified Advanced Database Administrator for DB2 9 DBA for LUW, Exam 734**

- Exam info: <http://www-03.ibm.com/certify/tests/obj734.shtml>

7

© 2010 IBM Corporation

IBM Guided Hands-on Technical Learning -Technical Education Bootcamps

- **Education**

- **DM Bootcamps**

- WW Bootcamps Available for:

- DB2 9.7 LUW & Migration Clinic
- DB2 pureXML
- Informix 11.5
- IBM InfoSphere Warehouse v9.7
- IBM Optim Solutions
- Guardium
- SolidDB
- InfoSphere Change Data Capture
- InfoSphere Information Server
- InfoSphere Master Data Management



2010 schedule, bootcamp agenda and registration available here:

www.ibm.com/developerworks/data/bootcamps/

8

© 2010 IBM Corporation

Individual Reading



• Reading Materials – printed

- www.ibm.com/software/data/education/bookstore
- Study Guides
 - **DB2 9 Fundamentals** 978-1-58-347072-5
 - **DB2 9 for Linux, UNIX, and Windows Database Administration** 158347-077-8
 - **DB2 9 for z/OS Database Administration** 978-158347-074-9
 - **DB2 9 for Linux, UNIX, and Windows Database Administration Upgrade** 158347-078-6
- Books
 - **DB2 9 for Linux, UNIX, and Windows – Sixth Ed.** 0-13-185514-X
 - **Understanding DB2: Learning Visually**
- Manuals: <http://www-01.ibm.com/support/docview.wss?rs=71&uid=swg27015148>
- **DB2 Information Center:** <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>
- **IBM Redbooks:** <http://www.redbooks.ibm.com/> with Examples 0-13-158018-3
 - **Understanding DB2 9 Security** 0-13-1345907
 - **DB2 9 for Developers** 978-158347-071-9



• Tutorials/Self-Study

- www.ibm.com/software/data/education/selfstudy.html

• Data Management Magazine – (Former IBM Database Magazine)

- <http://www.ibmdmmagazinedigital.com/>



• Performance Perspectives – Insights and ideas on Information on Demand

- <http://www-01.ibm.com/software/data/performance-perspectives/>

9

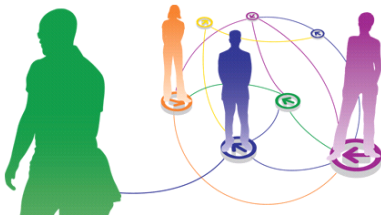
© 2010 IBM Corporation

Bringing it all Together -Information on Demand Conference



• IOD Global

- Education/Certification
- Business Prospecting
- Partner Networking
- IBM Relationships



10

© 2010 IBM Corporation

Access to Software

- DB2 Express-C

<http://www.ibm.com/db2/express/>



- Application Development Downloads
- IBM Data Studio: www.ibm.com/software/data/studio/
- DB2 Information Center
- DB2 Client
- Zend Core for IBM



- More Info: www.ibm.com/software/data/db2/ad/
- Download Via: www.ibm.com/db2/express/download.html
- Software Value Package:



https://www-304.ibm.com/jct09002c/partnerworld/mem/valuepack/mem_ben_value_resellers.html

DB2 Express-C vs. DB2 Express Fixex Term License (FTL)

Feature	DB2 Express-C Free (UNWARRANTED)	DB2 Express FTL* Paid Subscription
Core DB2 capabilities	YES	YES
Free admin tools	YES	YES
Free development tools	YES	YES
Autonomic capabilities	YES	YES
pureXML feature	YES	YES
No-charge community based assistance***	YES	YES
Official IBM 24x7 support	NO	YES
Fixpacks	NO	YES
High Availability (HADR)	NO	YES
Data Replication	NO	YES
Processor Usage Limit****	2 cores	4 cores (max 2 sockets)
Memory Usage Limit****	2 GB	4 GB
Update Availability	Full release at major milestones	Fixpacks generally every 3 months
Free access to old/current Product images	Yes, until major release only	Product Lifetime
Price per Server per Year**	\$ 0	\$ 2,995

* Features entitled with Subscription are available only while Subscriptions are valid

** Subscription Price indicated is for United States and subject to change. Pricing in other countries may vary.

*** No-charge community-based assistance is available via the online forum.

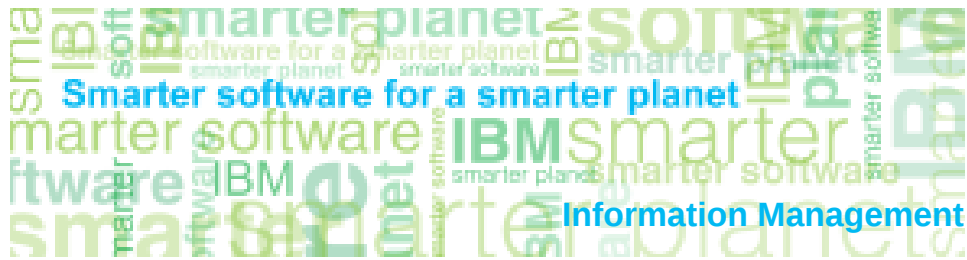
**** CPU and memory limitations for DB2 Express-C are not limitations of the machine size, rather they specify DB2 usage limits on those machines.

Questions?

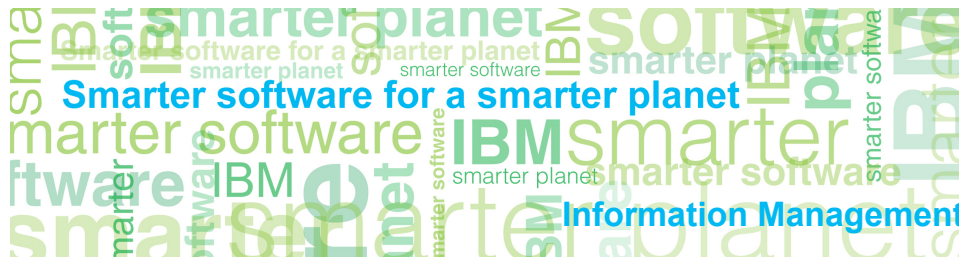
Summer/Fall 2010

E-mail: imschool@us.ibm.com

Subject: “DB2 Academic Workshop”



The Relational Data Model



© 2010 IBM Corporation



Agenda

- Data and Databases
- Database Management Systems (DBMS)
- Information Model & Data Model
- Relational Data Model & Relational Databases
- Normalization

What is Data?

- **Collection of facts or numbers**
- **Can be quantitative or qualitative**
- **Describes a variable or set of variables**
- **Essentially data can be thought of as the result of observations based on things like:**
 - measurements
 - statistics



3

© 2010 IBM Corporation

Data and Information

- **Data is simply facts**
- **Why is data important?**
 - By relating different pieces of data we are able to extract **valuable information** by presenting it in meaningful context
- **For that we need to be able to:**



- Store data → so it can be persisted
- Structure data → so it is easier to manipulate
- Organize data → in a meaningful way
- Process data → to derive data value from it

4

© 2010 IBM Corporation

Why Databases?

- **Data can be stored using multiple methods such as:**
 - Text files
 - Comma delimited data files
 - Spreadsheets
 - Databases
- **Why database?**
 - The way data is accessed
 - The way data is handled



5

© 2010 IBM Corporation

Managing Data

- **Using a database provides:**
 - a standard interface for accessing data
 - multiple users with simultaneous ability to insert, update and delete data
 - changes to the data without risk of losing data and its consistency
 - the capability to handle huge volumes of data and users
 - tools for data backup, restore and recovery
 - security
 - reduce redundancy
 - data independence

6

© 2010 IBM Corporation

Database Management Systems

▪ Database Management System (DBMS)

- It is the software system that manages databases
- Provides an interface of access to the databases
- Provides data services to applications
 - Efficient data querying and update mechanisms
 - Data integrity – guarantees data is always right even in case of software and hardware errors
 - Others: backup, compression, security, replication, etc.



▪ DB2 is a Database Management System

7

© 2010 IBM Corporation

Information Model

▪ Information Model

- Abstract management of objects at a conceptual level
- Independent of specific implementations and protocols
- Hides all protocol and implementation details
- Defines relationships between managed objects.

▪ Multiple implementations of an information model exists

- Data models

8

© 2010 IBM Corporation

Data Model

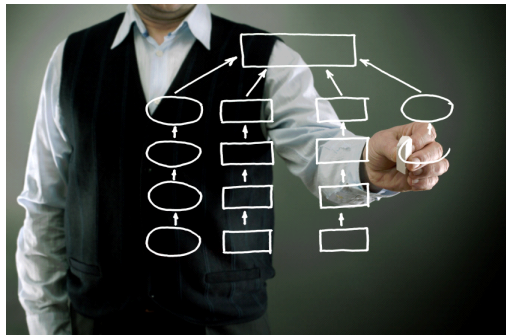
- **A model is a representation of an object or concept of the real world**
 - **3D model**: a graphical representation of an object
 - **Scale model**: a replica or copy of an object in a smaller size
 - **Business model**: describes how a company operates
- **Data Model**
 - Define how data is to be represented and structured
 - It can be used to map how data from the real world is to be represented in a software system
- **Characteristics of a Data Model**
 - Lower level of abstraction
 - Intended for the software developer
 - Includes specific implementation and protocol details

9

© 2010 IBM Corporation

Types of Data Models

- Extended Relational
- Entity-Relationship
- Hierarchical
- Network
- Object-oriented
- Object-relational
- **Relational**
- Semantic
- Semi-structured (XML)



10

© 2010 IBM Corporation

The Relational Data Model

- Proposed by E.F. Codd in 1970.
- It is mathematical model that describes data as a collection of **Relations** (sets) and the values of the data is defined by **Domains**.
- Focuses on providing better data independence
- Data are operated upon by means of a *relational calculus* or *relational algebra*
- Advantages
 - Based on a formal theoretical model and proven in practice
 - Provides logical view of the data
- It is implemented by most DBMS in the market, such as DB2.
 - There are called **Relational Database Management Systems**

11

© 2010 IBM Corporation

Components of the Relational Data Model

- The relational data model has its own unique terms used to define its concepts.

		Attribute			
		(ID, int)	(NAME, text)	(EXT, int)	(Active, boolean)
Relation	1	John S	54213	Y	Tuple
	2	Michael B	52137	Y	
	3	Jeremy W	50603	Y	
	4	Leah E	58963	N	

Domain

- **Domain (or data type)** defines the set of possible values that data can assume
- **Relation** is composed by a heading and a body
 - **Heading:** a set of attributes
 - **Body:** a set of tuples
- **Attribute** is composed by a name and a domain (type)
- **A tuple** is a set of attribute values

12

© 2010 IBM Corporation

Components of a Relational Database

- Concepts from Relational Data Model can be mapped to their implementation found in a Relational Database
- Relational databases store data using **tables**
 - A table consists of **columns** and **rows**
 - Each column has a specific **data type**
 - Each row features a certain **value** for each column

ID	NAME	EXTENSION	MANAGER
1	John S	54213	Y
2	Susan P	59867	N
4	Andrew J	55935	N
5	Michael B	52137	Y
6	Jeremy W	50603	Y
7	Leah E	58963	N

13

© 2010 IBM Corporation

Tables

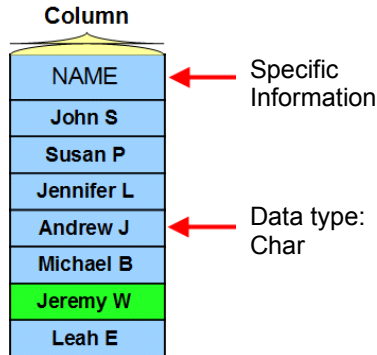
- **A Table is the counterpart of a Relation from the Relational Data Model**
- **A table stores data in rows and columns**
 - Rows are the same as Tuples
 - Columns are the same as Attributes
- **There can be multiple tables for different types of data to reduce redundant information**
 - Normalization (more on this later)
- **For example:**
 - You want to store data about a company
 - Data about branch offices will be stored in a table
 - Employee data for specific branches will be stored in its own table
 - Product data will be stored in another table

14

© 2010 IBM Corporation

Columns

- Columns are also known as fields
- Each field contains a specific type of information such as name, extension, position and so on
- Columns must be designated a specific data type such as DATE, VARCHAR, INTEGER and so on

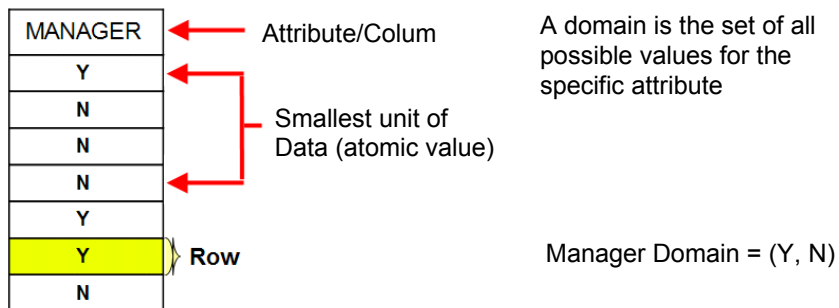


15

© 2010 IBM Corporation

Domains and Data Types

- Data Type**
 - This is counterpart of Domains in the Relational Model, which defines the smallest unit of data that can be stored
 - Columns always have a data type



16

© 2010 IBM Corporation

Components of a Relational Database

- **Primary Keys**

- Uniquely identifies each tuple (row) of the relation (table)
- Relations must always have a primary key
- Although it is recommended, tables in a relational database are not required to have a Primary Key

- **Examples**

- Driver's license of a person
- ISBN of a book
- Serial number of a product

Primary Key

ID
1
2
3
4
5
6
7

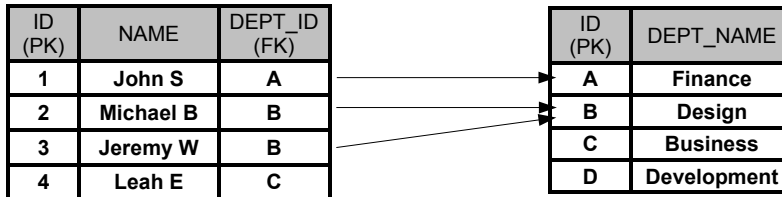
17

© 2010 IBM Corporation

Components of a Relational Database

- **Foreign Keys**

- Attribute in one relation whose values match a primary key of another relation
- Defines the relationship between two tables



18

© 2010 IBM Corporation

Why Normalization?

- **Why do we need normalization?**

- **Consider the following table:**

–Lists of task an employee is involved in:

ID	Name	Office City	Extension	Task
1	John S	Toronto	54213	Planning
1	John S	Toronto	54213	Marketing
1	John S	Toronto	54213	Testing
2	Susan P	New York	59867	Marketing
3	Jennifer L	Chicago	59415	Planning
3	Jennifer L	Chicago	59415	Testing

- **Example operation: if John moves to a new city, all entries related to John must be updated**

- redundancy
- anomalies

19

© 2010 IBM Corporation

Normalization

- **No anomalies, no redundancy**

- **No loss of information**

Employee Table

ID	Name	Office City	Extension
1	John S	Toronto	54213
2	Susan P	New York	59867
3	Jennifer L	Chicago	59415

Task Table

ID	Task
1	Planning
2	Marketing
3	Testing

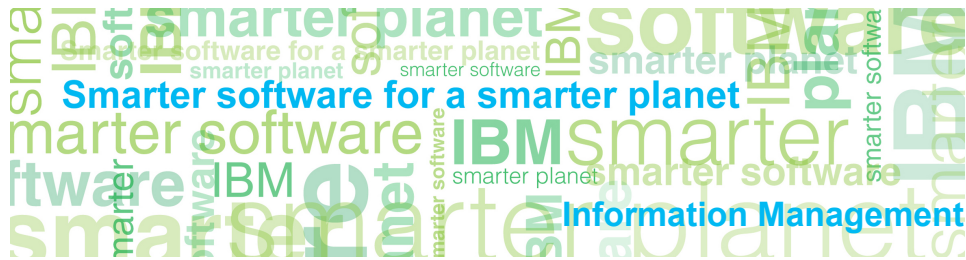
Employee Tasks Table

Employee ID	Task ID
1	1
1	2
1	3
2	2
3	1
3	3

20

© 2010 IBM Corporation

DB2® Fundamentals and IBM Data Studio



© 2010 IBM Corporation



Agenda

- **Product Overview**
 - Editions & Features
 - Licensing
- **Fundamentals**
 - Architecture
 - Users
 - Instances
 - DB2 Client
 - Storage
 - Table spaces
 - Buffer pools
 - Security
 - SQL and XQuery
- **Data Studio**
- **DSAC**

Break free with DB2



© 2010 IBM Corporation

DB2 Product Overview

- **Officially released June 2009**
 - FP1 released Dec 2009
 - FP2 tentative Q2 2010
- **Full Multi-Platform Support**
 - Linux, UNIX (AIX, HP-UX)
 - Windows 2000, 2003, 2008, XP, Vista, 7
 - Solaris
 - Beta: Express-C edition on MAC
- **Common code base “DB2 is DB2 is DB2”**
 - No need to port between platforms
 - New versions available on all platforms at the same time



DB2 LUW main site:

<http://www-01.ibm.com/software/data/db2/linux-unix-windows/>

3

© 2010 IBM Corporation

DB2 Packaging and Editions



4

© 2010 IBM Corporation

Editions: for Small and Medium Businesses

Express-C

- Free to develop, distribute, deploy
- Optimized to use up to **2 processor cores** and **2 GB memory**
- Available for Linux, Windows, Solaris (x64)
- Unsupported and without warranty
- Does not include replication services and high availability
- Includes pureXML



www.ibm.com/db2/express

5

Express

- Entry level, fully supported data server
- Optimized to use up to **4 GB memory** and **4 processor cores**
- Available for Linux, Windows
- Includes pureXML
- Available add-ons include **Performance Expert** and **High Availability Feature**
 - High Availability Feature provides 24 x 7 continuous availability for your DB2 data server
- Fixed Term License (FTL) available, a 12-month subscription which includes HADR

© 2010 IBM Corporation

Editions: for Larger Enterprises

Workgroup Server

- Designed for larger workloads than DB2 Express
- Limited to **16 GB Memory** and **16 processing cores** or **4 sockets**
- Available for Linux, UNIX, Windows
- Identical to DB2 Express, but includes **High Availability Feature Pack** (TSA, HADR and Online Reorg)
- Available add-ons include **Performance Expert**

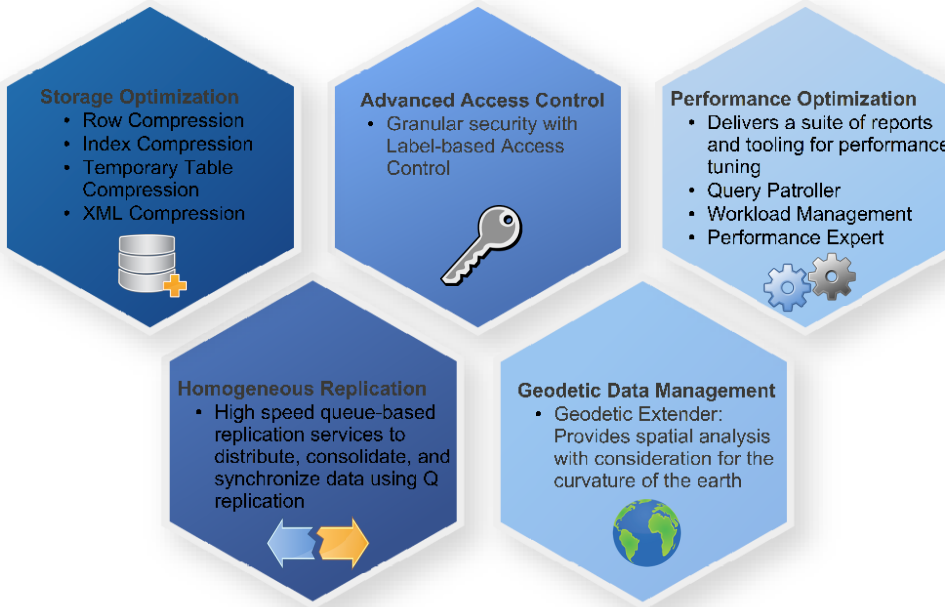
6

Enterprise Server

- Designed for heavy workloads, large data warehouses
- **No memory usage and processor core limits**
- Available for Linux, UNIX, Windows, zLinux
- Includes **pureXML, HADR, Online REORG, Homogenous Federation, DB2 Governor, MQT, MDC, Query Parallelism, Connection Concentrator, Table Partitioning.**
- Advanced features are available as add-ons

© 2010 IBM Corporation

Add-on Features for Enterprise Edition



7

© 2010 IBM Corporation

DB2 Features and Functionality by Edition

Features	Express Edition (FTL)	Express Edition	Workgroup Server Edition	Enterprise Server Edition
Label-based access control (LBAC)	No	No	No	DB2 Advanced Access Control feature
Geodetic Extender	No	No	No	DB2 Geodetic Data Management feature
Compression: row level	No	No	No	DB2 Storage Optimization feature
Workload management	No	No	No	IBM DB2 Performance Optimization Feature for Enterprise Server Edition
Query Patroller	No	No	No	
Performance Expert	No	No	No	
Homogenous Q Replication	No	No	No	IBM Homogenous Replication Feature for DB2 Enterprise Server Edition
Connection concentrator	No	No	No	Yes
DB2 Governor	No	No	No	Yes
Materialized query tables (MQT)	No	No	No	Yes
Multidimensional clustering (MDC) tables	No	No	No	Yes
Query parallelism	No	No	No	Yes
Table partitioning	No	No	No	Yes
Advanced Copy Services	Yes	IBM DB2 High Availability Feature for Express Edition	Yes	Yes
High availability disaster recovery	Yes		Yes	Yes
Online reorganization	Yes		Yes	Yes
Tivoli® System Automation	Yes	Yes	Yes	Yes
Compression: backup	Yes	Yes	Yes	Yes
Homogenous Federation	Yes	Yes	Yes	Yes
Homogenous SQL Replication	Yes	Yes	Yes	Yes
Net Search Extender	Yes	Yes	Yes	Yes
pureXML® storage	Yes	Yes	Yes	Yes
Spatial Extender	Yes	Yes	Yes	Yes

8

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>

© 2010 IBM Corporation

Licensing

All DB2 Database editions can be purchased by:

- **Per authorized user**
 - License type: "Authorized User Option"
- **Per processor (priced by PVU)**
 - License type: "CPU Option"
- **Sub capacity pricing available**

Other options of pricing, depending on edition:

- **Per socket**
 - Workgroup edition
- **Per server**
 - Express edition

Check the type of license being used:

- 1) Use command **db2licm -l**
- 2) Licensing center

9

© 2010 IBM Corporation

Licensing:: Per Authorized User

Authorized user:

single individual with a specific identity within or outside your organization

- **IDs cannot be shared or transferred (unless change in employment status)**
- **ID can establish one or more connections to the DB2 database system and counts as a single authorized user**
- **ID is needed for each data server. Single user connecting to two data servers would need two authorized user licenses**
- **Minimum number of users required for various editions**
Eg: DB2 Express Edition and DB2 Workgroup Server Edition each require a minimum of five authorized users for each server. Enterprise Server edition requires min. 25 Aus per 100 PVUs.



10

© 2010 IBM Corporation

Licensing: Processor Value Unit Pricing

Processor value unit (PVU):

a unit of measure that is assigned to each processor core

- **Sub-capacity Licensing: Enables the licensing of DB2 to a subset of the processor cores on the server**
- **Value defined by processor vendor, brand, type and model number**
- **Allows unlimited users to access DB2 on that server**

PVU licensing for Distributed Software

http://www-01.ibm.com/software/lotus/passportadvantage/pvu_licensing_for_customers.html



x86

Processor Vendor	Processor Technologies			Processor Model Number ¹	PVUs per Core
	Processor Brand	Maximum number of sockets per server	Processor Type		
Intel®	Xeon® (Nehalem-E) ²	≥ 4	Core i7	7500 to 7599	120
		4	Core i5	6500 to 6599	100
	Xeon® (Nehalem-EP)	2	Core i7	3400 to 3599	70
		2	Core i5	5500 to 5999	70
Xeon® (pre-Nehalem)	All	Core i7	3000 to 3399	50	
			5000 to 5499	50	
AMD	Opteron	All	Core i7	7000 to 7499	50
Any	Any single-core	All	Core i7	All	100

¹ Requirements as of Publish Date: Apr 2, 2010

RISC and System z

Processor Vendor	Processor Technologies			Processor Type			Processor Model Number	PVUs per Core
	Processor Brand	Server model numbers	Maximum number of sockets per server	Cores per socket	Processor Type	Processor Type		
IBM	POWER7 ¹	770-780	8	■	■	All	120	
		750-755	4	■	■	All	100	
	POWER6	550-560, 570, 575-595	All	■	■	All	120	
		520	All	■	■	All	80	
		JS12, JS22, JS23, JS43	All	■	■	All	100	
	POWER5, POWER4	All	All	■	■	All	100	
	POWER5 OCM	All	All	■	■	All	50	
	System z10 ¹	All	All	■	■	All	120	
	System z9, z990, S/390 ^{1,2}	All	All	■	■	All	100	
	PowerPC 970	All	All	■	■	All	50	
PowerPC405 ^{1,2} , Cell BE ^{1,2,3,4}	All	All	■	■	All	30		
HP	Itanium® 1.2	All	All	■	■	All	100	
Intel®	P4-RISC	All	All	■	■	All	100	
Sun	SPARC64 V1, V11	All	All	■	■	All	100	
Fujitsu	UltraSPARC IV	All	All	■	■	All	100	
	UltraSPARC T2	All	All	■	■	All	50	
	UltraSPARC T1	All	All	■	■	All	30	
Any	Any single-core	All	All	■	■	All	100	

¹ Requirements as of Publish Date: Apr 2, 2010

Licensing: Per Server

Limited use virtual server (LUV server):

is a physical server OR a virtual server that is created by partitioning the resources available to a physical server



- **Only available for DB2 Express Edition**
- **Allows unlimited users to access DB2 on that server**
- **All instances cannot collectively exceed 4 processor cores and 4 GB of memory**

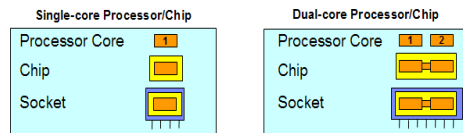


Licensing: Per Socket



Socket:
defined as electronic circuitry that accepts a processor chip

Processor chip:
the electronic circuitry that contains one or more processor cores and plugs into a socket



- Only available for **Workgroup Edition**
- 1 license for each socket on the server
- Allows unlimited users to access DB2 on that server
- Limit to 16 GB of memory and 4 sockets on a physical server
- Existing workgroup customers licensed by PVU can either
 - Trade-in PVU license for DB2 Workgroup per socket license at ratio of 100 PVUs per socket
 - Retain PVU license

13

© 2010 IBM Corporation

Licensing: Metrics and Summary

	Personal	Express-C	Express	Workgroup	Enterprise
Pricing metric	Per install (Assumes one user)	Free Download (Unsupported)	Authorized Users (minimum of 5 per server) or Per Server or PVUs (limited to 200 PVUs) Eligible for Sub-capacity pricing	Authorized Users (minimum of 5 per socket) or Per Socket or PVUs (limited to 480 PVUs) Eligible for Sub-capacity pricing	Authorized Users (minimum of 25 per 100 PVUs) or PVUs Eligible for Sub-capacity pricing
Processor limit	N/A	DB2 throttles itself to use maximum of 2 cores	DB2 throttles itself to use maximum of 4 cores	DB2 throttles itself to use maximum of 16 cores and 4 sockets	No Limit
Memory limit	N/A	DB2 throttles itself to use maximum of 2 GB	DB2 throttles itself to use a maximum of 4GB	DB2 throttles itself to use a maximum of 16GB	No Limit
Platforms supported	Windows & Linux	Windows, Linux, Solaris (x64)	Windows & Linux	Windows, Linux, AIX, Solaris, HP-UX	Windows, Linux, AIX, Solaris, HP-UX

14

© 2010 IBM Corporation

DB2 Process Model

- **Single process and multithreaded model**

- Process: db2sysc
- Threads: Engine Dispatchable Units (EDU)
- Multithreaded architecture benefits:
 - ✓ New thread requires less resources than a new process
 - ✓ Less time for context switching
 - ✓ Easy configuration across platforms
 - ✓ Dynamically allocate memory for sharing among EDUs

 Use `db2pd -edus`
to list all active EDUs

- **DB2 Agents (db2agent)**

- Special type of EDU to handle application requests
- The DB2 engine keeps a pool of agents available to service requests
- An application is mapped to a coordinator agent

- **DB2 has firewall to protect DB and DB manager**

- Application runs on different address space to prevent app errors leading to corruption of dbm files or internal buffer

15

© 2010 IBM Corporation

SQL in a nutshell

- **Data Definition Language (DDL)**

- Defines properties of data objects
CREATE, ALTER, DROP, TRANSFER OWNERSHIP

- **Data Manipulation Language (DML)**

- Used to retrieve, add, edit and delete data
SELECT, INSERT, UPDATE, DELETE

- **Data Control Language (DCL)**

- Controls access to databases and data objects
GRANT, REVOKE

- **Transaction Control Languages (TCL)**

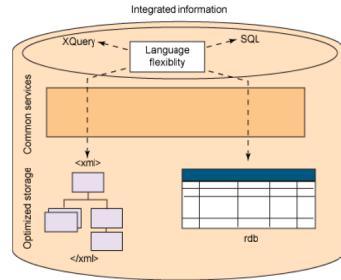
- Groups DML statements into transactions that can collectively be applied to a database or undone in the event of a failure
COMMIT, ROLLBACK, SAVEPOINT

16

© 2010 IBM Corporation

pureXML & XQuery

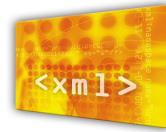
- **DB2 is fully hybrid DBMS with pureXML technology**
 - Native storage of XML data type
- **XQuery can be used for querying and modifying XML data**
 - Search for objects that are at unknown levels of the hierarchy.
 - Perform structural transformations on the data
 - Return results that have mixed types.
 - Update existing XML data



Returns xml data in the column

```
xquery db2-fn:xmlcolumn("XMLCUSTOMER.INFO");

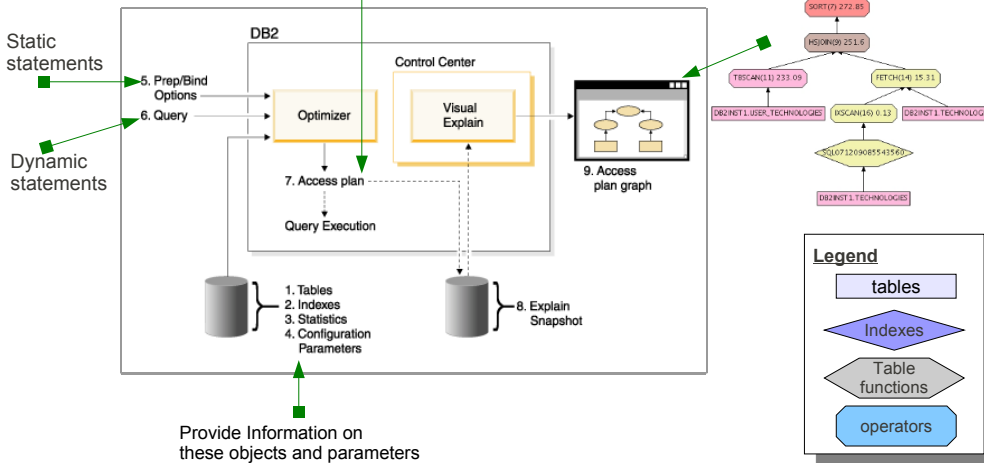
xquery
for $p in db2-fn:xmlcolumn("XMLPRODUCT.DESCRPTION")/product
let $limit := 0.05
where $p/description/price > $limit
order by $p/data(@pid) descending
return ( $p/description/name )
```



Retrieve all XML documents from an XML column, then process them with an XQuery expression

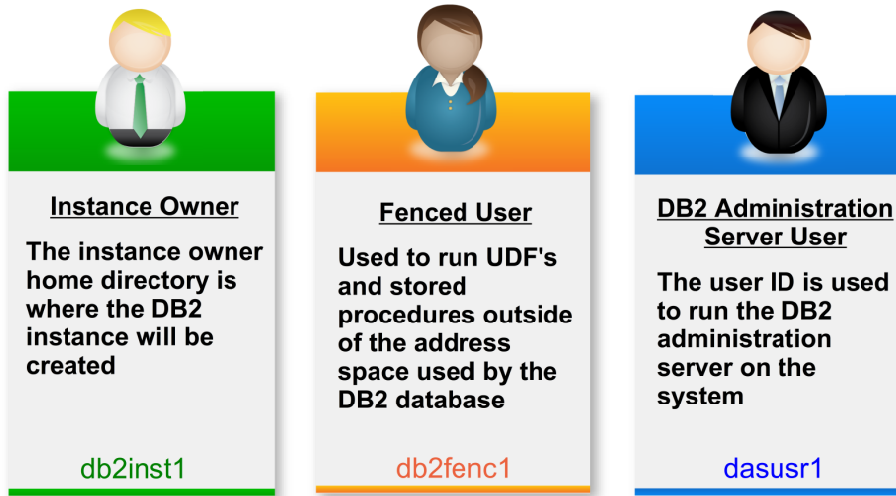
DB2 Access Plan

An **access plan** specifies the order of operations for accessing data necessary to resolve a SQL or XQuery statement

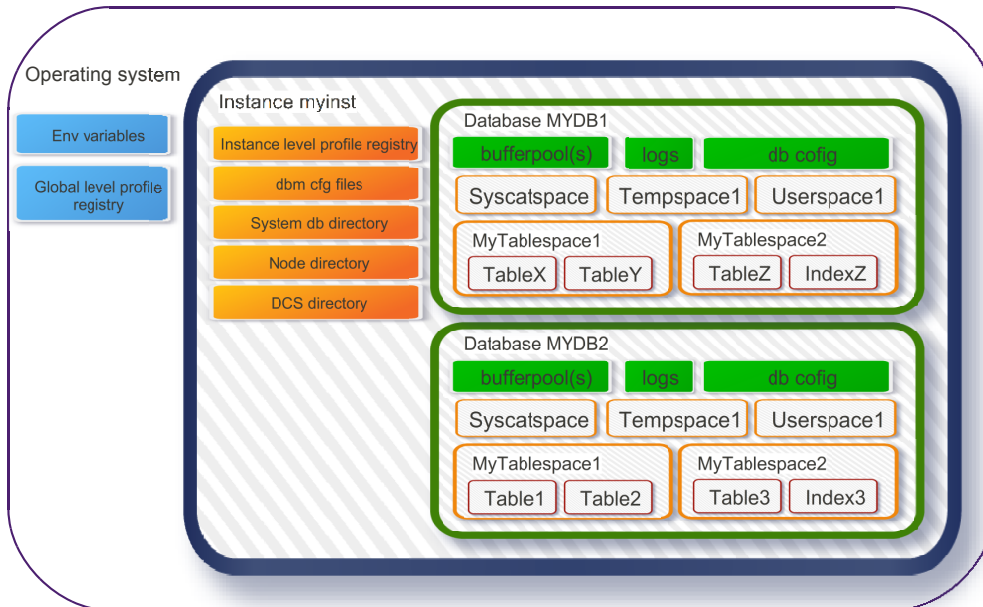


DB2 Users

Three users and groups are required

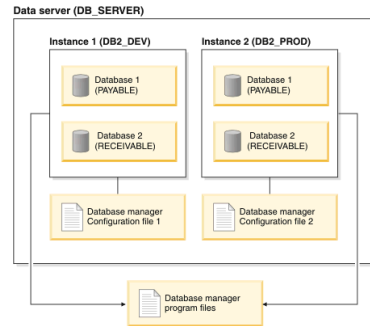


DB2 Environment



Instances

- **Stand-alone DB2 environment**
- **Can have multiple instances per data server**
- **All instances share the same executable binary files**
- **Each instance has its own configuration**
- **DB2 allows installations of different versions (binaries) in the same machine**



Command	Description	Example
db2start	Start the default instance	db2start
db2stop	Stop the current instance	db2stop -f
db2icrt	Create an instance	db2icrt -u db2fenc1 db2inst1
db2idrop	Drop an instance	db2idrop -f db2inst1
db2ilist	List all instances	db2ilist
db2imigr	Migrate an instance after upgrading DB2	db2imigr -u db2fenc1 db2inst1
db2iupdt	Update an instance after installation of a fix pack	db2iupdt -u db2fenc1 db2inst1


21

© 2010 IBM Corporation

Logging:: db2diag.log


- **Trouble shooting and diagnostic purposes**
- **Located in \$DB2INSTANCE_HOME/sqllib/db2dump/ by default**
- **General log which contains all DB2 errors and warnings**

2 forms:



Single diagnostic log file (db2diag.log)

single active log file that grows indefinitely. DEFAULT behavior



Rotating diagnostic log files (db2diag.N.log)

set of files that the active log file closes and opens db2diag.N+1.log when it reaches the limit size

Configuration parameters:

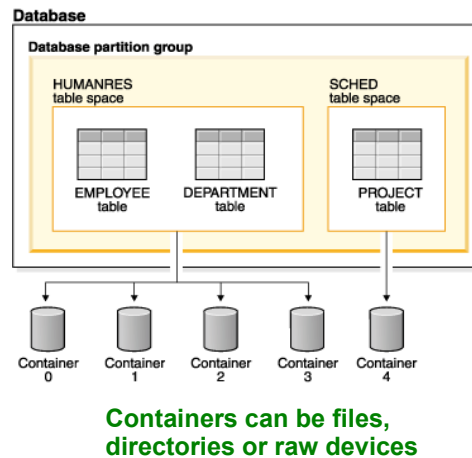
- **Diagsize:** size of the log files for rotating log files form; 0 for single log file form
- **Diagpath:** Location of the log file(s)
- **Diaglevel:** Types of errors to be written to log

22

© 2010 IBM Corporation

DB2 Storage:: Table Spaces Overview

- Logical objects in between logical table and physical containers
- Allows assignment of the location of data to particular logical devices or portions thereof
- All tables, indexes, and other data are stored in a table space
- Can be associated to a specific buffer pool



23

© 2010 IBM Corporation

DB2 Storage:: Table Space Management

- **System Managed Spaces (SMS)**
 - Data stored in files representing data objects
 - Space is allocated on demand
 - Access to data controlled using standard I/O functions of the OS
 - ✓ Ideal for small, personal databases and databases that grow/shrink rapidly
 - ✗ Low maintenance and monitoring

```
CREATE TABLESPACE tbsp1 MANAGED BY SYSTEM
USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
```

- **Database Managed Spaces (DMS)**
 - Data stored in files or on raw devices
 - Storage space pre-allocated in file system, typically contiguous physically
 - ✓ Ideal for performance-sensitive applications
 - ✗ Increased maintenance and monitoring

```
CREATE TABLESPACE tbsp2
PAGESIZE 8K MANAGED BY DATABASE
USING (FILE '/storage/dms1' 10 M) AUTOESIZE YES
```

24

© 2010 IBM Corporation

DB2 Storage:: Table Space Management

▪ Automatic Storage Table Space

- DBM creates and extends containers as needed up the limits imposed by the storage paths associated with the database
- Automatically handles resizing table spaces
- Creates a DMS table space for regular/large table spaces
- Creates a SMS table space for user or system temporary table spaces

New DB & TBSP
are handled by automatic storage
by DEFAULT

```
CREATE DATABASE mydb AUTOMATIC STORAGE YES

CONNECT TO mydb

CREATE TABLESPACE tbsp1 MANAGED BY AUTOMATIC STORAGE
```

25

© 2010 IBM Corporation

DB2 Storage:: Buffer Pools

▪ Area of main memory used to cache table and index data

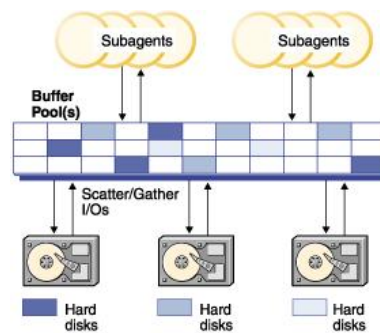
▪ Each database must have at least one buffer pool

- By default IBMDEFAULTBP is used
- Buffer pools can be created, dropped or altered
- SYSCAT.BUFFERPOOLS catalog view accesses the information for the buffer pools defined in the database

▪ Every table space associates a specific buffer pool of the same page size

- Match buffer pool size with purpose of table to increase hit ratio

▪ Self-Tuning Memory Manager (STMM) available



```
CREATE BUFFERPOOL bp4k PAGESIZE 4K
CREATE TABLESPACE tbsp1 PAGESIZE 4K BUFFERPOOL bp4k
```

26

© 2010 IBM Corporation

DB2 Security

Access to DB2



Authentication:

System verifies a user's identity
"You are who you say you are"

Access within DB2 database management system

Authorities:

Various degrees of control over functions. Can be at system level, database level or object level

Privileges:

Permissions to perform an action or a task

Label Based Access Control (LBAC) credentials:

Decide exactly who has write access and who has read access to individual rows and individual columns

27

© 2010 IBM Corporation

DB2 Sample Database

- To create the sample database populated with both relational data and XML data
- Verify the database creation by simply connecting and querying the data

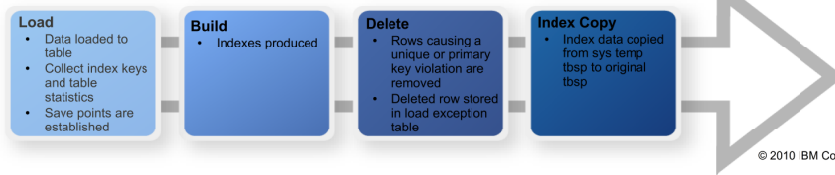
```
db2samp1 -dbpath $HOME -sql -xml  
  
db2 catalog database sample as sample  
      at node mynode1  
  
db2 drop database sample
```

28

© 2010 IBM Corporation

Export, Import and Load Utility

- **Oracle tools**
 - Exporting data: Oracle exp; SQL*Plus
 - Importing data: Oracle imp; SQL*Loader
- **Export Utility**
 - Move data from table or view to files
- **Import Utility**
 - Performs SQL INSERTs
- **Load Utility**
 - Moving large quantities of data into newly created tables, or into tables that already contain data
 - Writes formatted pages directly into the database
 - Does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes)
 - Handles most data types: XML, LOBs, UDTs
 - 4 distinctive phases:

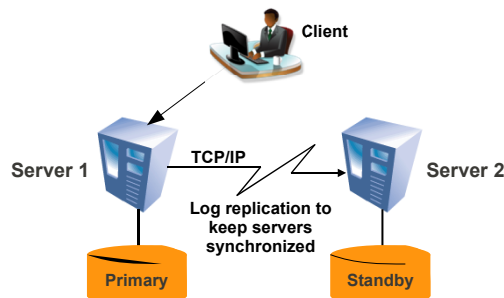


29

© 2010 IBM Corporation

What is HADR?

- **An HADR pair consists of a primary and a standby database**
 - Primary
 - Handles all client connections and processes transactions
 - Continuously ships DB2 HADR log files to the standby over TCP/IP network
 - Standby
 - Originally initialized with cloned database from the primary
 - Keep in sync with primary by applying received transaction logs buffers

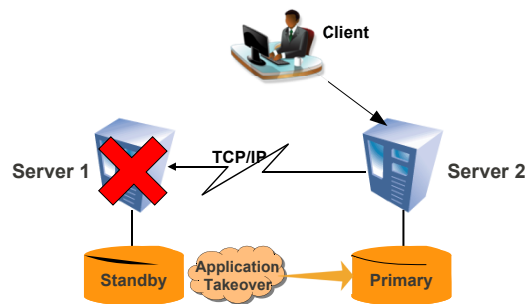


30

© 2010 IBM Corporation

What is HADR?

- **When primary goes offline (planned or unplanned), standby to take over the transactional workload**
 - Standby becomes the new primary
- **Manual or Automatic Takeover via Tivoli System Automation (TSA)**
- **Clients transparently re-routed with Automatic Client Reroute (ACR)**



31

© 2010 IBM Corporation

IBM Data Studio 2.2 Overview



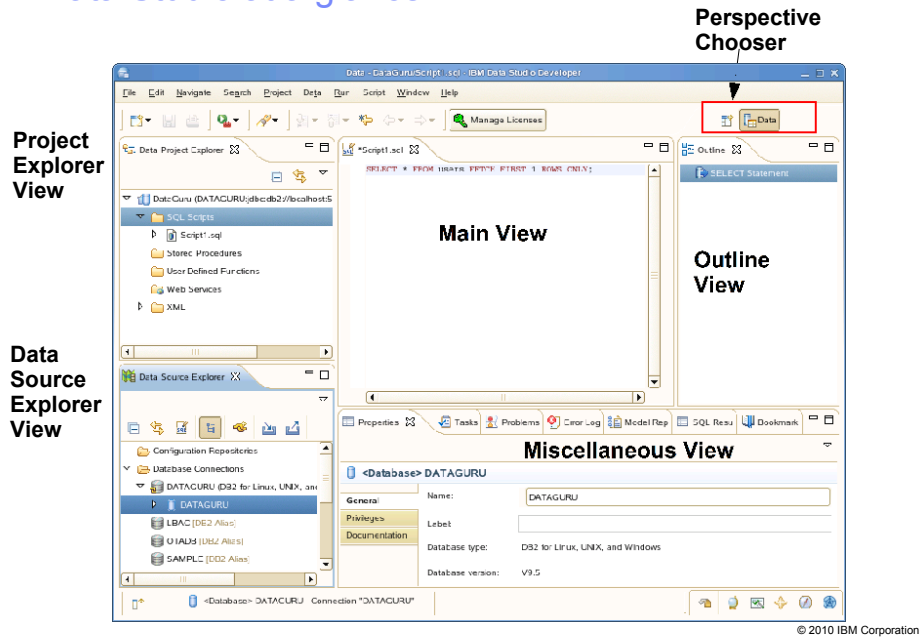
- **No-charge Integrated Development Environment (IDE)**
 - Geared towards application developers and DBAs
 - Supports DB2 for LUW, i5/OS and z/OS, Apache Derby, Informix IDS, and others
- **Benefits**
 - Integrates features previously available in separate tools to minimize context switching
 - Built on the Eclipse platform, offers low learning curve
 - Simplifies development and administration functionality to increase productivity for all roles throughout the data life cycle

Download now at <http://www.ibm.com/software/data/studio>

32

© 2010 IBM Corporation

IBM Data Studio at a glance



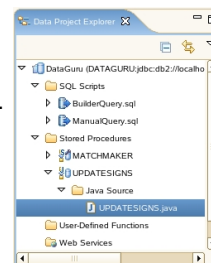
33

© 2010 IBM Corporation

Key Features

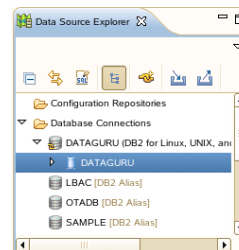
Data application developer features

- **Routine wizards, editors and debugger** to create, test, debug, and deploy routines, eg. stored procedures and UDFs
- **SQL Query builder** and the SQL and XQuery editor to create, edit, and run SQL queries.
- **Visual Explain** to tune routines and SQL queries
- **Create Web services** that expose database operations to client applications
- **XML Wizards and editors** to develop XML applications
- **Develop SQLJ applications** in a Java project



Data and database object management features

- **Establish connection to data sources**
- **Work with data objects:** browse, modify privileges, drop
- **Data object editors and wizards** to create and alter data objects
- **Change impact analysis**
- **Work with data:** basic support for extracting and loading data
- **Use data diagrams** to visualize the relationships between data objects



34

© 2010 IBM Corporation

Integrated Data Management (IDM) Portfolio

- **IBM Optim Integrated Data Management solutions**

- Manage data from requirements to retirement
- Boost performance
- Empower collaboration
- Improve governance across applications, databases and platforms.



- **Integrated Data Management Information Center**

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp>

- **In addition to Data Studio for development, there are paid editions with additional functionality**

- ★ **Optim Development Studio 2.2**

- **Create and test database and pureQuery applications**
 - **Support for Oracle databases**

- ★ **Optim Database Administrator 2.2**

- **Automates and simplifies complex database structural changes**

35

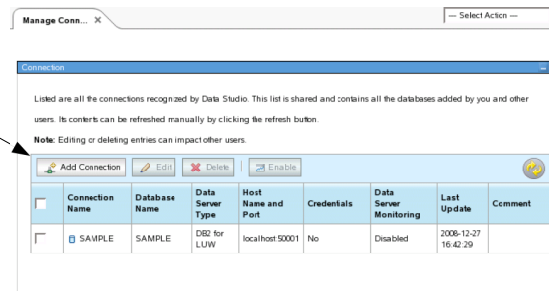
© 2010 IBM Corporation

Data Studio Administration Console (DSAC)

- Web based tool for database health monitoring
- Provides a single portal for viewing the status of all your databases
- Available for Linux, UNIX and Windows
- Available as a free download from:

<http://www.ibm.com/developerworks/spaces/optim?pageid=649>

Add connection by specifying the database name, host, port, user and password



36

© 2010 IBM Corporation

DSAC Capabilities

Health Summary tab quickly summarizes the status of all databases monitored by DSAC

Alert List tab displays the warnings associated for each database

The top screenshot shows the 'Alert List' tab in the DSAC interface. It features a search bar with 'Search', 'Clear Search', 'Filter', and 'Delete' buttons. Below the search bar is a table with columns for 'Severity', 'Alert Type', 'First Time', 'Last Time', and 'Connection Name'. A message box at the bottom of the table states 'No alerts to show'.

The bottom screenshot shows the 'Dashboard' tab for a database named 'SAMPLE'. It displays a 'Group Overview' section with a timeline and several performance metrics: Transaction Rate, Failed Transaction Rate, I/O Volume, Logging Volume, Lock Waits, and Lock Escalation Rate. Each metric shows a 'Current' value and a '90 minutes trend'.

37

© 2010 IBM Corporation

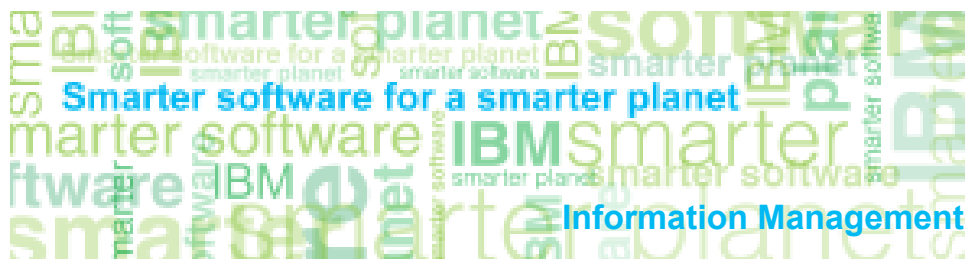
Information Management Ecosystem Partnerships
IBM Canada Lab



Questions?

Summer/Fall 2010

E-mail: imschool@us.ibm.com
Subject: "DB2 Academic Workshop"



© 2010 IBM Corporation



IBM DB2[®] 9.7

**IBM Data Studio
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

1. INTRODUCTION.....	3
2. OBJECTIVES	3
3. SUGGESTED READING	3
4. GETTING STARTED: THE BASICS OF IBM DATA STUDIO	4
4.1 ECLIPSE FUNDAMENTALS.....	4
4.1.1 DATA ORGANIZATION – WORKSPACES AND PROJECTS.....	4
4.1.2 USER INTERFACE – VIEWS AND PERSPECTIVES	6
5. ENVIRONMENT SETUP REQUIREMENTS	7
5.1 INITIAL STEPS	7
6. LAUNCHING DATA STUDIO	9
6.1 DATABASE CONNECTIONS.....	11
6.1.1 CREATING A NEW CONNECTION.....	11
6.1.2 MODIFYING DATABASE PARAMETERS.....	13
6.1.3 STOPPING – STARTING YOUR DB2 INSTANCE	14
6.1.4 DISCONNECTING AND RECONNECTING	15
7. SUMMARY.....	15

1. Introduction

With the advent of IBM® Data Studio comes a major advance in the way DB2® developers and administrators alike carry out their day to day functions. Historically, depending on the tasks to be completed, it was common to switch back and forth between disparate tools such as Control Center, Health Monitor, Developer Workbench, and even the DB2 Command Line Processor (CLP).

The release of IBM Data Studio changes all this, facilitating DB2 administration, design, development and monitoring, all within an integrated Eclipse-based environment. Data Studio, the same tool that allows tuning of buffer pools and restriction of access to data objects, can now be used to develop data web services and debug stored procedures. By leveraging the power of IBM Data Studio, users are certain to enjoy increased productivity as they find themselves able to perform a majority of their tasks within a single environment.

2. Objectives

By the end of this lab, you will be able to:

- ▶ Understand the basics of an Eclipse-based environment
- ▶ Establish a database connection
- ▶ Modify database parameters
- ▶ Start and stop a DB2 instance

3. Suggested reading

IBM Data Studio: Get Started with Data Web Services

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0711pauser-i.html>

An introduction to data web services development, deployment, and testing using IBM Data Studio.

IBM Data Studio Information Center

<http://publib.boulder.ibm.com/infocenter/dstudio/v1r1m0/index.jsp>

A repository complete with tutorials on developing and administering with IBM Data Studio.

4. Getting Started: The Basics of IBM Data Studio

This section of the lab introduces you to the basics of IBM Data Studio and how you can quickly get up and running with it.

After completing this section, you will be able to:

- ▶ Launch Data Studio
- ▶ Create a new database connection
- ▶ Disconnect and reconnect to a database

4.1 Eclipse Fundamentals

IBM Data Studio is built upon the Eclipse platform and, as such, is said to be an Eclipse-based development environment. The Eclipse platform is a framework that allows integrated development environments (IDE) to be created; plug-ins exist to allow development in Java, C/C++, PHP, COBOL, Ruby, and more. Developers using Eclipse will appreciate the familiar look and feel that IBM Data Studio offers.

4.1.1 Data Organization – Workspaces and Projects

In an Eclipse-based environment, all development takes place within a *project*, which is a directory that contains all of the source code, graphics, and other collateral. This is a concept with which most are familiar from using other IDE's. In Data Studio, you will typically work with *Data Development* projects, but other project types exist for Java development, web development, and more.

Each project you create must be contained within a *workspace*, which is a directory in your file system. A workspace directory contains subdirectories for each of the projects created within it. For example, Figure 4-1 demonstrates a scenario in which a workspace has been created on the path `/workspace`, and three projects – BankApp, BookStore, and WebSite – have been created within the workspace. Notice that the projects have all been created as subdirectories of `/workspace`.

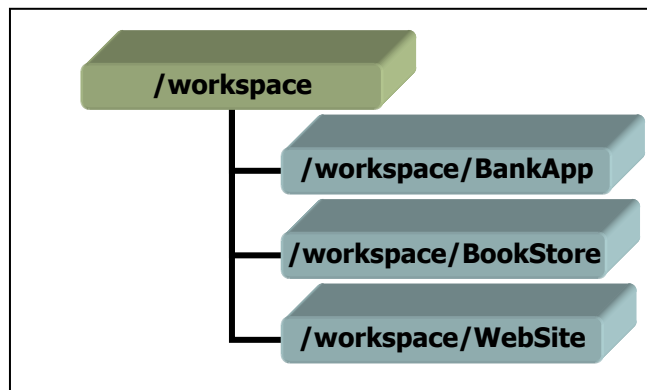


Figure 4-1 – Workspace-project hierarchy

In an Eclipse-based environment, workspaces and projects can easily be navigated via the *Project Explorer* view (we'll cover views in an upcoming section).

When an Eclipse-based environment is opened, the user chooses which workspace to use in the dialog displayed in Figure 4-2. It is possible to create a new workspace by entering a new, non-existent path, or to work with an existing workspace by specifying an existing path.

Additionally, users can choose to only work with one particular workspace (and to never be bothered again!) by checking the **Use this as the default and do not ask again** checkbox. Of course, this can always be undone by modifying a setting in the program preferences.

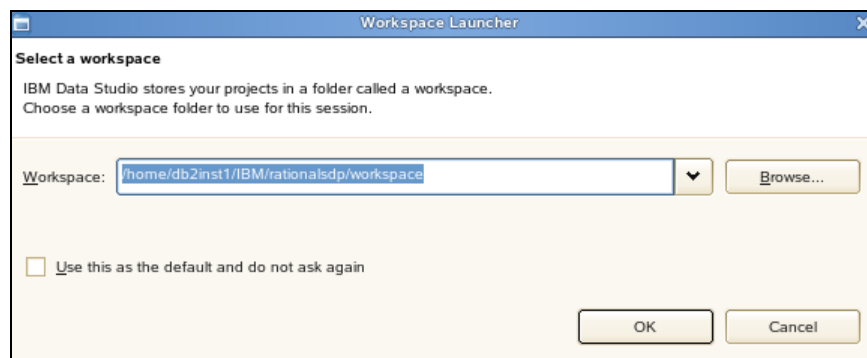


Figure 4-2 – Workspace selection

Figure 4- shows how the workspace hierarchy from Figure 4-1 looks in the Project Explorer.

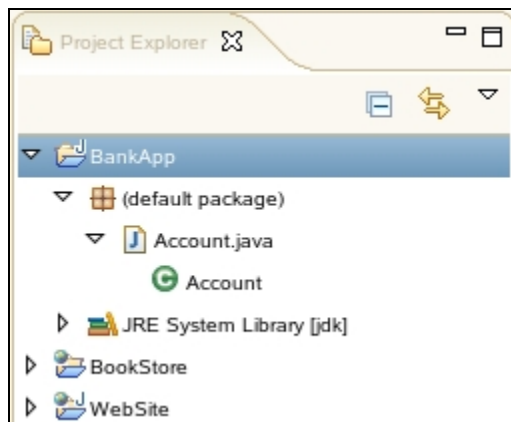


Figure 4-3 – The Project Explorer view

4.1.2 User Interface – Views and Perspectives

Eclipse-based environments offer easy-to-use, customizable graphical interfaces through the use of *views* and *perspectives*. Just as workspaces contain projects, Eclipse perspectives contain views.

In fact, we've already seen an example of a view. In Figure 4-, we saw that the *Project Explorer* view shows all projects in a workspace and files contained within them. A view is nothing more than a task pane – a docked window that allows objects to be viewed and possibly manipulated. There are also many other views, such as the *Data Source Explorer* view, shown in Figure 4-2, which allows users to work with database connections.

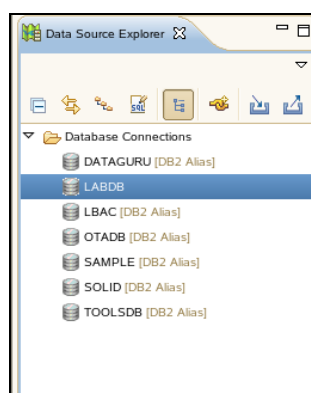


Figure 4-2 – The Data Source Explorer view

Eclipse-based environments define perspectives as a collection of views appropriate for a particular task or line of work. When a perspective is opened, all views associated with it are opened in the environment, and any other views previously opened are hidden.

In IBM Data Studio, you will generally work with the *Data* perspective shown in Figure 4-4. This perspective provides the *Data Project Explorer* view, *Data Source Explorer* view, *Data Output* view, and others.


To switch between perspectives, click the desired name in the toolbar displayed in Figure 4-3. If the perspective you are looking for is not displayed, simply click the  toolbar icon to bring up a list of available perspectives.



Figure 4-3 – Changing perspectives on the toolbar

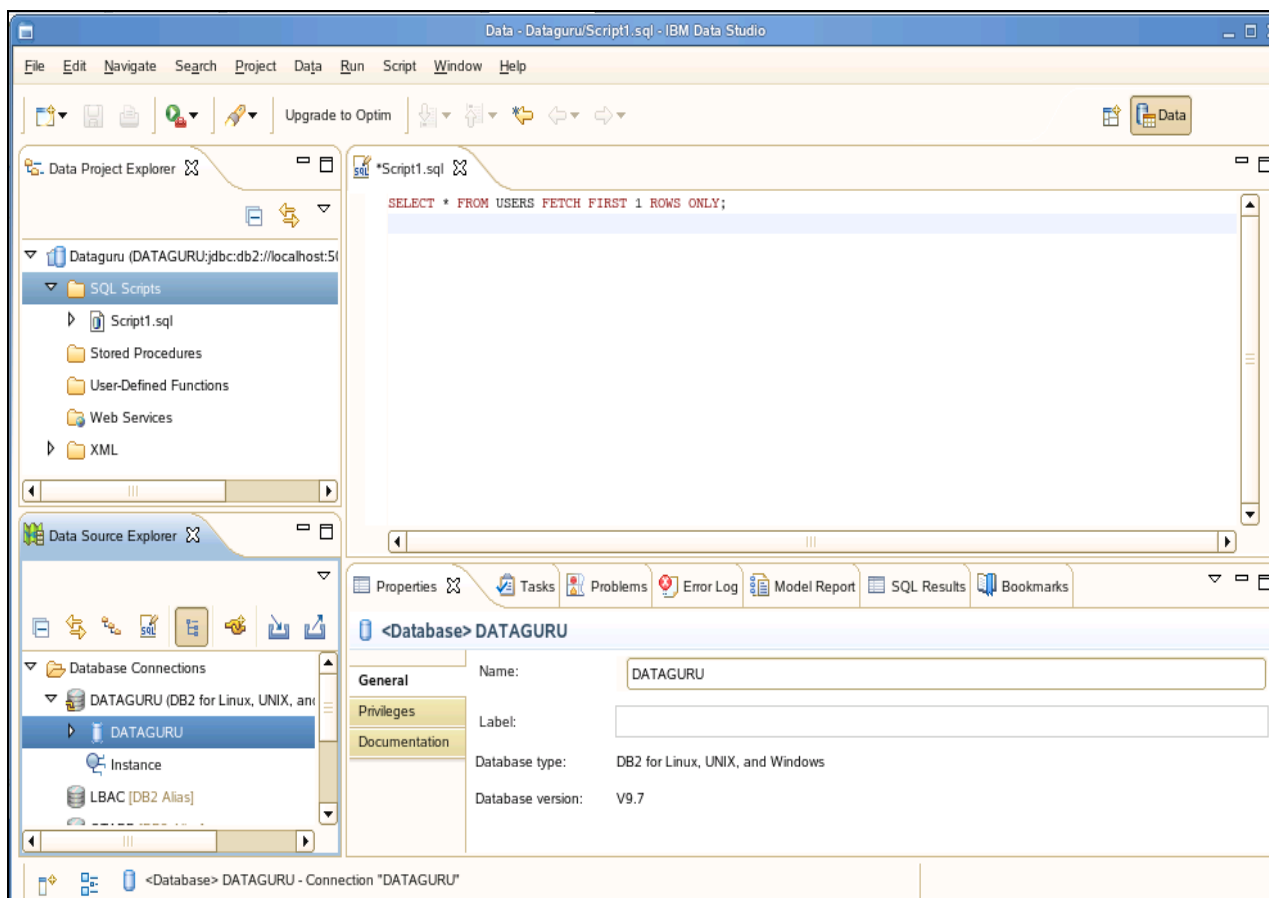


Figure 4-4 – The Data perspective

Eclipse-based environments allow creation of custom perspectives by specifying which views to load.

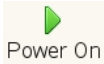
5. Environment Setup Requirements

To complete this lab you will need the following:

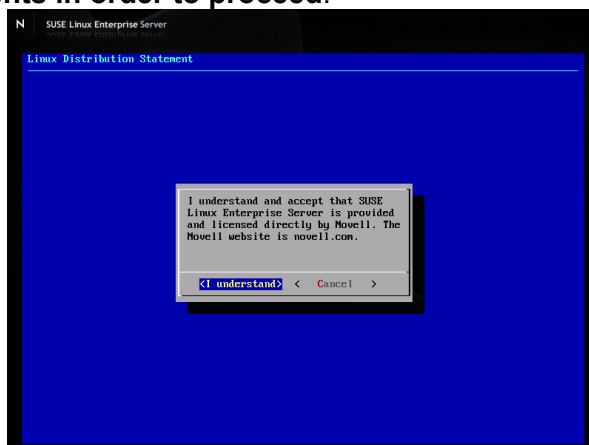
- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 6.x or later

For help on how to obtain these components please follow the instructions specified in **VMware Basics and Introduction**.

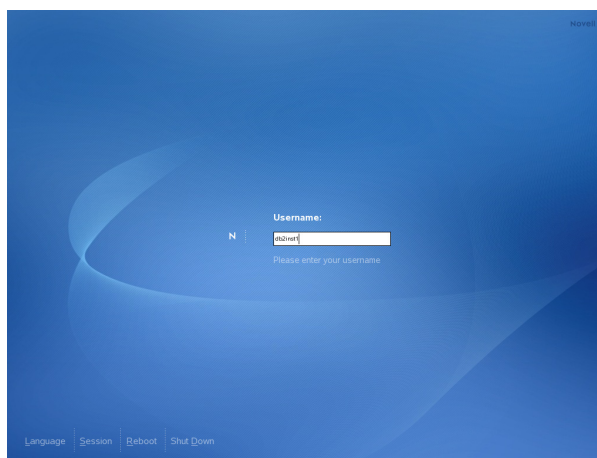
5.1 Initial Steps

1. Start the VMware image by clicking the  button in VMware.

- At the login prompt, login with the following credentials:
 - ▶ Username: `root`
 - ▶ Password: `password`
- Read and accept the license agreement. **You must accept and understand the license agreements in order to proceed.**

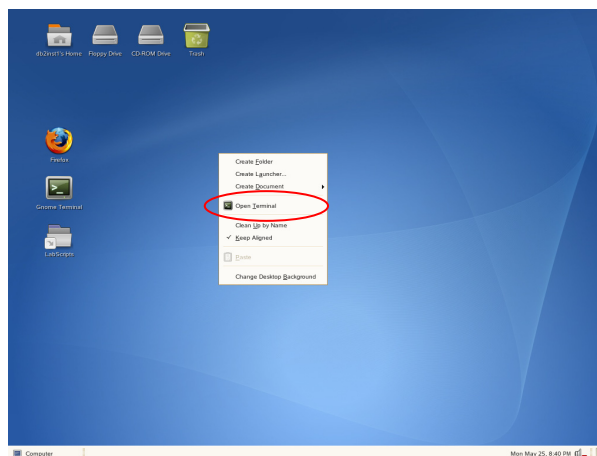


- At the new login prompt, login with the db2inst1 credentials:
 - ▶ Username: `db2inst1`
 - ▶ Password: `password`



Note: It is very important **not to login as root** user at this point.

5. Open a terminal window by right-clicking on the **Desktop** and choosing the **Open Terminal** item.



6. Ensure that the DB2 Database Manager has been started by issuing the following command at the prompt:

```
db2inst1@db2rules:~> db2start
```

Note: This command will only work if you logged in as the user `db2inst1`. If you accidentally logged in as another user, type `su - db2inst1` at the command prompt password: `password`.

7. This lab assumes you have the `SAMPLE` database created. You can check the list of existing databases using the command below:

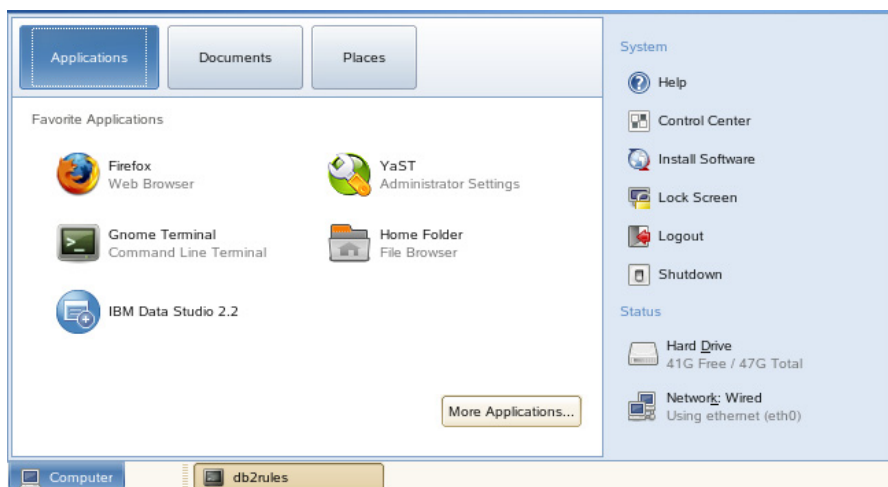
```
db2inst1@db2rules:~> db2 list db directory
```

8. If the `SAMPLE` database is not on the list, you can create it using the following command:

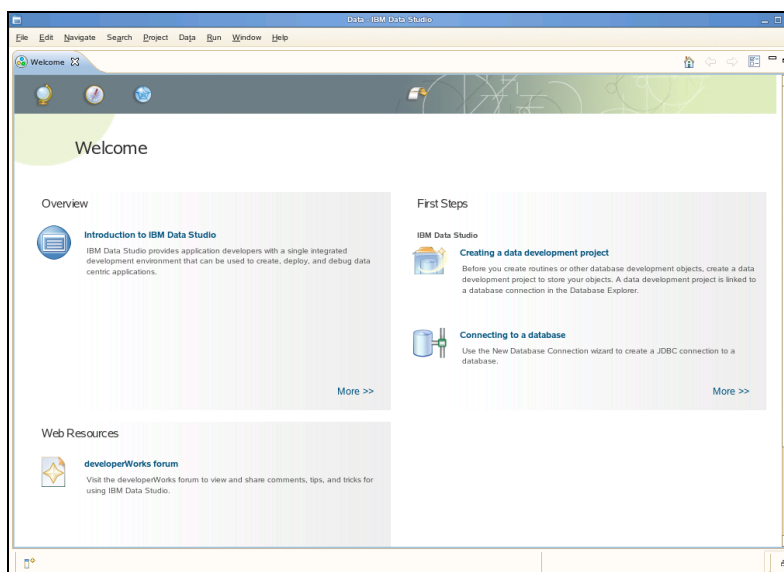
```
db2inst1@db2rules:~> db2sampl
```


6. Launching Data Studio

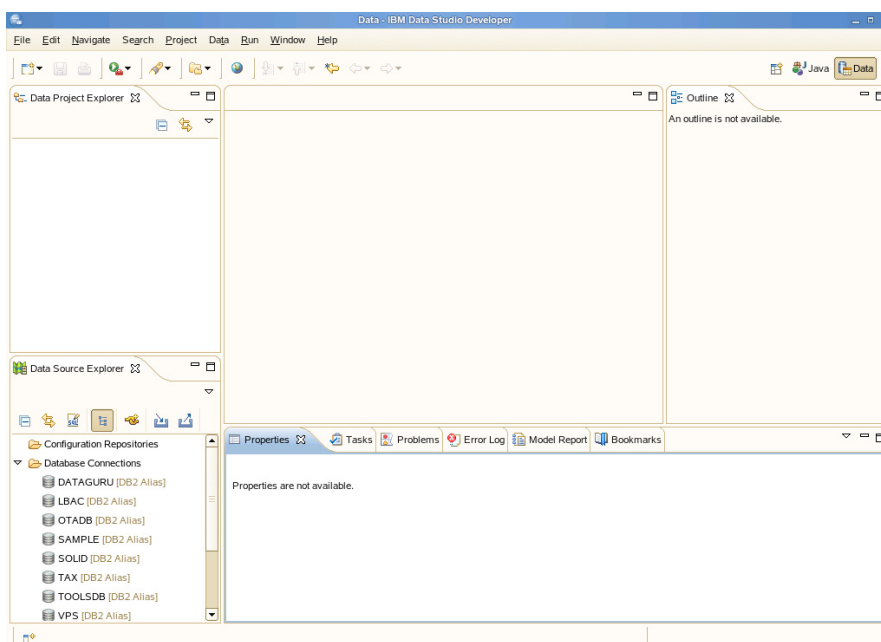
1. Click on the **Computer** button in the bottom left corner of the screen, and select **Data Studio 2.2**.



2. In the **Select a workspace** dialog, accept the default path. Click **OK**.
3. Data Studio will now start with the Welcome homepage.



4. Minimize this window by clicking the minimize button () located at the top right to bring you into the **Data** perspective as shown below.



6.1 Database Connections

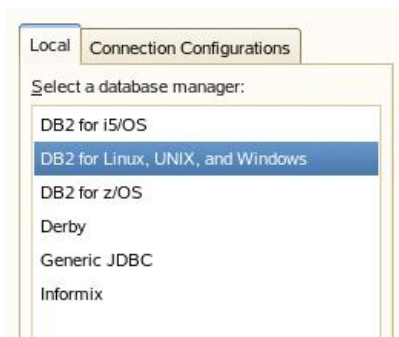
Before you can do anything productive with Data Studio, a connection must be established to a database. The Data Source Explorer view in Data Studio allows you to do this. From this view it is possible to interact with and manipulate database artifacts. Since we will be working with the SAMPLE database, let's create a connection to it.

6.1.1 Creating a New Connection

1. In Data Studio navigate to the **Data Source Explorer** view, right-click on the **Database Connections** folder and select **New...**

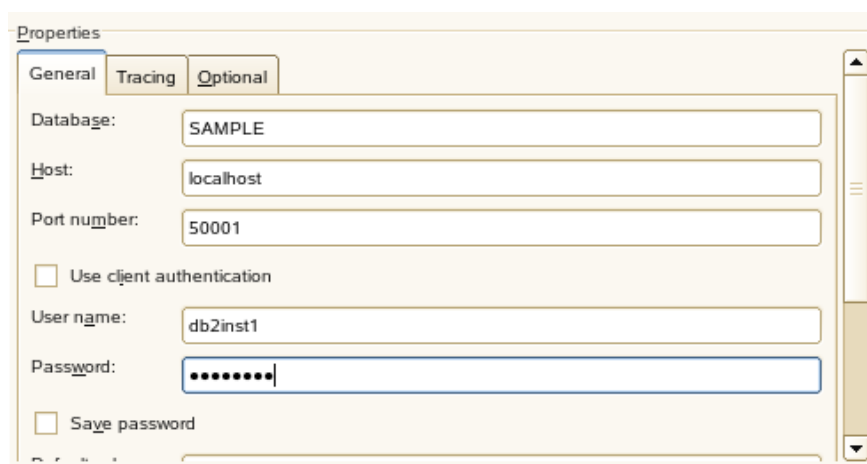
► **Note:** You can also click the  icon in the Data Source Explorer toolbar.

2. Since we're using DB2 on Linux®, select **DB2 for Linux, Unix®, and Windows®**.



3. In the **Properties** pane, you specify the name of the database to which you wish to connect, the host, port number, name of the database instance and the password. Enter the following information:

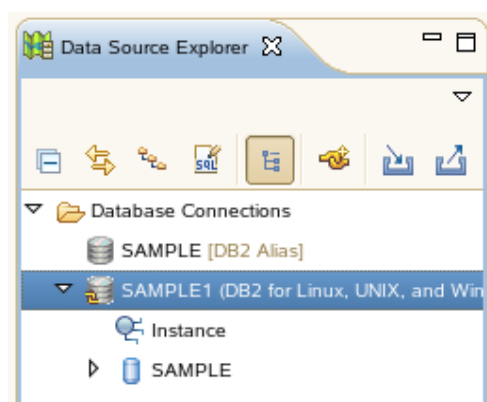
- ▶ **Database:** SAMPLE
- ▶ **Host:** localhost
- ▶ **Port number:** 50001
- ▶ **User name:** db2inst1
- ▶ **Password:** password



The screenshot shows a 'Properties' dialog box with three tabs: 'General', 'Tracing', and 'Optional'. The 'General' tab is active. It contains the following fields and options:


- Database:** A text box containing 'SAMPLE'.
- Host:** A text box containing 'localhost'.
- Port number:** A text box containing '50001'.
- Use cJent authentication
- User name:** A text box containing 'db2inst1'.
- Password:** A text box containing a series of dots to mask the password.
- Save password

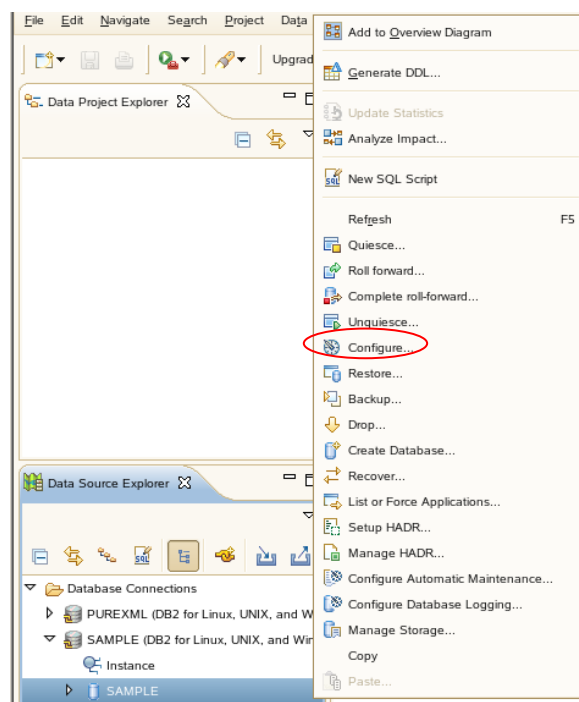
4. Click the **Test Connection** button located on the left. You should receive a message indicating that the connection succeeded. If not, repeat steps 2 - 3, ensuring that your spelling is correct, and try again. Click **Next** when the test is successful.
5. The next page allows you to filter out the data objects that you see by the schema in which they exist. We'll just leave it as is for now, and see another way to filter by schema later on. Click **Finish** to create the connection.
6. In the Data Source Explorer view, expand the **Database Connection** folder if necessary by clicking the ▶ icon. Notice the **SAMPLE1** entry. Also notice that the connection icon beside **SAMPLE1** has a chain, while the others don't. This means that **SAMPLE1** is the only database connection currently open.



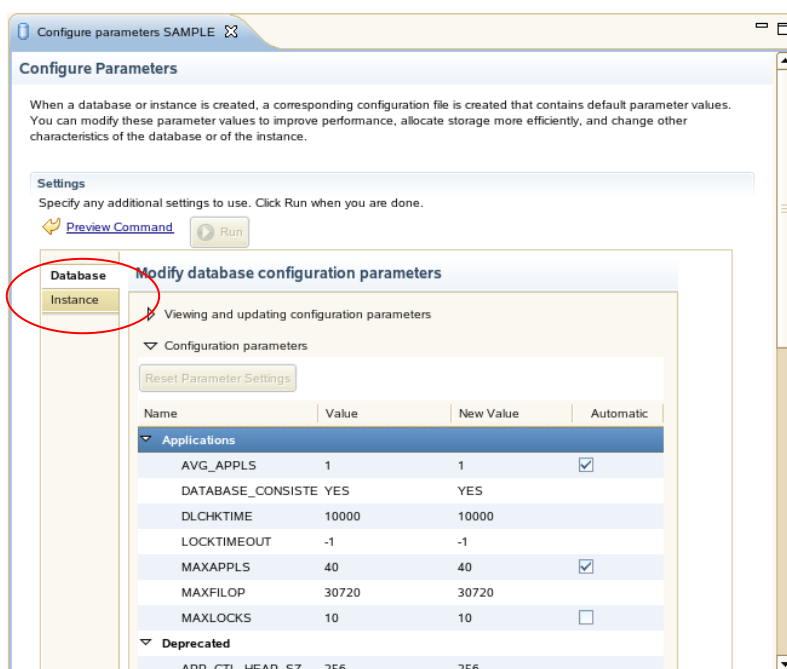
6.1.2 Modifying Database Parameters

Data Studio can perform several administrative functions within DB2. One of these functions is the ability to manipulate database parameters.

1. Right-click on the **SAMPLE** database  **SAMPLE** within the **SAMPLE1** connection, and select **Configure**. Notice that this action will open a new view to configure parameters for the SAMPLE database.



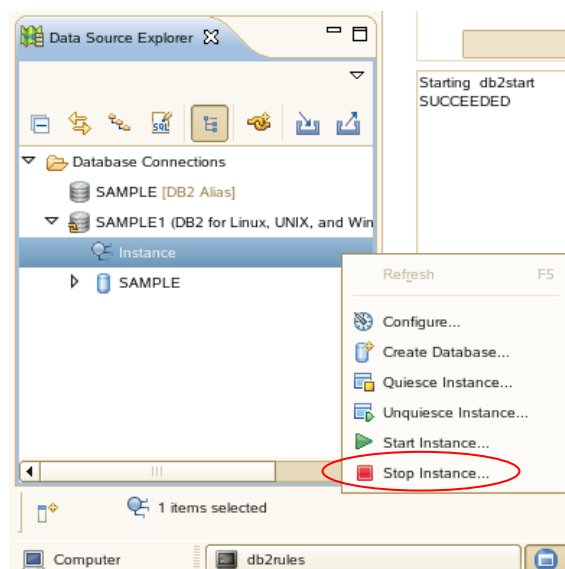
2. From this view it is possible to modify several parameters related to the database configuration as well as parameters related to the instance to which this database belongs. We will not modify any parameters at this time, so simply close the view after you are done exploring.



6.1.3 Stopping and Starting your DB2 instance

In the previous section you notice that it is possible to modify instance level parameters. Some modifications actually require an instance re-start to come into effect. That is why from Data Studio you have the ability to stop and start the instance.

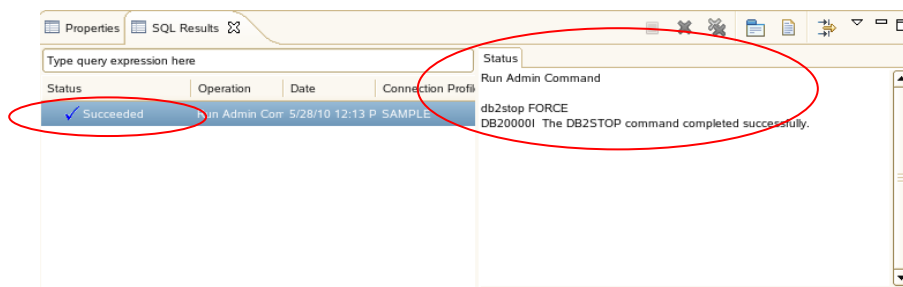
1. To stop the instance, right-click on the instance icon for the **SAMPLE1** database and select **Stop Instance**.



- Notice that the Stop Instance db2inst1 view will open. Click the **Run** button



within the view to stop your DB2 instance. In the SQL Results view you will be able to notice the status of your command that will go from “Running” while in process to “Succeeded” once it is completed. In the Status panel you can also appreciate the SQL command executed as well as the output from the console.



- Now start the instance once again. Right-click on the instance icon for the **SAMPLE** database and select **Start Instance**. Click the **Run** button



within the view to start your DB2 instance.

6.1.4 Disconnecting and Reconnecting

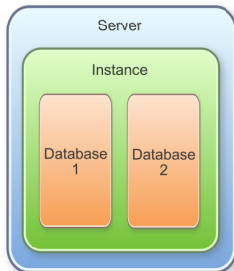
- Right-click on the **SAMPLE1** node, and select **Connect**. Notice that the connection icon has a chain, signifying that the connection has been established.
- Right-click on the **SAMPLE1** node, and select **Disconnect**. Notice that the connection icon no longer has a chain, signifying that the connection has been terminated.
- Right-click on the **SAMPLE1** node again, and select **Delete**. If prompted, confirm that you would like to delete the profile.
- Exit Data Studio.

7. Summary

You can hopefully see by now that IBM Data Studio is a highly productive environment for DB2 development and administration. Over the course of the following labs, we'll see how fast and easy it is to create and execute SQL and XQuery scripts; develop and test stored procedures in SQL and Java; create and alter database objects; analyze query execution; etc.

Servers, Instances, and Databases

DB2 views the world as a hierarchy of objects



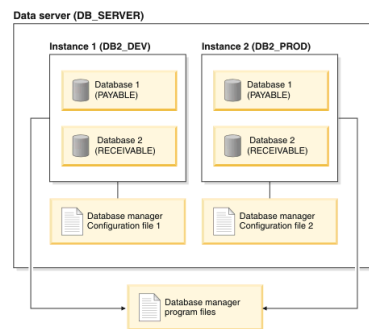
- ..upon installation, the **DB2 Database Manager** (program files) are copied to the **server**, and an instance of the DB2 Database Manager is created.
- ..**instances** are responsible for managing system resources and databases that fall under their control.
- ..**databases** are responsible for managing the storage, modification, and retrieval of data.

3

© 2010 IBM Corporation

Instances

- **Stand-alone DB2 environment**
- **Can have multiple instances per data server**
- **All instances share the same executable binary files**
- **Each instance has its own configuration**
- **Different software level for an instance**



Command	Description	Example
db2start	Start the default instance	db2start
db2stop	Stop the current instance	db2stop -f
db2icrt	Create an instance	db2icrt -u db2fenc1 db2inst1
db2idrop	Drop an instance	db2idrop -f db2inst1
db2ilist	List all instances	db2ilist
db2imigr	Migrate an instance after upgrading DB2	db2imigr -u db2fenc1 db2inst1
db2iupdt	Update an instance after installation of a fix pack	db2iupdt -u db2fenc1 db2inst1

4

© 2010 IBM Corporation

DB and DBM configurations

Description	Example
View Database Manager Settings	db2 get dbm cfg show detail
Change a Database Manager Setting	db2 update dbm cfg using <parameter> <value>

Description	Example
View Database Settings	db2 get db cfg for <database> db2 connect to <database> db2 get db cfg show detail
Change a DB Setting	db2 update db cfg using logprimary 10

Change value of **logretain** & **userexit** db config parameters

Retrieving original value (on disk) and updated value (in memory)

Querying the resulting global Temporary table (DB_CONFIG)

```

UPDATE DB CFG USING LOGRETAIN RECOVERY USEREXIT ON
CALL SYSPROC.GET_DB_CONFIG()
SELECT DBCONFIG_TYPE, LOGRETAIN, USEREXIT FROM SESSION.DB_CONFIG
    
```

Result:

DBCONFIG_TYPE	LOGRETAIN	USEREXIT
0	1	1
1	0	0

5

© 2010 IBM Corporation

Databases

▪ What makes up a DB2 database?

- A DB2 **database** is made up of a collection of objects
- A database contains the following objects:
 - **Tables, views, indexes, schemas**
 - **Locks, triggers, stored procedures, packages**
 - **Buffer pools, log files, table spaces**

▪ Which tools can help you create DB2 databases?

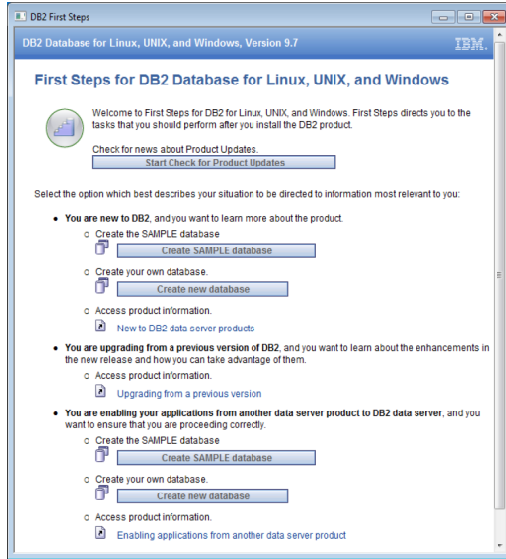
- First Steps
- Control Center (GUI wizard)
- Command Line Processor (CLP)

6

© 2010 IBM Corporation

Creating a DB2 Database – First Steps

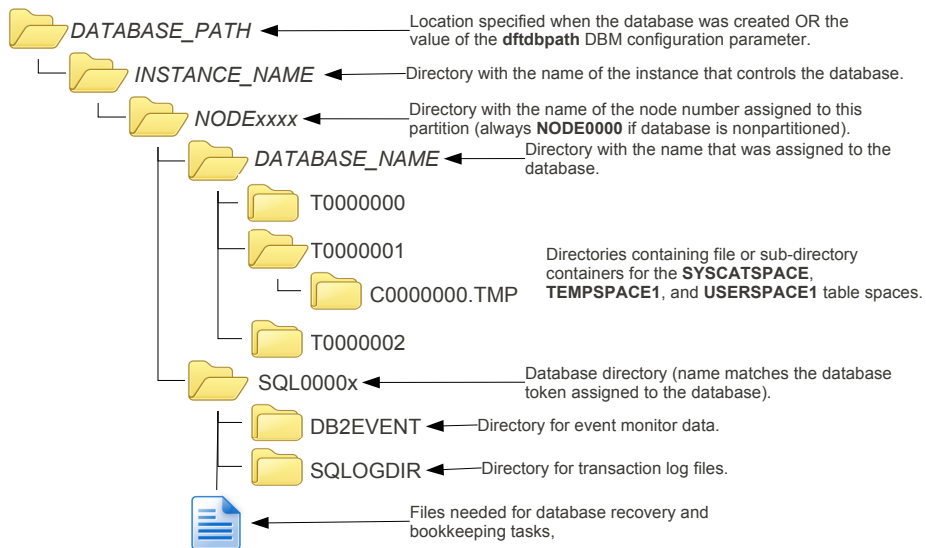
- As part of the DB2 installation process, the **First Steps** panel is displayed allowing the user to generate a number of a sample databases to work with
- To launch the first steps interface issue **db2fs** from a DB2 command line



7

© 2010 IBM Corporation

Typical Directory Hierarchy Tree



8

© 2010 IBM Corporation

Creating a DB2 Database – Command Line

- The **CREATE DATABASE** command initializes a new database

>>-CREATE---+--DATABASE-+---database-name----->... .

Examples:

CREATE DATABASE TESTDB1

database-name

CREATE DATABASE TESTDB2 ON C:

Database is created on drive C:

CREATE DATABASE TESTDB3
 AUTOMATIC STORAGE YES
 ON C:,D: DBPATH ON E:

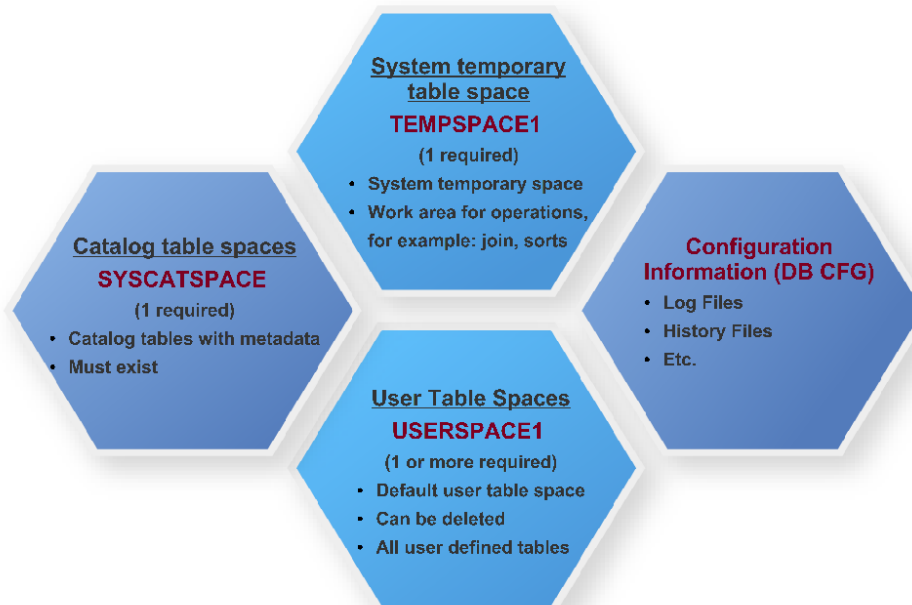
Database is created on drive E:

Automatic store is enabled and storage paths are C: and D:

9

© 2010 IBM Corporation

With databases, DB2 automatically creates...



10

© 2010 IBM Corporation

DB2 Data Server Clients

IBM Data Server Runtime Client

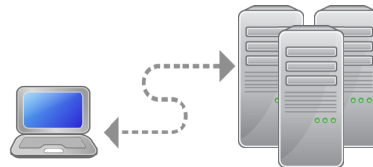
- JDBC, ADO.NET, OLE DB, ODBC, CLI, PHP, and Ruby
- Has CLP but GUI tools are **not included**
- Support LDAP exploitation, TCP/IP and Named Pipe, cataloging

IBM Data Server Client

- All the functionality of IBM Data Server Runtime Client
- Plus functionality for database administration, application development, and client/server configuration.
- Capabilities **include GUI tools** such as configuration assistant, control center, visual studio tools

IBM Data Server Drivers

- Light weight deployment solution for ISVs
- Must be installed manually
 - **IBM Data Server Driver for JDBC and SQLJ**
 - Java stored procedures and user-defined functions
 - JDBC, SQLJ
 - **IBM Data Server Driver for ODBC and CLI**
 - ODBC API, or CLI API
 - **IBM Data Server Driver Package**
 - ODBC, CLI, .NET, OLE DB, PHP, Ruby, JDBC, or SQLJ



11

© 2010 IBM Corporation

Cataloging

▪ DB2 has multiple directories that are used to access databases

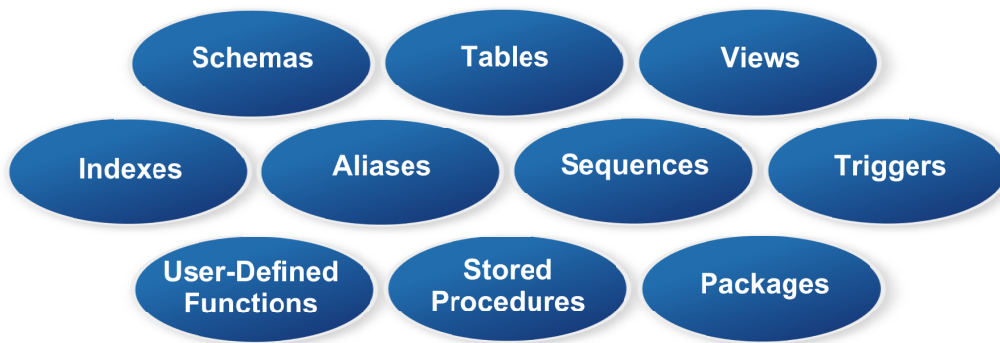
- The **system database directory** contains a list and pointer to where all the known databases can be found.
- The **node directory** contains information relating to how and where remote systems or instances can be found.
- The **Database Connection Services (DCS) Directory** contains information relating to how and where databases on DRDA systems can be found.

12

© 2010 IBM Corporation

Database (Data) Objects

- **Database objects**, also known as **data objects**, are used to control how all user data (and some system data) is stored and organized within a DB2 database



15

© 2010 IBM Corporation

Schemas

- **Schemas** (unique identifiers) are objects that are used to logically classify and group other objects in the database.
- Schemas have **privileges** associated with them that allow the schema owner to control which users can create, alter, and drop objects within them.
- **Benefits of a schema:**
 - Tedious to search through entire database for objects with the same name
 - The name of each object needs to be unique only within its schema
 - Access control

16

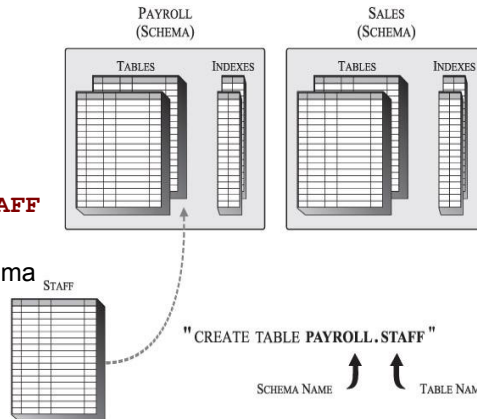
© 2010 IBM Corporation

Schemas

- **Most database objects have a two-part object name**
 - The *first part* being the **schema name** (or qualifier)
 - The *second* is the **object name**
- **When an object is created, you can assign it to a specific schema**
 - `SCHEMA_NAME.OBJECT_NAME`

- **For example**

- `CREATE SCHEMA PAYROLL`
- `CREATE TABLE PAYROLL.STAFF`
- Table named **STAFF** is assigned to the **PAYROLL** schema



17

© 2010 IBM Corporation

Schemas – Example :: Using the command line

- `CREATE SCHEMA payroll AUTHORIZATION user1;`

Schema name

Creates a schema for an individual user with the authorization ID "USER1"

- `COMMENT ON SCHEMA payroll IS 'schema for payroll application';`

Schema to comment on

Comment string

18

© 2010 IBM Corporation

Tables

- A relational database presents data as a collection of **tables**
- A table consists of data logically arranged in columns and rows (records)
 - each column contains values of the same data type
 - each row contains a set of values for each column available

- The storage representation of a row is called a **record**
- the storage representation of a column is called a **field**
- each intersection of a row and column is called a **value**

DEPARTMENT Table

DEPTID	DEPTNAME	COSTCENTER
A000	ADMINISTRATION	10250
B001	PLANNING	10820
C001	ACCOUNTING	20450
D001	HUMAN RESOURCES	30200
E001	R & D	50120
E002	MANUFACTURING	50220
E003	OPERATIONS	50230

Value (points to a cell intersection)

Record (Row) (points to a row)

Field (Column) (points to a column)

19

© 2010 IBM Corporation

Tables – 3 Main Types of Tables

- **Base Tables**
 - User-defined tables designed to hold persistent user data

```
CREATE TABLE department
(
  deptid CHAR(4),
  deptname VARCHAR(30),
  costcenter INTEGER);
```

column-name: names a column of the table

table-name

data-type

- **Result Tables**
 - DB2 Database Manager-defined tables populated with rows retrieved from one or more base tables in response to a query
- **Declared Temporary Tables**
 - User-defined tables used to hold nonpersistent data temporarily, on behalf of a single application.
 - Explicitly created by an application when they are needed and implicitly destroyed when the application that created them terminates its last database connection.
 - Created with **DECLARE GLOBAL TEMPORARY TABLE** statement

20

© 2010 IBM Corporation

Views

- **Views** can be seen as virtual tables derived from one or more tables or views

- Created to limit access to sensitive data or group together data from different tables in a single object.
- Views do not contain real data.
 - Only the view definition itself is actually stored in the database
- Can be deletable, updatable, insertable, and read-only.
- When changes are made to data through a view, the data is changed in underlying table itself.
- Can be used interchangeably with tables when retrieving data.

21

© 2010 IBM Corporation

Views – Example :: Simple view referencing two base tables

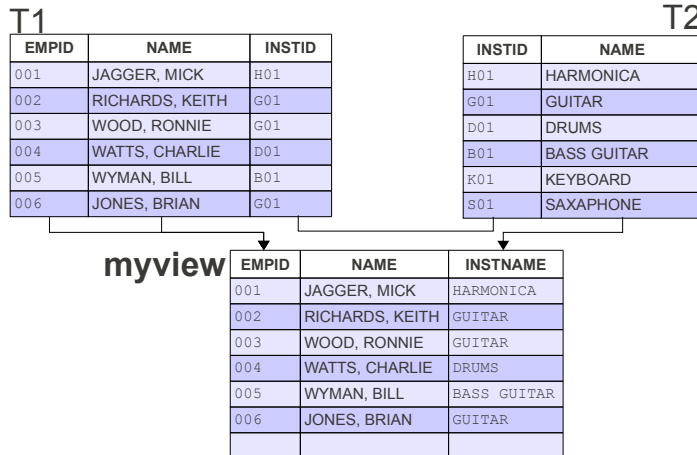
Defines the view

```
CREATE VIEW myview (empid, name, instname) AS
SELECT T1.empid, T1.name, T2.name
FROM T1, T2
WHERE T1.institd=T2.institd
```

view-name

column-name(s)

Identifies the view definition

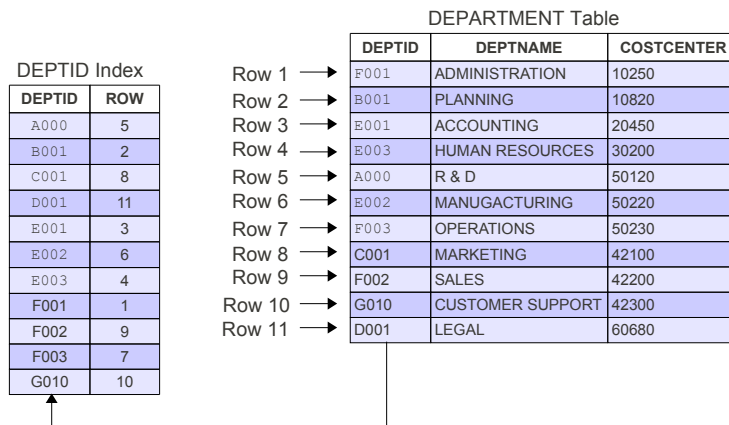


22

© 2010 IBM Corporation

Indexes

- An **index** is an object that contains an ordered set of pointers that refer to rows in a base table. They are based upon one or more columns but **stored as a separate entity**.



23

© 2010 IBM Corporation

Indexes – Importance

- ..provide a fast, efficient method for locating specific rows of data in very large tables.
- ..provide a logical ordering of the rows of a table
- ..can be used to enforce the uniqueness of records stored in a table.
- ..can force a table to use clustering storage, which causes the rows of a table to be physically arranged according to the ordering of their index column values.

24

© 2010 IBM Corporation

Indexes – Example

- Suppose you had the following **EMPLOYEES** base table..

Column Name ...	Data Type ...
EMPNO	INTEGER
FNAME	CHAR(20)
LNAME	CHAR(30)
TITLE	CHAR(10)
DEPARTMENT	CHAR(20)
SALARY	DECIMAL(6,2)

..and you wanted to create an index such that the index key consists of the column named **EMPNO** and all employee numbers entered will be guaranteed to be unique..

25

© 2010 IBM Corporation

Indexes – Example (continued)

UNIQUE prevents the table from containing two or more rows with the same value of the index key

```
CREATE UNIQUE INDEX empno_indx
ON employees (empno)
```

table-name identifies a table on which an index is to be created

column-name identifies a column that is to be part of the index key

26

© 2010 IBM Corporation

Aliases

- An **alias** is an alternate name for an object such as a table or view.
- Like other objects, an alias can be created, dropped, and have comments associated with it.
- Aliases are publicly referenced names, so **no special authority** or privilege is required to use them.
 - However, access to the table or view that an alias refers to still requires appropriate **authorization**.
- Aliases can be created by executing the **CREATE ALIAS** SQL statement

27

© 2010 IBM Corporation

Aliases

▪ Example

- ..you wanted to create an alias that references a table named **EMPLOYEES** and you wanted to assign it the name **EMPINFO**..

```
CREATE ALIAS empinfo FOR employees
```

Names the alias. The name must not identify a table, view, nickname, or alias that exists in the current database.

Identifies the table, view, nickname, or alias for which alias-name is defined.

▪ Why use an alias instead of the actual object name?

- So that SQL statements can be constructed such that they are **independent** of the qualified names that identify the base tables or views they reference.

28

© 2010 IBM Corporation

Sequences

- A **sequence** is an object that is used to generate data values automatically. Unlike an **identity column**, a sequence is not tied to any specific column or any specific table.
 - An **identity column** provides a way for DB2 to automatically generate a unique numeric value for each row that is added to the table.
- **Sequences have the following characteristics:**
 - Values generated can be any exact numeric data type that has a scale of zero (SMALLINT, BIGINT, INTEGER, or DECIMAL).
 - Consecutive values can differ by any specified increment value.
 - Counter values are recoverable. Counter values are reconstructed from logs when recovery is required.
 - Values generated can be cached to improve performance.

29

© 2010 IBM Corporation

Sequences

- **Sequences can generate values in one of three ways:**
 - By incrementing or decrementing by a specified amount, **without bounds**
 - By incrementing or decrementing by a specified amount **to a user-defined limit and stopping**
 - By incrementing or decrementing by a specified amount **to a user-defined limit, and then cycling back to the beginning and starting again**
- **Example: if you wanted to create a sequence that generates numbers, starting with the number 100 and incrementing each subsequent number produced by 10**
- **CREATE SEQUENCE emp_id START WITH 100 INCREMENT BY 10**

Names the sequence

Specifies the first value for the sequence

Specifies the interval between consecutive values of the sequence

30

© 2010 IBM Corporation

Triggers

- A **trigger** defines a set of actions that are performed in response to an **insert**, **update**, or **delete** operation on a specified table.
- Like **constraints**, triggers are often used to enforce data integrity and business rules.
- Unlike **constraints**, triggers can also be used to update other tables, automatically generate or transform values for inserted or updated rows, and invoke functions to perform tasks such as issuing errors or alerts.
- Using triggers places the logic that **enforces business rules** inside the database.

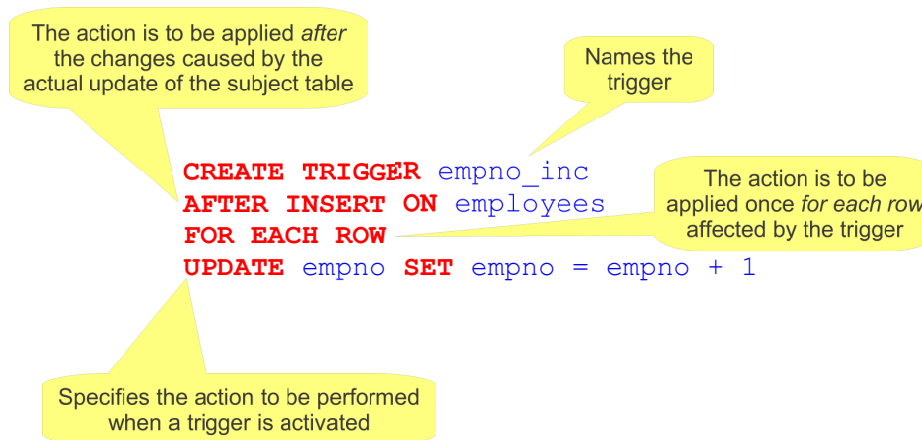
Triggers – Example

- Suppose you had the following **EMPLOYEES** base table..

Column Name ...	Data Type ...
EMPNO	INTEGER
FNAME	CHAR(20)
LNAME	CHAR(30)
TITLE	CHAR(10)
DEPARTMENT	CHAR(20)
SALARY	DECIMAL(6,2)

..and you wanted to create a trigger for **EMPLOYEES** that will cause the value for the column named **EMPNO** to be incremented each time a row is added to the table

Triggers – Example cont'd



User-defined Functions

- **User-defined functions (UDFs)** are special objects that are used to extend and enhance the support provided by the built-in functions available with DB2.
- Unlike DB2's built-in functions, user-defined functions can take advantage of [system calls](#) and [DB2's administrative APIs](#)
- User-defined functions are created (or registered) by executing the **CREATE FUNCTION** SQL statement.
- **SQL Scalar, Table, or Row**. Constructed using only SQL statements and can return a value, row or table.
- **External Scalar/Table**. Written using a high-level programming language such as C, C++, or Java and returns a single value or table.

Stored Procedures

- An **SQL stored procedure** is an ordinary program composed entirely of SQL statements that can be called by an application.
- Stored procedures can be called **locally or remotely**
- An **external stored procedure** is a stored procedure that is written using a high-level programming language
 - External stored procedures can be **more powerful** than SQL stored procedures because they can take advantage of system calls and administrative APIs along with SQL statements.

Stored Procedures – Advantages / Benefits

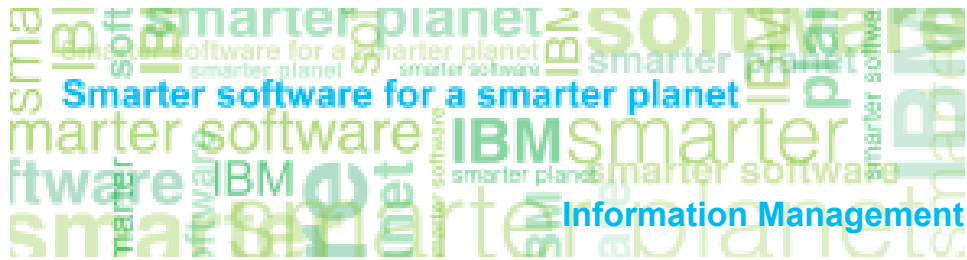
- **Reduces network traffic**
- **Access to features that exist only on the server**
- **Enforcement of business rules**

- **A remote stored procedure provides the most advantages:**
 - It splits the application logic and encourages an even distribution of the computational workload
 - It provides an easy way to call a remote program

Questions?

Summer/Fall 2010

E-mail: imschool@us.ibm.com
Subject: “DB2 Academic Workshop”





IBM DB2® 9.7

Working with Databases and Database Objects

Information Management Emerging Partnership and Technologies

IBM Canada Lab

Contents

CONTENTS	2
1. INTRODUCTION	3
2. OBJECTIVES	3
3. SUGGESTED READING	3
4. GETTING STARTED	3
4.1 ENVIRONMENT SETUP REQUIREMENTS	3
4.2 INITIAL STEPS	4
5. WORKING WITH DB2 DATABASES	5
5.1 FIRST STEPS	5
5.2 COMMAND LINE & ASSOCIATED DATABASE FILES	7
5.3 CONNECTING TO A DB2 DATABASE	10
5.3.1 USING CONNECT	10
5.3.2 CATALOGING A DB2 DATABASE	11
6. WORKING WITH DB2 DATA OBJECTS	13
6.1 TABLES	13
6.1.1 SCHEMAS.....	17
6.2 VIEWS	20
6.3 ALIASES	22
6.4 INDEXES	24
6.5 SEQUENCES	26
6.6 TRIGGERS	28
7. SUMMARY	30

1. Introduction

This module is designed to introduce you to instances and databases, to walk you through the database creation process, and to provide you with an overview of the various objects that can be developed once a database has been created.

2. Objectives

By the end of this lab, you will be able to:

- ▶ Create a DB2 database
- ▶ Catalog a database for use
- ▶ Examine and manipulate objects within a database

3. Suggested reading

IBM DB2 Database for Linux, UNIX, and Windows Information Center

<http://publib.boulder.ibm.com/infocenter/dstudio/v1r1m0/index.jsp>

A repository with information describing how to use the DB2 family of products and features

DB2 9 Fundamentals Certification Study Guide (Author: Roger E. Sanders)

Learn the basics and get ready for certification


4. Getting Started

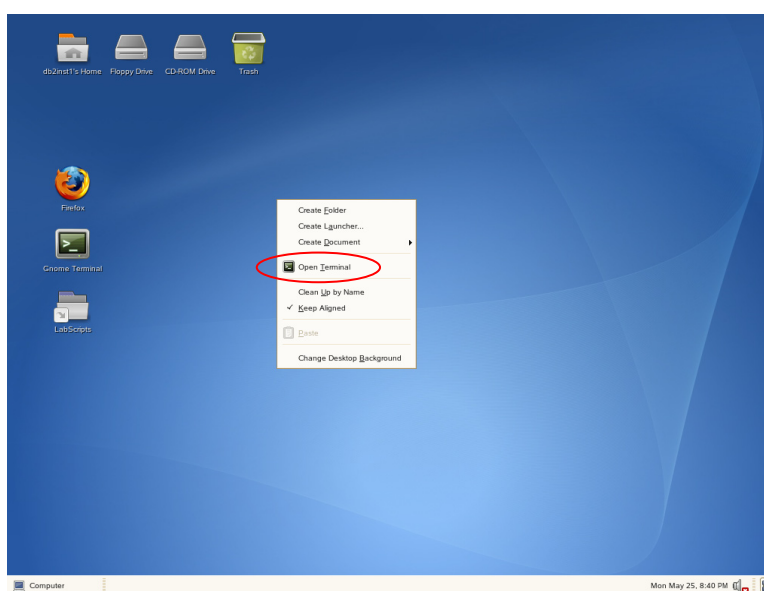
4.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 Academic Associate Bootcamp VMware image
- VMware Player 2.x or VMware Workstation 5.x or later

4.2 Initial Steps

1. Start the VMware image by clicking the  button in VMware.
2. At the login prompt, login with the following credentials:
 - ▶ Username: `db2inst1`
 - ▶ Password: `password`
3. Open a terminal window by right-clicking on the **Desktop** and choosing the **Open Terminal** item.



4. Ensure that the DB2 Database Manager has been started by issuing the following command at the prompt:

```
db2inst1@db2rules:~> db2start
```

Note: This command will only work if you logged in as the user `db2inst1`. If you accidentally logged in as another user, type `su - db2inst1` at the command prompt password: `password`.

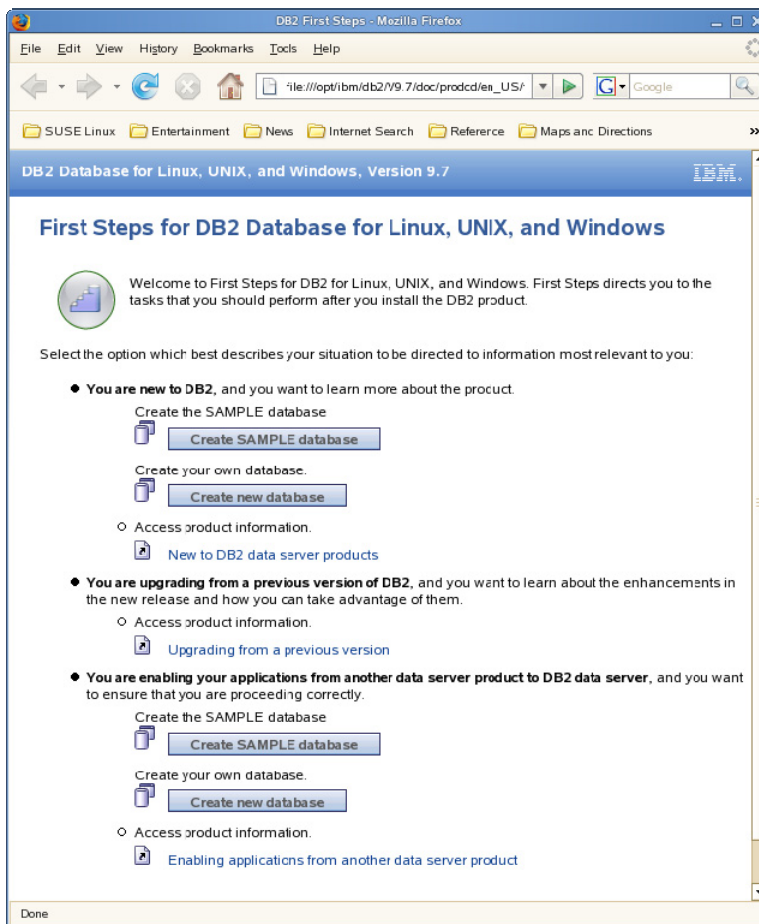
5. Throughout the lab, the SAMPLE database will be used to explore the features of DB2. To create the SAMPLE database we need to first remove the existing SAMPLE database by issuing the following command.

```
db2inst1@db2rules:~> db2 drop db sample
```

5. Working with DB2 Databases

5.1 First Steps

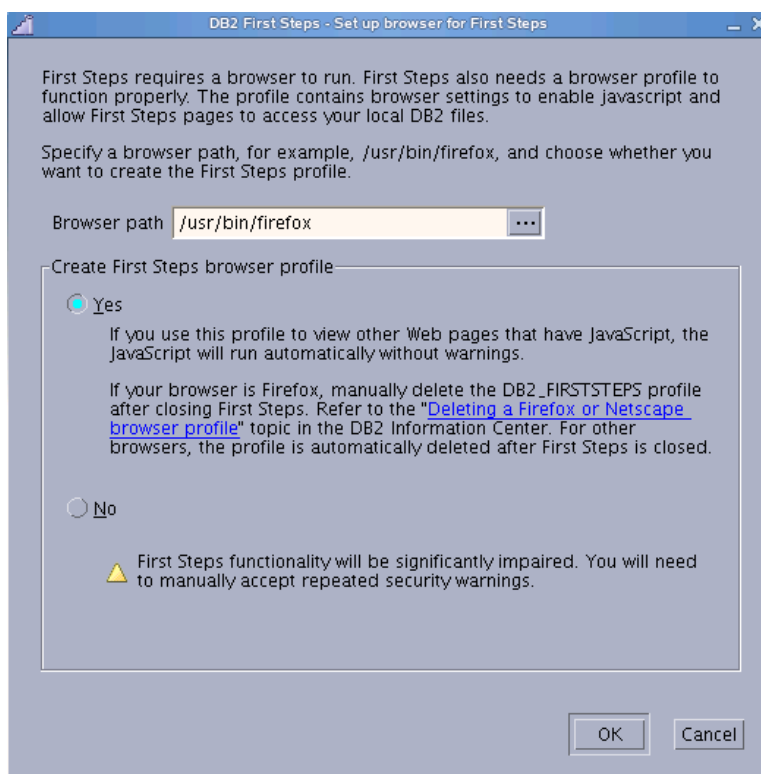
First Steps is a graphical tool that helps get you started with DB2. As part of the DB2 installation process, the First Steps panel is displayed allowing the user to generate a number of sample databases to work with:



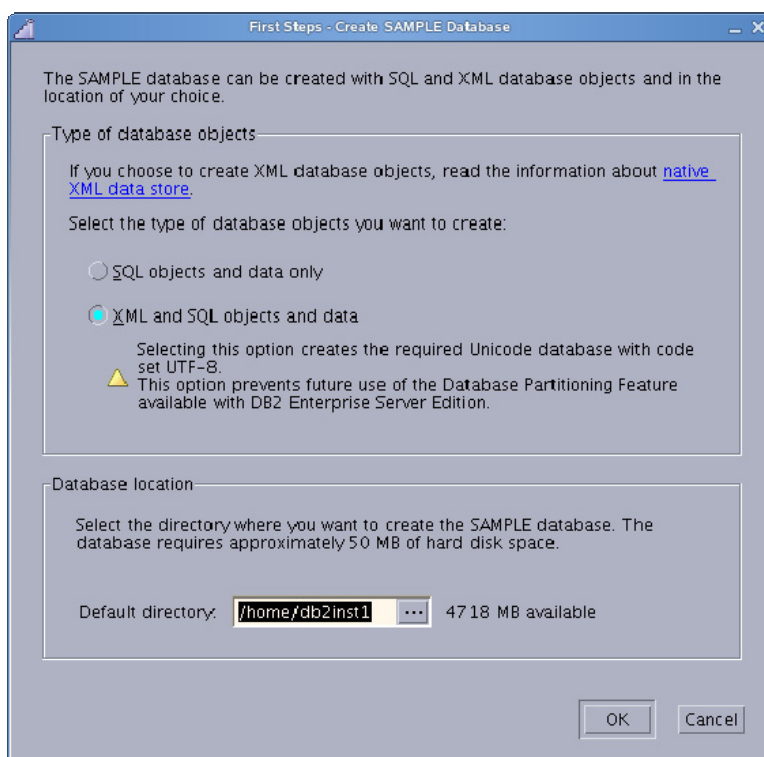
Most users will want to create the SAMPLE database and use that to explore the features of DB2. This panel can be invoked by issuing the command `db2fs` from a command-line prompt.

```
db2inst1@db2rules:~> db2fs
```

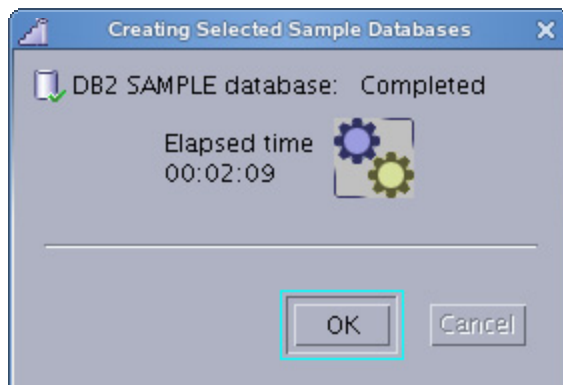
First Steps requires a browser and browser profile to run and function properly. Select Yes to create the browser profile and select OK to continue.



In addition, issuing the command `db2samp1` from a command-line prompt will also generate the SAMPLE database. Once the SAMPLE button has been selected, an additional panel is displayed to determine where the SAMPLE database will be created.



When creating the SAMPLE database, it is recommended that you select the XML and SQL objects and data option. This option will generate the database in UTF-8 (Unicode) format that will allow you to manipulate XML objects. If you do not select the XML option, you will not be able to add XML objects to your SAMPLE database.



Now let's move on to creating a DB2 database without a GUI.

5.2 Command Line & Associated Database Files

If you created the SAMPLE database using the First Steps method describe above, drop the sample database so we can see how to create it using the command line.

```
db2inst1@db2rules:~> db2 drop db sample
```

As mention in the previous section, we can issue the command `db2sampl` from a command-line prompt in order to also generate the SAMPLE database.

```
db2inst1@db2rules:~> db2sampl
```

When you create a database, DB2 creates a number of files. These files include log files, configuration information, history files, and three table spaces. These table spaces are:

- SYSCATSPACE: This is where the DB2 system catalog is kept that tracks all of the metadata associated with DB2 objects.
- TEMPSPACE1: A temporary work area where DB2 can place intermediate results.
- USERSPACE1: A place where all user objects (tables, indexes) reside by default.

Let's take a look at these tablespaces that DB2 created when we issued to create the SAMPLE database. Connect to the sample database (discussed in further details later) and list the tablespaces for the database by issuing the following commands:

```
db2inst1@db2rules:~> db2 connect to sample
```

```
db2inst1@db2rules:~> db2 list tablespaces
```

You should observe an output similar to the following:

```
db2inst1@db2rules:~> db2 list tablespaces

          Tablespaces for Current Database

Tablespace ID          = 0
Name                   = SYSCATSPACE
Type                   = Database managed space
Contents               = All permanent data. Regular table space.
State                  = 0x0000
  Detailed explanation:
    Normal

Tablespace ID          = 1
Name                   = TEMPSPACE1
Type                   = System managed space
Contents               = System Temporary data
State                  = 0x0000
  Detailed explanation:
    Normal

Tablespace ID          = 2
Name                   = USERSPACE1
Type                   = Database managed space
Contents               = All permanent data. Large table space.
State                  = 0x0000
  Detailed explanation:
    Normal
```

As mentioned, when a DB2 database instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance and other characteristics of the instance or database.

For simple applications, this default configuration may be sufficient for your needs. However you can modify these parameter values to improve performance and other characteristics of the instance or database.

Configuration files contain parameters that define values such as the resources allocated to the DB2 database products and to individual databases, and the diagnostic level. There are two types of configuration files:

- The **database manager configuration** file for each DB2 instance
- The **database configuration** file for each individual database

The *database manager configuration* file is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

```
db2inst1@db2rules:~> db2 get dbm cfg
```

You can also append the `show detail` option to view current and delayed values:

```
db2inst1@db2rules:~> db2 get dbm cfg show detail
```

A *database configuration* file is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes may be required, depending on the type of activity in a specific database.

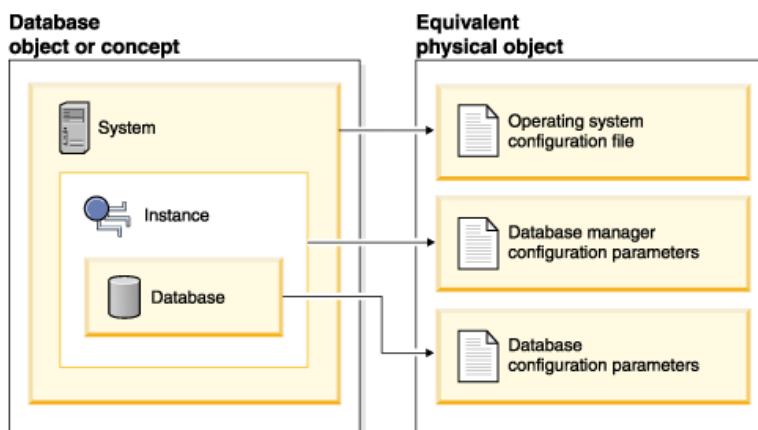
To view the database configuration, issue the following command (make sure you are connected to a database first):

```
db2inst1@db2rules:~> db2 get db cfg
```

Again, as with the database manager configuration, you can also append the `show detail` option to view current and delayed values:

```
db2inst1@db2rules:~> db2 get db cfg show detail
```

The following figure show the relationship between database objects and their corresponding configuration file:



5.3 Connecting to a DB2 Database

Before working with a database, a user or application program must establish a connection with that database. You connect to databases using the `CONNECT` statement.

5.3.1 Using `CONNECT`

Before you can issue a SQL statement, you have to connect to a database.

To connect to our sample database, enter the command:

```
db2inst1@db2rules:~> db2 CONNECT TO sample USER db2inst1 USING password
```

You can also connect to a database and have DB2 prompt you for the password by issuing the command:

```
db2inst1@db2rules:~> db2 CONNECT TO sample USER db2inst1
```

Or simply to connect to a database using the default user ID, issue the command:

```
db2inst1@db2rules:~> db2 CONNECT TO sample
```

At any point in time you can also issue the following command to view the currently active databases:

```
db2inst1@db2rules:~> db2 list active databases
```

You should see an output similar to the following:

```

Active Databases
Database name           = SAMPLE
Applications connected currently = 1
Database path          = /home/db2inst1/NODE0000/SQL00001/

```

Anytime that you need to terminate the connection to the database, you can issue the TERMINATE command:

```
db2inst1@db2rules:~> db2 terminate
```

However, before a client application can access a remote database, the database must be cataloged on the client. When you create a database, the database is automatically cataloged in the local database directory and another entry in the system database directory with a database alias that is the same as the database name. This is why **we were able to establish a connection to the sample database** with the commands specified above.

5.3.2 Cataloging a DB2 Database

So, why does a database have to be cataloged? Without this information, an application can't connect to a database! DB2 has multiple directories that are used to access databases. These directories allow DB2 to find databases known to it whether they are on the local system or a remote system. The system database directory contains a list and pointer indication where each of the known databases can be found.

To put an entry into any of these directories, a CATALOG command is used. To remove an entry, the UNCATALOG command is used.

As previously mention, we were able to connect to the SAMPLE database because the database was already cataloged by default upon creation.

To view the entries in the system databases directory, execute the command:

```
db2inst1@db2rules:~> db2 list database directory
```

The output should be similar to the following:

```
System Database Directory

Number of entries in the directory = 1

Database 1 entry:

Database alias           = SAMPLE
Database name           = SAMPLE
Local database directory = /home/db2inst1
Database release level  = d.00
Comment                 =
Directory entry type    = Indirect
Catalog database partition number = 0
Alternate server hostname =
Alternate server port number =
```

Here we can see the sample database cataloged on our system. This information is used to connect to the database.

However, if this information was not set here (ie, the database was not cataloged upon creation) we would not be able to connect to the database.

Let's take a look at how it would affect us if the database was not cataloged. Issue the UNCATALOG command on the SAMPLE database:

```
db2inst1@db2rules:~> db2 uncatalog database sample
```

Then try connecting to the SAMPLE database.

```
db2inst1@db2rules:~> db2 connect to sample
```

Notice that it is not possible. You will most likely receive a SQL1013N message:

```
SQL1013N  The database alias name or database name "SAMPLE"
could not be found.  SQLSTATE=42705
```

The fact is that the sample database and the files associated with it still exist within our system, however the information in the database directory does not exist for the DB2 client to establish a connection. You can verify this by checking the system database directory as before:

```
db2inst1@db2rules:~> db2 list database directory
```

```
SQL1057W  The system database directory is empty.  SQLSTATE=01606
```

Catalog the database by entering the following commands in the command line processor:

```
db2 catalog database database_name as database_alias on path/drive
```

where:

- *database_name* represents the name of the database you want to catalog.
- *database_alias* represents a local nickname for the database you want to catalog.
- *path/drive* specifies the path on which the database being cataloged resides.

To catalog the database called *sample* so that it has the local database alias *mysample*, enter the following command:

```
db2inst1@db2rules:~> db2 catalog database sample as mysample
```

Issue the following command to check the database directory for this new entry:

```
db2inst1@db2rules:~> db2 list database directory
```

System Database Directory

Number of entries in the directory = 1

Database 1 entry:

Database alias	= MYSAMPLE
Database name	= SAMPLE
Local database directory	= /home/db2inst1
Database release level	= d.00
Comment	=
Directory entry type	= Indirect
Catalog database partition number	= 0
Alternate server hostname	=
Alternate server port number	=

Now, since the database is cataloged and the information is back in the database directory, we can connect to it using the alias we have specified with the catalog statement above and issue our SQL statements.

```
db2inst1@db2rules:~> db2 connect to mysample
```

Before continuing on with the next section, issue a TERMINATE command to terminate the connection to the SAMPLE database.

```
db2inst1@db2rules:~> db2 terminate
```

6. Working with DB2 Data Objects

Before we get started with understanding and creating some basic and fundamental database objects, let us create a new database which we will use to highlight some of the concepts within this section.

```
db2inst1@db2rules:~> db2 create db testdb
```

Once the TESTDB database is created, issue a CONNECT statement, as show below, to establish a connection to the newly created database.

```
db2inst1@db2rules:~> db2 connect to testdb
```

6.1 Tables

A relational database presents data as a collection of tables. A table consists of data logically arranged in columns and rows (generally known as records).

Tables are created by executing the CREATE TABLE SQL statement. In its simplest form, the syntax for this statement is:

```
CREATE TABLE [TableName]
  ([ColumnName] [DataType], ...)
```

where:

- `TableName` identifies the name that is to be assigned to the table to be created.
- `ColumnName` identifies the unique name that is to be assigned to the column that is to be created.
- `DataType` identifies the data type to be assigned to the column to be created; the data type specified determines the kind of data values that can be stored in the column.

Thus, if you wanted to create a table named EMPLOYEES that has three columns, one of which is used to store numeric values and two that are used to store character string values, as shown below,

Column	Type
empid	INTEGER
name	CHAR(50)
Dept	CHAR(9)

you could do so by executing a CREATE TABLE SQL statement that looks something like this:

```
db2inst1@db2rules:~> db2 "CREATE TABLE employees
  (empid INTEGER,
   name CHAR(50),
   dept INTEGER)"
```

You can execute a DESCRIBE command to view the basic properties of the table:

```
db2inst1@db2rules:~> db2 describe table employees
```

Column name	Data type		Column Length	Scale	Nulls
	schema	Data type name			
EMPID	SYSIBM	INTEGER		4	0 Yes
NAME	SYSIBM	CHARACTER		50	0 Yes
DEPT	SYSIBM	INTEGER		4	0 Yes

3 record(s) selected.

But, now we notice that the department data type was specified as INTEGER not CHAR as originally intended. Therefore we need a way to change this data type from INTEGER to CHARACTER. We can do this using the alter statement.

Alter

```
db2inst1@db2rules:~> db2 "alter table employees alter column dept
set data type char(9)"
```

We can view the change by issuing the DESCRIBE command once again:

```
db2inst1@db2rules:~> db2 describe table employees
```

Column name	Data type		Column		
	schema	Data type name	Length	Scale	Nulls
EMPID	SYSIBM	INTEGER	4	0	Yes
NAME	SYSIBM	CHARACTER	50	0	Yes
DEPT	SYSIBM	CHARACTER	9	0	Yes

3 record(s) selected.

Notice now that the DEPT column is now using a CHARACTER data type opposed to an INTEGER data type.

So now that we have our table created to our preference, we can start to input data for the table to hold. We can do some very simple data manipulation language statements, such as insert, update, and delete.

Insert

Let's insert some basic data into our table with the following statement:

```
db2inst1@db2rules:~> db2 "INSERT INTO employees (EMPID, NAME, DEPT)
VALUES (1, 'Adam', 'A01 '),
(2, 'John', 'B01'),
(3, 'Peter', 'B01'),
(4, 'William', 'A01')"
```

You should receive an SQL0668N message:

```
SQL0668N Operation not allowed for reason code "7" on table
"DB2INST1.EMPLOYEES". SQLSTATE=57016
```

What does this mean? If we issue the "? SQL0668N" command, we can view the problem explanation and user response of the particular message.

```
db2inst1@db2rules:~> db2 "? SQL0668N"
```

```
SQL0668N Operation not allowed for reason code "<reason-code>" on table
"<table-name>".
```

Explanation:

```
Access to table "<table-name>" is restricted. The cause is based on the
following reason codes "<reason-code>":
```

```

...
7
    The table is in the reorg pending state. This can occur after
    an ALTER TABLE statement containing a REORG-recommended
    operation.
...
User response:
...
7
    Reorganize the table using the REORG TABLE command.
...

```

So, now we can conclude that the reason we cannot enter this data is that we did an ALTER on the table previously and it was placed in a reorg pending state. Therefore to resolve this issue, we should do as is recommended and “Reorganize the table using the REORG TABLE command:”

```
db2inst1@db2rules:~> db2 reorg table employees
```

Now try again and issue the insert statement as was shown previously:

```
db2inst1@db2rules:~> db2 "INSERT INTO employees (EMPID, NAME, DEPT)
VALUES (1, 'Adam', 'A01 '),
(2, 'John', 'B01'),
(3, 'Peter', 'B01'),
(4, 'William', 'A01')"
```

To verify the data has been inserted, you can issue a very basic SELECT statement on the table.

```
db2inst1@db2rules:~> db2 "select * from employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Peter	B01
4	William	A01

4 record(s) selected.

Update

We can also make update operations for our table. For example, Peter needs to move from department B01 to department A01. We can make the change in the table with the following update statement:

```
db2inst1@db2rules:~> db2 "update employees set dept='A01' where
name='Peter'"
```

Again, verify that the update has taken place.

```
db2inst1@db2rules:~> db2 "select * from employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Peter	A01
4	William	A01

4 record(s) selected.

Delete

Finally, let's try one more operation on our table, and that is to delete one of the entries. For example, William is no longer one of our employees; therefore we should delete him from our table. We can do so with the following DELETE statement:

```
db2inst1@db2rules:~> db2 "delete employees where name='William'"
```

Again, verify that the delete has taken place.

```
db2inst1@db2rules:~> db2 "select * from employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Peter	A01

3 record(s) selected.

6.1.1 Schemas

A schema is a collection of named objects. Schemas provide a logical classification of objects in the database. A schema can contain tables, views, nicknames, triggers, functions, packages, and other objects.

Most objects in a database are named using a two-part naming convention. The first (leftmost) part of the name is called the schema name or qualifier, and the second (rightmost) part is called the object name. Syntactically, these two parts are concatenated and delimited with a period:

```
schema_name.object_name
```


A schema is also an object in the database. A schema can be created in 2 ways mainly:

1. It can be implicitly created when another object is created, provided that the user has IMPLICIT_SCHEMA database authority.
2. It is explicitly created using the CREATE SCHEMA statement with the current user.

Let's take a look at the schema under which our EMPLOYEES table resides. We can do this by listing the tables within our database after we have established a connection.

```
db2inst1@db2rules:~> db2 list tables
```

The output will show a column specifying the Schema which each specific table belongs to:

Table/View	Schema	Type	Creation time
EMPLOYEES	DB2INST1	T	2010-03-30-16.37.05.046385

1 record(s) selected.

This is an example where the schema will be *implicitly* created when another object is created. So this means when creating our EMPLOYEES table, a schema name was created implicitly as we did not specify any. By default, DB2 is going to use the ID of the user who created the object as the schema name. This is shown in the output above.

Now, as mentioned, we can also *explicitly* create a schema and then assign objects to it upon creation. Let's take an example. We want to create a new schema named MYSCHEMA and create a new table, STORE under this newly created schema. We also want the authorization of this schema to have the authorization ID of our current user (db2inst1).

First, we need to create the schema using the CREATE SCHEMA command:

```
CREATE SCHEMA <name> AUTHORIZATION <name>
```

In our case,

```
db2inst1@db2rules:~> db2 CREATE SCHEMA myschema AUTHORIZATION db2inst1
```

To list all schemas available in the corresponding database you can issue the following command after a connection to the database is established:

```
db2inst1@db2rules:~> db2 select schemaname from syscat.schemata
```

SCHEMANAME
DB2INST1
MYSHEMA
NULLID
SQLJ

```

SYSCAT
SYSFUN
SYSIBM
SYSIBMADM
SYSIBMINTERNAL
SYSIBMTS
SYSPROC
SYSPUBLIC
SYSSTAT
SYSTOOLS

```

```
14 record(s) selected.
```

Next, we have to create the table which will belong to MYSCHEMA opposed to DB2INST1. We can do this using the following statement:

```
db2inst1@db2rules:~> db2 "CREATE TABLE myschema.store
                          (storeid INTEGER,
                           address CHAR(50)) "
```

(Note: The table name specified must be unique within the schema the table is to be created in.)

We can now list the tables for this new schema:

```
db2inst1@db2rules:~> db2 list tables for schema myschema
```

Table/View	Schema	Type	Creation time
STORE	MYSHEMA	T	2010-03-31-00.16.27.223473

```
1 record(s) selected.
```

We could have also issued the following command to see tables for ALL schemas:

```
db2inst1@db2rules:~> db2 list tables for all
```

Now, you may be wondering why anyone would want to explicitly create a schema using the CREATE SCHEMA statement. The primary reason for explicitly creating a schema has to do with **access control**. An explicitly created schema has an owner, identified either by the authorization ID of the user who executed the CREATE SCHEMA statement or by the authorization ID provided to identify the owner when the schema was created (db2inst1 in our case). The schema owner has the authority to create, alter, and drop any object stored in the schema; to drop the schema itself; and to grant these privileges to other users.

Finally, besides the benefit of access control, we can also have tables with the same name within a single database. This is because the name of each object needs to be unique only within its schema. Let's take a look.

We already have a table called EMPLOYEES within our db2inst1 schema; now lets create another table named employees but under myschema:

```
db2inst1@db2rules:~> db2 "CREATE TABLE myschema.employees
                          (storeid INTEGER,
                           address CHAR(50) ) "
```

It is successful because it is under a different schema and still within the same database!

In all these examples we used tables, but schemas also apply to objects such as: views, indexes, user-defined data types, user-defined functions, nicknames, packages, triggers, etc.

6.2 Views

A *view* is an alternative way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

A view can:

- Control access to a table
- Make data easier to use
- Simplify authorization by granting access to a view without granting access to the table
- Show only portions of data in the table
- Show summary data for a given table
- Combine two or more tables in meaningful ways
- Show only the selected rows that are pertinent to the process that uses the view

In this section we will create a view that will omit certain data from a table, thereby shielding some table data from end users.

In this example, we want to create a view of the EMPLOYEES which will omit the department employee information and rename the first two columns.

Meaning, this is what we want to achieve:

Column	Type
employee_id	INTEGER
first_name	CHAR(50)
Dept	CHAR(9)

To define the view, we must use the CREATE VIEW statement as follows:

```
db2inst1@db2rules:~> db2 "CREATE VIEW empview (employee_id, first_name)
                          AS SELECT EMPID, NAME
                          FROM employees"
```

Verify the view has been created:

```
db2inst1@db2rules:~> db2 list tables
```

Table/View	Schema	Type	Creation time
EMPLOYEES	DB2INST1	T	2010-03-30-16.37.05.046385
EMPVIEW	DB2INST1	V	2010-03-31-21.22.26.130570

2 record(s) selected.

Now, describe the view to ensure it is setup the way we originally intended:

```
db2inst1@db2rules:~> db2 describe table empview
```

Column name	Data type schema	Data type name	Column Length	Scale	Nulls
EMPLOYEE_ID	SYSIBM	INTEGER		4	0 Yes
FIRST_NAME	SYSIBM	CHARACTER		50	0 Yes

2 record(s) selected.

This matches what we have initially planned. Now for a final test, let's issue a SELECT * statement to retrieve all data from the view:

```
db2inst1@db2rules:~> db2 "select * from empview"
```

EMPLOYEE_ID	FIRST_NAME
1	Adam
2	John
3	Peter

3 record(s) selected.

Notice how the column names have changed appropriately as desired, and we cannot receive any data from the department column as we have not included it with our view. We could have also similarly created views which will combine data from different base tables and also based on other views or on a combination of views and tables. We will leave those out of our examples at this point in time but it is important to know that these options are possible.

Although views look similar to base tables, they do not contain real data. Instead, views refer to data stored in other base tables. Only the view definition itself is actually stored in the database. (In fact, when changes are made to the data presented in a view, the changes are actually made to the data stored in the base table(s) the view references.)

For example, update the view and verify that the underlying table contains the corresponding change.

```
db2inst1@db2rules:~> db2 "update empview
      SET FIRST_NAME='Piotr'
      WHERE employee_id=3"
```

Verify.

```
db2inst1@db2rules:~> db2 "SELECT * FROM employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Piotr	A01

3 record(s) selected.

6.3 Aliases

Aliases are alternative names for tables or views. An alias can be referenced the same way the table or view the alias refers to can be referenced.

Aliases are publicly referenced names, so no special authority or privilege is required to use them. However, access to the table or view that an alias refers to still requires appropriate authorization.

Aliases can be created by executing the CREATE ALIAS SQL statement.

Let us try and see how to create an alias (named EMPINFO) for our EMPLOYEES table which we have been working with in this section.

```
db2inst1@db2rules:~> db2 CREATE ALIAS empinfo FOR employees
```

Now we have this empinfo alias that we can use to reference the underlying employees table opposed to directly using the table name.

To view this alias, you can issue a command to list the tables:

```
db2inst1@db2rules:~> db2 list tables
```

Table/View	Schema	Type	Creation time
EMPINFO	DB2INST1	A	2010-07-05-11.05.48.124653
EMPLOYEES	DB2INST1	T	2010-07-05-16.37.05.046385
EMPVIEW	DB2INST1	V	2010-07-05-16.30.40.431174

3 record(s) selected.

Let's try a simple select statement with our newly created alias

```
db2inst1@db2rules:~> db2 "SELECT * FROM empinfo "
```

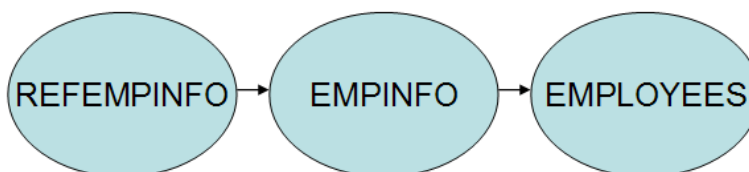
EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Peter	A01

3 record(s) selected.

As you can see, it provides the same output as selecting from the original table name, because after all it is referencing the same table.

Like tables and views, an alias can be created, dropped, and have comments associated with it. *Unlike* tables (but similar to views), aliases can refer to other aliases—a process known as **chaining**. We can take an example of this.

Right now we have our table EMPLOYEES and our EMPINFO alias. We can create this *chain* by creating another alias (REFEMPINFO) which will reference the EMPINFO alias. The situation can be represented by the following diagram:



To do this simply execute the CREATE ALIAS command again, but this time reference the EMPINFO alias opposed to the underlying EMPLOYEES base table name:

```
db2inst1@db2rules:~> db2 CREATE ALIAS refempinfo FOR empinfo
```

List the tables to see this new alias:

```
db2inst1@db2rules:~> db2 list tables
```

Table/View	Schema	Type	Creation time
EMPINFO	DB2INST1	A	2010-07-05-11.05.48.124653
EMPLOYEES	DB2INST1	T	2010-07-05-16.37.05.046385
EMPVIEW	DB2INST1	V	2010-07-05-16.30.40.431174
REFEMPINFO	DB2INST1	A	2010-07-05-16.42.36.059937

4 record(s) selected.

Again we can query this alias which will retrieve the data from the underlying table the name references:

```
db2inst1@db2rules:~> db2 "SELECT * FROM refempinfo"
```

EMPID	NAME	DEPT
-------	------	------

1	Adam	A01
2	John	B01
3	Peter	A01
3 record(s) selected.		

In conclusion, by using aliases, SQL statements can be constructed in such a way that they are independent of the qualified names that identify the base tables or views they reference.

6.4 Indexes

An *index* is an ordered set of pointers to rows of a table. DB2 can use indexes to ensure uniqueness and to improve performance by clustering data, partitioning data, and providing efficient access paths to data for queries. In most cases, access to data is faster with an index than with a scan of the data.

The three main purposes of indexes are:

- To improve performance.
- To ensure that a row is unique.
- To cluster the data.

An index is stored separately from the data in the table. Each index is physically stored in its own index space.

We will work with two examples of indexes in this section. One will illustrate how indexes can benefit to ensure that a row is unique and the other to show how we can improve performance of queries on the database.

In our previous example, we have our EMPLOYEES table with the following structure:

Column	Type
empid	INTEGER
name	CHAR(50)
Dept	CHAR(9)

When we created this table, we didn't define any primary key or unique constraints. Thus we can have multiple entries into our table with the same employee ID which is not a situation which we want to have. Therefore, we can use indexes to ensure that there are not two entries in the table with the same value for EMPID:

```
db2inst1@db2rules:~> db2 "CREATE UNIQUE INDEX unique_id
ON employees(empid) "
```

NOTE: You will receive the following message if you try to create this unique index with already duplicate entries for the key you are creating the index with. There cannot be duplicate entries when creating a unique index:

```
SQL0603N  A unique index cannot be created because the table
contains data that would result in duplicate index entries.
SQLSTATE=23515
```

Verify the index has been created with the DESCRIBE command:

```
db2inst1@db2rules:~> db2 DESCRIBE INDEXES FOR TABLE employees
```

The output will look similar to the following:

Index schema	Index name	Unique rule	Number of columns	Index type	Index partitioning
DB2INST1	UNIQUE_ID	U	1	RELATIONAL DATA	-

1 record(s) selected.

The unique rule column determines whether the index is unique or not. There are three different types of unique rules

- D = means duplicate allowed
- P = means primary index
- U = means unique index

Now, lets try to insert a row with a employee ID which already exists (ie, EMPID=3)

```
db2inst1@db2rules:~> db2 "INSERT INTO employees VALUES(3, 'William',
'A01')"
```

You should receive an error message saying that:

```
SQL0803N  One or more values in the INSERT statement, UPDATE
statement, or foreign key update caused by a DELETE statement are
not valid because the primary key, unique constraint or unique
index identified by "1" constrains table "DB2INST1.EMPLOYEES"
from having duplicate values for the index key.
SQLSTATE=23505
```

This means that our unique index is working properly because we cannot insert a row with an employee ID which already exists (the property on which we defined our index).

Also, as mentioned, index can help improve performance. Something that we can do with indexes is to also collect statistics. Collecting index statistics will allow the optimizer to evaluate whether an index should be used to resolve a query.

We can create an index to collect statistics automatically:


```
db2inst1@db2rules:~> db2 "CREATE INDEX idx
                        ON employees(dept) COLLECT STATISTICS "
```

You can view the index and its properties with the DESCRIBE command as before:

```
db2inst1@db2rules:~> db2 DESCRIBE INDEXES FOR TABLE employees
```

Except for changes in performance, users of the table are unaware that an index is in use. DB2 decides whether to use the index to access the table.

Be aware that indexes have both benefits and disadvantages. A greater number of indexes can simultaneously improve the access performance of a particular transaction and require additional processing for inserting, updating, and deleting index keys. After you create an index, DB2 maintains the index, but you can perform necessary maintenance, such as reorganizing it or recovering it, as necessary.

6.5 Sequences

A sequence is an object that is used to generate data values automatically.

Sequences have the following characteristics:

- Values generated can be any exact numeric data type that has a scale of zero.
- Consecutive values can differ by any specified increment value.
- Counter values are recoverable (reconstructed from logs when necessary).
- Values generated can be cached to improve performance.

In addition, sequences can generate values in one of three ways:

- By incrementing or decrementing by a specified amount, without bounds
- By incrementing or decrementing by a specified amount to a user-defined limit and stopping
- By incrementing or decrementing by a specified amount to a user-defined limit, and then cycling back to the beginning and starting again

Let's begin right away with creating a sequence named `emp_id` that starts at 4 and increments by 1, does not cycle, and caches 5 values at a time. To do so, we must issue the following statement:

```
db2inst1@db2rules:~> db2 "CREATE SEQUENCE emp_id
                        START WITH 4
                        INCREMENT BY 1
                        NO CYCLE
                        CACHE 5"
```

We will use this sequence to insert a new employee into our table without having to explicitly specify an individual employee ID; the sequence will take care of this for us.

To facilitate the use of sequences in SQL operations, two expressions are available: PREVIOUS VALUE and NEXT VALUE. The PREVIOUS VALUE expression returns the most recently generated value for the specified sequence, while the NEXT VALUE expression returns the next sequence value.

Create a new employee named Daniel in department B01 using the NEXT VALUE of our newly created sequence:

```
db2inst1@db2rules:~> db2 "INSERT INTO EMPLOYEES
                           VALUES (NEXT VALUE FOR emp_id, 'Daniel',
                                   'B01')"
```

Do a select statement of the table to view the results of how our sequence worked.

```
db2inst1@db2rules:~> db2 "SELECT * FROM employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Piotr	A01
4	Daniel	B01

4 record(s) selected.

We see that our sequence worked properly. Next time we use the NEXT VALUE statement we will increment by 1. For example

```
db2inst1@db2rules:~> db2 "INSERT INTO EMPLOYEES
                           VALUES (NEXT VALUE FOR emp_id, 'Stan',
                                   'B01')"
```

```
db2inst1@db2rules:~> db2 "SELECT * FROM employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Piotr	A01
4	Daniel	B01
5	Stan	B01

5 record(s) selected.

However, since we are caching 5 values at a time, we have to be careful because this value identifies the number of values of the identity sequence that are to be pre-generated and kept in memory. (Pre-generating and storing values in the cache reduces synchronous I/O to the log when values are generated for the sequence. However, in the event of a system failure, all cached sequence values that have not been used in committed statements are lost; that is, they can never be used.)

Let's take a look. Terminate the connection and reconnect to the database

```
db2inst1@db2rules:~> db2 terminate
```

```
db2inst1@db2rules:~> db2 connect to testdb
```

Now try the same operation we did previously to add an entry using the sequence and then verify with SELECT *.

```
db2inst1@db2rules:~> db2 "INSERT INTO EMPLOYEES
                           VALUES (NEXT VALUE FOR emp_id, 'Bill',
                                   'B01')"
```

```
db2inst1@db2rules:~> db2 "SELECT * FROM employees"
```

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
9	Bill	B01
3	Piotr	A01
4	Daniel	B01
5	Stan	B01

6 record(s) selected.

Notice that the next value is **9** NOT 6. Why? ..because we specified to cache the next 5 values in the memory before terminating the connection. Ie, we had values: 4, 5, 6, 7, 8 in memory. When the connection was lost (system failure) we lost these 5 values and now cannot use them again, thus we had to start the sequence with the next value after those which were cached.

6.6 Triggers

A trigger is used to define a set of actions that are to be executed whenever an insert, update, or delete operation is performed against a table or updatable view. Triggers are often used to enforce data integrity rules and business rules.

You can use triggers to:

- Validate input data
- Generate a value for a newly-inserted row
- Read from other tables for cross-referencing purposes
- Write to other tables for audit-trail purposes

Let's jump straight into an example. We want to create a trigger which increase the employee ID number (increasing EMPID by 1) each time a new person is added to the EMPLOYEES table.

To create a trigger from the command line, enter:

```
CREATE TRIGGER <name>
  <action> ON <table_name>
  <operation>
  <triggered_action>
```

The following statement creates a trigger which satisfies our condition above.

```
db2inst1@db2rules:~> db2 "CREATE TRIGGER new_emp
  AFTER INSERT ON EMPLOYEES
  FOR EACH ROW
  UPDATE employees SET empid = empid + 1"
```

Let's check the current values for our employees table so we can see the comparison of results after the trigger is fired.

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Piotr	A01
4	Daniel	B01
5	Stan	B01
9	Bill	B01

6 record(s) selected.

Now, let's see if it works! Issue a simple insert statement on the employees table and then SELECT * to verify if our trigger actually does update the employee ID by 1 when we insert into the table:

```
db2inst1@db2rules:~> db2 "INSERT INTO EMPLOYEES
  VALUES(10, 'Steve', 'B01') "
```

```
db2inst1@db2rules:~> db2 "SELECT * FROM EMPLOYEES"
```

EMPID	NAME	DEPT
2	Adam	A01
3	John	B01
4	Piotr	A01
10	Bill	B01
11	Steve	B01
5	Daniel	B01
6	Stan	B01

7 record(s) selected.

Notice that all the employee IDs have increased by a value of 1. This includes the newly added row because we created an AFTER trigger, meaning the increase operation took place after the row was inserted into the table.

We won't go through more example of this process because it can be quite repetitive, but it is very important to know that triggers can be specified for **INSERT, UPDATE, DELETE events**, and the possible **activation times are: BEFORE, AFTER, INSTEAD OF** of a particular query.

Benefits:

- **Faster application development:** Because a trigger is stored in the database, you do not have to code the actions that it performs in every application.
- **Easier maintenance:** Once a trigger is defined, it is automatically invoked when the table that it is created on is accessed.
- **Global enforcement of business rules:** If a business policy changes, you only need to change the trigger and not each application program.

Finish the lab by terminating all the connections and deleting the databases.

```
db2inst1@db2rules:~> db2 force applications all
db2inst1@db2rules:~> db2 drop db testdb
db2inst1@db2rules:~> db2 drop db mysample
```

7. Summary

This exercise introduced you to the objects that make up a DB2 database, and to the factors that affect how the database is created. You should now have a solid introduction to DB2 objects in terms of reasoning as to why we use them and how to create them as well. The remaining modules will make use of the information in this section; therefore it is quite important that everything is clear on these base concepts.

SQL in a nutshell

- **Standard language of relational database access is SQL (Structured Query Language), designed for accessing tabular data**
- **Data Definition Language (DDL)**
 - Defines properties of data objects
CREATE, ALTER, DROP, TRANSFER OWNERSHIP
- **Data Manipulation Language (DML)**
 - Used to retrieve, add, edit and delete data
SELECT, INSERT, UPDATE, DELETE
- **Data Control Language (DCL)**
 - Controls access to databases and data objects
GRANT, REVOKE
- **Transaction Control Languages (TCL)**
 - Groups DML statements into transactions that can collectively be applied to a database or undone in the event of a failure
COMMIT, ROLLBACK, SAVEPOINT

3

© 2010 IBM Corporation

Data Definition Language

- **To create, modify, delete objects in a database, SQL Data Definition Language (DDL) is used.**
- **DDL has four basic SQL statements:**
 - CREATE
 - ALTER
 - DROP
 - DECLARE

4

© 2010 IBM Corporation

CREATE Statement

Used to **CREATE** database objects

- Table
- Index
- Schema
- View
- User-defined function
- User-defined data type
- Buffer pool
- Stored procedures
- Trigger
- Alias
- Method
- Nickname
- Sequence
- Table space

CREATE TABLE <table name>...

CREATE SCHEMA <schema name>...

5

© 2010 IBM Corporation

ALTER Statement

Used to **CHANGE** database objects

- Table
- Table space
- Database partition
- Procedure
- Function
- Nickname
- Schema
- Sequence
- Type
- View
- Method
- User mapping
- Buffer pool
- Index

ALTER TABLE <table name>...

ALTER INDEX <index name>...

6

© 2010 IBM Corporation

DROP Statement

Used to **REMOVE** database objects

- Can drop any object created with **CREATE** **<database object >** and **DECLARE** **<table>** statements.

DROP TABLE **<table name>...**

DROP INDEX **<index name>...**

7

© 2010 IBM Corporation

Data Manipulation Language

- **Retrieving Data**
 - Select Statement
- **Inserting Data**
 - Insert Statement
- **Updating Data**
 - Update Statement
- **Deleting Data**
 - Delete Statement

8

© 2010 IBM Corporation

Retrieving Data – Select Statement

- The **SELECT** statement is used to retrieve data from a database table

–Syntax:

```
SELECT <%EXPRESSIONS%>  
FROM <%TABLE REFERENCES%>
```

- **Expressions: Column names, values, function calls etc.**
- **Table References is a source that returns tabular data**
 - Usually a list of table names
 - Subqueries or tabular functions are also accepted

Additional Clauses – Where, Order By, Group By

- **WHERE** allows you to filter the query
 - Specify filters based on expressions
- **GROUP BY** allows you to consolidate the query
 - Groups by the field you specify
 - This function allows you to utilize aggregate functions
- **HAVING** also allows you to filter the aggregate data
- **ORDER BY** allows you to sort the query
 - Specify the sort order of the returned set
 - Can specify ASC, DESC
 - e.g. ORDER BY NAME DESC
- **Clauses must appear in the order above after the select statement**

Select Statement

EXAMPLES

```
SELECT *
FROM EMPLOYEE
```

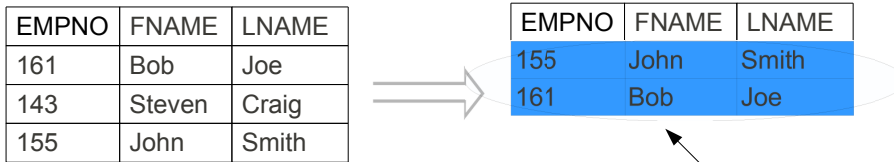
- Retrieves all fields in the employee table
- Note: * will return all fields



Select Statement

```
SELECT EMPNO, FNAME, LNAME
FROM EMPLOYEE WHERE EMPNO > 150
```

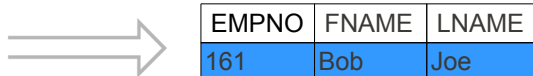
- Retrieves the EMPNO, FNAME, LNAME columns from the table EMPLOYEE where the EMPNO is greater than 150



No particular order!

```
SELECT EMPNO, FNAME, LNAME
FROM EMPLOYEE
WHERE EMPNO > 150 AND FNAME = 'Bob'
```

- Finding the Bobs' that have EMPNO greater than 150



Select Statement

Examples

```
SELECT MAX (SALARY) as MAX_SALARY, TITLE
FROM EMPLOYEE
GROUP BY TITLE
```

Renaming the column to MAX_SALARY

- Get the Highest salary for each title bracket

EMPNO	FNAME	LNAME	TITLE	SALARY
151	Joe	Brian	MGR	100 000
123	Steve	Jack	ENTRY	35 000
166	Tom	Jones	SENIOR	75 000
190	Eric	James	MGR	120 000
202	Kevin	Bob	ENTRY	45 000

Calculated from the Aggregate function MAX()

MAX_SALARY	TITLE
120 000	MGR
45 000	ENTRY
75 000	SENIOR

Non-Aggregate columns must be grouped together

13

© 2010 IBM Corporation

Select Statement

```
SELECT TITLE,AVG (SALARY) AS SALARY_AVG
FROM EMPLOYEE
GROUP BY TITLE
```

```
HAVING AVG (SALARY) > 50000
```

- Get all of the Titles that have an average salary above 50000

TITLE	SALARY_AVG
SENIOR	75 000
MGR	110 000

TITLE	SALARY_AVG
MGR	110 000
SENIOR	75 000

```
SELECT TITLE,AVG (SALARY) AS SALARY_AVG
FROM EMPLOYEE
GROUP BY TITLE
HAVING AVG (SALARY) > 50000
ORDER BY TITLE
```

- The above query, ordered by Title

14

© 2010 IBM Corporation

Inserting Data – Insert Statement

- The **INSERT** statement allows you to insert a single record or multiple records into a table

–Syntax:

```
INSERT INTO <%TABLE REFERENCE%>
( <%EXPRESSION%> )
VALUES
(%VALUES%)
```

OR

```
INSERT INTO <%TABLE REFERENCE%>
( <%EXPRESSION%> **)
SELECT <%EXPRESSION%>
FROM <%TABLE REFERENCE%>
**Explicit mapping is not necessary
```

15

© 2010 IBM Corporation

Insert Statement

Examples

```
INSERT INTO EMPLOYEE (FNAME , LNAME)
VALUES ('Joe' , 'Bob')
```

–Inserts the name Joe Bob into the table Employee

TITLE	FNAME	LNAME
NULL	Joe	Bob

Value was not provided, a NULL (if column is nullable) will place hold

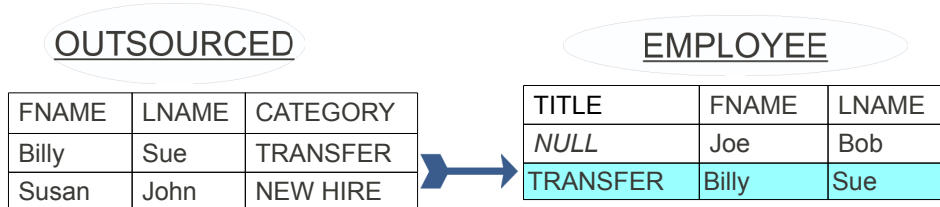
16

© 2010 IBM Corporation

Insert Statement

```
INSERT INTO EMPLOYEE (FNAME, LNAME, TITLE)
SELECT FNAME, LNAME, CATEGORY
FROM OUTSOURCED
WHERE CATEGORY = 'TRANSFER'
```

- Selects the people from table Outsourced with the category field as Transfer into the Employee Table
 - Note: If both tables have the exact same fields then an explicit mapping is not required.



17

© 2010 IBM Corporation

Data Modifications – Update Statement

- The **UPDATE** statement is used to update existing records in a table

-Syntax:

```
UPDATE <%TABLE REFERENCE%>
SET <%EXPRESSION%> = %VALUE%
```

- **UPDATE** uses the same **WHERE** clause as the **SELECT**

18

© 2010 IBM Corporation

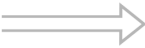
Update Statement

Example

```
UPDATE EMPLOYEE
SET BONUS = SALARY/10
```

- Update every entry in the employee table and set the bonus field to 10% of the salary field

SALARY	BONUS
50 000	NULL
35 000	20 000
100 000	NULL
150 000	NULL



SALARY	BONUS
50 000	5 000
35 000	3 500
100 000	10 000
150 000	15 000


19

© 2010 IBM Corporation


Update Statement

```
UPDATE EMPLOYEE
SET TITLE = 'Executive'
WHERE TITLE = 'Manager' and Tenure = 10
```


- Update title of Employees who has tenure of 10 and is currently a manager in the Employee Table



FNAME	LNAME	TITLE	TENURE
Stan	Jones	Manager	5
Bob	Billy	Executive	12
Joe	Moe	Senior	14
Susie	Susan	Manager	10



FNAME	LNAME	TITLE	TENURE
Stan	Jones	Manager	5
Bob	Billy	Executive	12
Joe	Moe	Senior	14
Susie	Susan	Executive	10



20

© 2010 IBM Corporation

Rules

▪ INSERT Rules

- **INSERT** rule is implicit when a foreign key is specified.
- A row can be inserted at any time into a parent table without any action being taken in dependent table.
- A row cannot be inserted into dependent table unless there is a row in parent table with a parent key value equal to foreign key value of row being inserted, unless foreign key value is null.
- If an **INSERT** operation fails for one row during an attempt to insert more than one row, all rows inserted by the statement are removed from the database.

▪ UPDATE Rules

- **RESTRICT** – Update for parent key will be rejected if a row in dependent table matches original values of key.
- **NO ACTION** – Update operation for parent key will be rejected if any row in dependent table does not have a corresponding parent key when update statement is completed (**default**).

Data Modifications – Delete Statement

- The **DELETE** statement is used to remove rows in a table

DELETE <%TABLE REFERENCE%>

- **DELETE** uses the same **WHERE** clause as a **SELECT**
- Delete removes data only not the table itself

Delete Statement

Example

DELETE EMPLOYEE

- All entries inside employee are deleted

EMPNO	STATUS
1232	RETIRED
141	ACTIVE
51	RETIRED

⇒

EMPNO	STATUS
-------	--------

DELETE EMPLOYEE WHERE STATUS = 'Retired'

- Remove all the entries where the status is retired

EMPNO	STATUS
1232	RETIRED
141	ACTIVE
51	RETIRED

⇒

EMPNO	STATUS
141	ACTIVE

23

© 2010 IBM Corporation

DELETE Rules

- **RESTRICT**
 - Prevents any row in parent table from being deleted **if any dependent rows are found.**
- **NO ACTION (default)**
 - Enforces the presence of a parent row for every child after all the referential constraints are applied.
 - The difference between **NO ACTION** and **RESTRICT** is based on when constraint is enforced.
- **CASCADE**
 - Implies that deleting a row in parent table **automatically deletes any related rows** in dependent table.
- **SET NULL**
 - Ensures that deletion of a row in parent table **sets values of foreign key in any dependent row to null (if nullable).**
 - Other parts of row are unchanged.

24

© 2010 IBM Corporation

Joins

- Linking of one or more tables

```
SELECT *
FROM EMPLOYEE E
INNER JOIN TEAM T ON E.TEAMID = T.ID
```

- Several Types of joins:
 - Cartesian Product
 - Inner Join
 - Left Outer Join
 - Right Outer Join
 - Full Outer Join

25

© 2010 IBM Corporation

Cartesian Product

- Joins every entry to a corresponding entry regardless of criteria
- Also called a Cross Join
- This is particularly useful if you need all possible permutations
 - For Example:

EMPNO	ROLE	EXT	TITLE	X	EMPNO	ROLE	EXT	TITLE
-------	------	-----	-------	---	-------	------	-----	-------

All possible employee partners

```
SELECT * FROM DETAIL D1, DETAIL D2
WHERE D1.EMPNO <> D2.EMPNO
```

- Need to be careful – 30,000 X 100,000 = 3,000,000,000 Rows!

26

© 2010 IBM Corporation

Cross Join Example

SELECT * FROM INFO I, DETAIL D

INFO (I)

EMPNO	FNAME	LNAME
001	Emily	Stevens
002	John	Doe
003	James	Smith

CROSS JOIN

DETAIL (D)

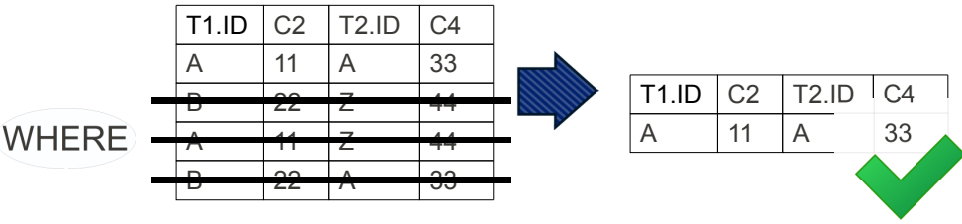
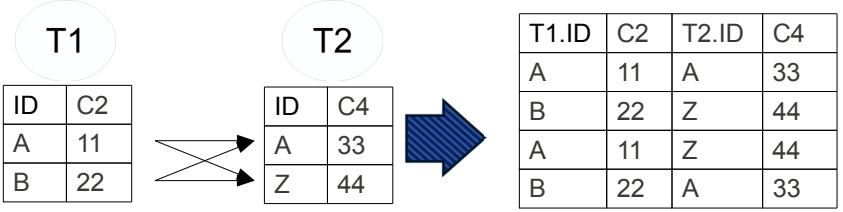
EMPNO	ROLE	EXT	TITLE
001	MGR	445	Mrs.
004	EXEC	101	Dr.



EMPNO	FNAME	LNAME	EMPNO	ROLE	EXT	TITLE
001	Emily	Stevens	001	MGR	445	Mrs.
002	John	Doe	004	EXEC	101	Dr.
003	James	Smith	001	MGR	445	Mrs.
001	Emily	Stevens	004	EXEC	101	Dr.
002	John	Doe	001	MGR	445	Mrs.
003	James	Smith	004	EXEC	101	Dr.

Walk Through – Cartesian Product

SELECT *
FROM TABLE T1, TABLE T2
WHERE T1.ID = T2.ID



INNER Join

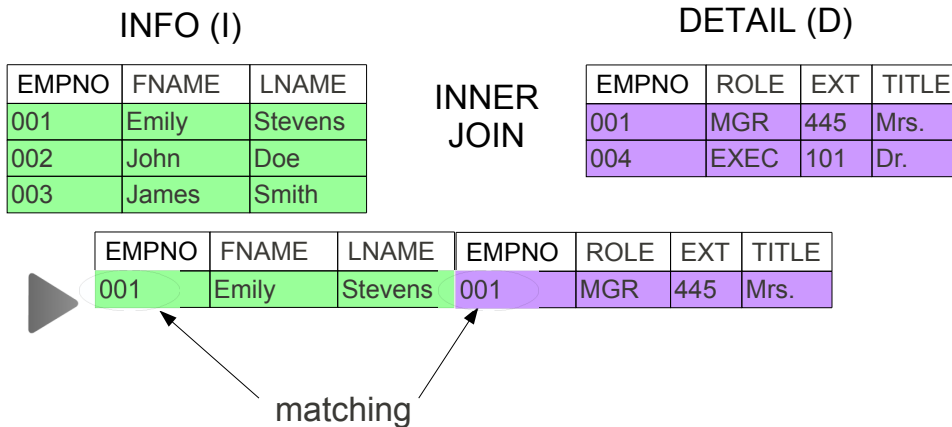
- **INNER join requires a match on both sides**
- **Both entries must satisfy the condition that was set in the ON clause**
- **This is useful if you need to create views that have relevance to each other**
 - For example:
 - Require Descriptions from a Look-up table linking with an ID
- **Will return nothing if there is nothing in common**

29

© 2010 IBM Corporation

Inner Join Example

```
SELECT *
FROM INFO I
INNER JOIN DETAIL D ON I.EMPNO =
D.EMPNO
```

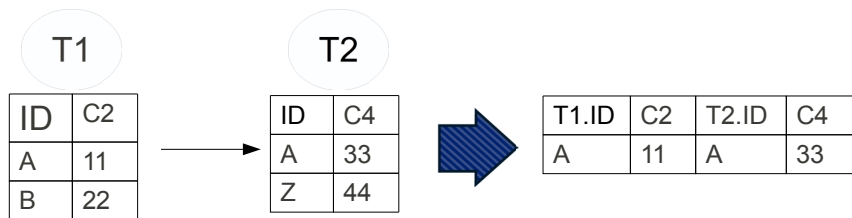


30

© 2010 IBM Corporation

Walk Through – Inner Join

```
SELECT *
FROM TABLE T1 INNER JOIN T2
ON T1.ID = T2.ID
```



31

© 2010 IBM Corporation

Right Outer Join

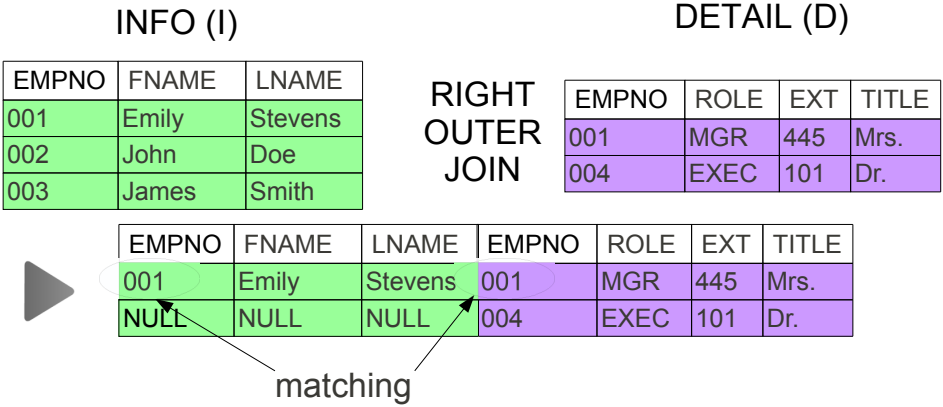
- **Right Outer Join joins tables on the condition that it matches the fields that are specified in the ON clause**
 - Right Outer – Condition must be met for the Right Side
- **If condition not met**
 - take entry from the designated side and fill NULL in for the non-designated side

32

© 2010 IBM Corporation

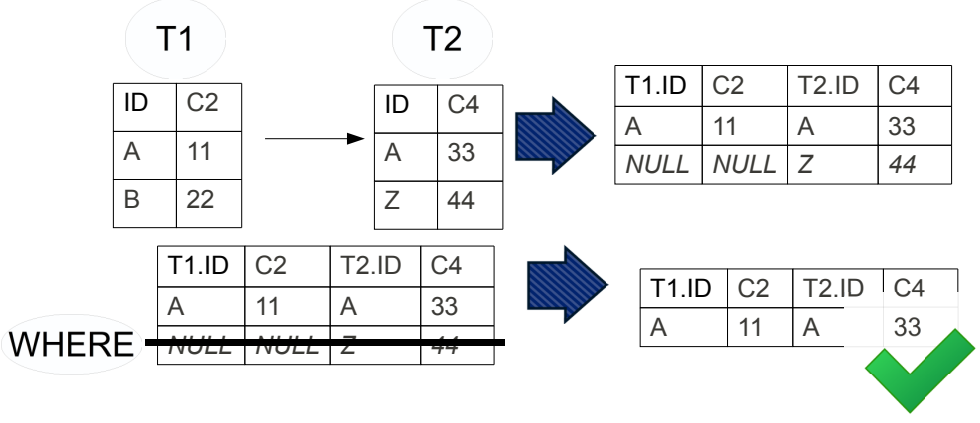
Right Outer Join Example

```
SELECT *
FROM INFO I
RIGHT OUTER JOIN DETAIL D ON I.EMPNO =
D.EMPNO
```



Walk Through – Right Outer Join

```
SELECT *
FROM TABLE T1 RIGHT OUTER JOIN TABLE T2
ON T1.ID = T2.ID
WHERE T1.ID IS NOT NULL
```



WHERE ~~NULL NULL Z 44~~

Left Outer Join

- **Left Outer Join joins tables on the condition that it matches the fields that are specified in the ON clause**
 - Left Outer – Condition must be met for the Left Side
- **If condition not met**
 - take entry from the designated side and fill NULL in for the non-designated side

35

© 2010 IBM Corporation

Left Outer Join Example

```
SELECT *
FROM INFO I
LEFT OUTER JOIN DETAIL D ON I.EMPNO =
D.EMPNO
```

INFO (I)

EMPNO	FNAME	LNAME
001	Emily	Stevens
002	John	Doe
003	James	Smith

DETAIL (D)

EMPNO	ROLE	EXT	TITLE
001	MGR	445	Mrs.
004	EXEC	101	Dr.

LEFT
OUTER
JOIN

EMPNO	FNAME	LNAME	EMPNO	ROLE	EXT	TITLE
001	Emily	Stevens	001	MGR	445	Mrs.
002	John	Doe	NULL	NULL	NULL	NULL
003	James	Smith	NULL	NULL	NULL	NULL



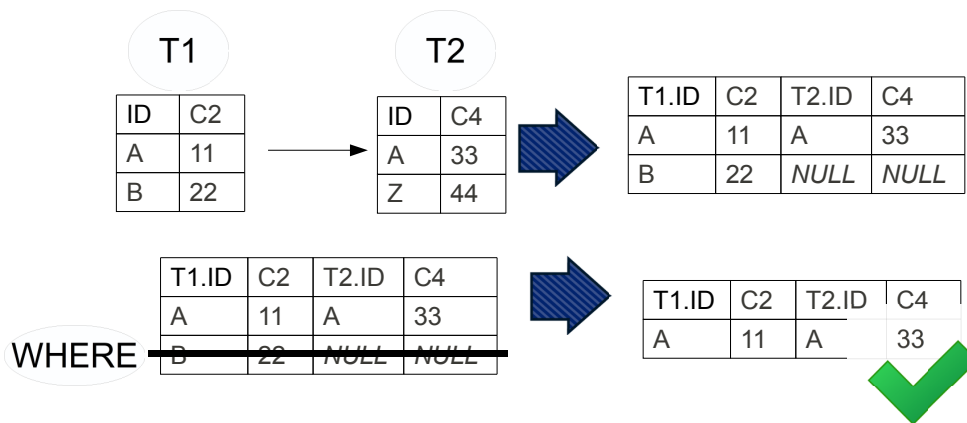
matching

36

© 2010 IBM Corporation

Walk Through – Left Outer Join

```
SELECT *
FROM TABLE T1 LEFT OUTER JOIN TABLE T2
ON T1.ID = T2.ID
WHERE T2.ID IS NOT NULL
```



37

© 2010 IBM Corporation

Full Outer Join

- Will match records on columns where there is a match
- If condition is not met
 - The side that the condition is not met will fill with NULLs
 - The side that the condition is met will be replicated
 - Repeats for both sides
- This is useful for when you require information from both tables that have relevance with each other
 - For example:
 - Find the cars that are blue, cars that are white or neither

38

© 2010 IBM Corporation

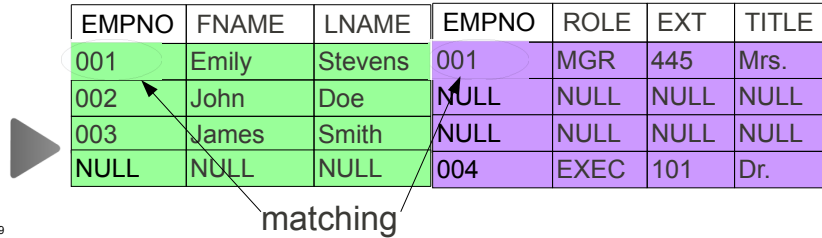
Full Outer Join Example

```
SELECT *
FROM INFO I
FULL OUTER JOIN DETAIL D ON I.EMPNO =
D.EMPNO
```

EMPNO	FNAME	LNAME
001	Emily	Stevens
002	John	Doe
003	James	Smith

EMPNO	ROLE	EXT	TITLE
001	MGR	445	Mrs.
004	EXEC	101	Dr.

FULL OUTER JOIN

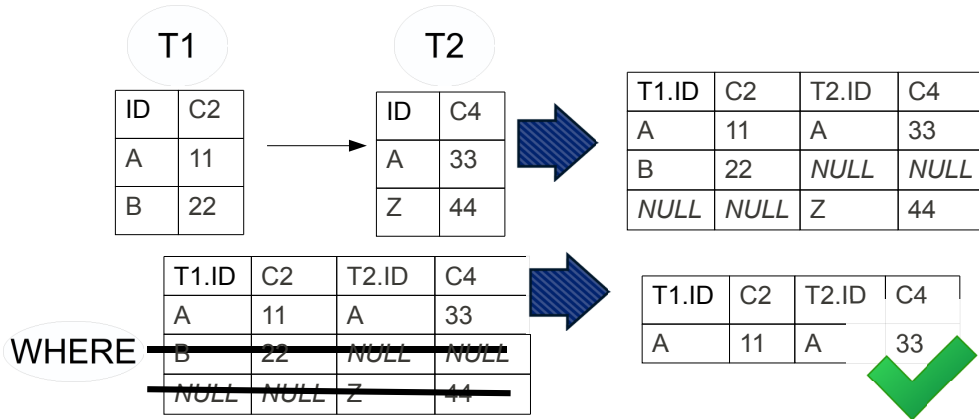


39

© 2010 IBM Corporation

Walk Through – Full Outer Join

```
SELECT *
FROM TABLE T1 FULL OUTER JOIN TABLE T2
ON T1.ID = T2.ID
WHERE T2.ID IS NOT NULL AND T1.ID IS NOT NULL
```



40

© 2010 IBM Corporation

Challenge Question

```

SELECT E.EMPNO, A.RANK, T.DESCRPTION
FROM EMPLOYEE E
LEFT OUTER JOIN ATTRIBUTE A ON E.AID = A.ID
FULL OUTER JOIN TYPE T ON E.TID = T.ID
WHERE A.RANK = 'Major'
    
```

Employee			Type		Attribute	
EMPNO	AID	TID	ID	DESCRIPTION	ID	RANK
9021	1	3	1	Army	1	Private
4533	3	3	2	Marine	2	Sergeant
1345	1	1	3	Air Force	3	Captain
9038	4	1			4	Major
					5	General

RESULT:

EMPNO	RANK	DESCRIPTION
9038	Major	Army

Challenge Question

```

SELECT RANK
FROM EMPLOYEE
INNER JOIN ATTRIBUTE
ON EMPLOYEE.AID = ATTRIBUTE.ID
GROUP BY RANK
HAVING COUNT (EMPNO) > 1
    
```

Employee			Attribute	
EMPNO	AID	TID	ID	RANK
9021	1	3	1	Private
4533	3	3	2	Sergeant
1345	1	1	3	Captain
9038	4	1	4	Major
			5	General

RESULT:

RANK
Private

Challenge – Prove they are equivalent

How are they equivalent?

```
SELECT *
FROM TABLE T1 INNER JOIN T2
ON T1.ID = T2.ID
    IS EQUIVALENT TO
SELECT *
FROM TABLE T1, TABLE T2
WHERE T1.ID = T2.ID
    IS EQUIVALENT TO
SELECT *
FROM TABLE T1 RIGHT OUTER JOIN TABLE T2
ON T1.ID = T2.ID
WHERE T1.ID IS NOT NULL
```

Aliases

- **Aliases** : a reference to an object

```
SELECT E.FNAME AS 'First Name'
FROM EMPLOYEE E
```

- **E** is now referenced to the table Employee
- **AS 'First Name'** lets the column be returned as First Name rather than FNAME
- Important when distinguishing objects from different sources that may have conflicting names

Data Control Language (DCL)

- **Data Control Language SQL statements control the security and permissions of the objects or parts of the database(s)**
- **GRANT** - gives user's access privileges to database
- **REVOKE** - withdraw access privileges given with the GRANT command

GRANT SELECT ON <table name>...

REVOKE ALL ON <table name>...



45

© 2010 IBM Corporation

Transaction Control Statements (TCL)

- **Transaction Control statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.**
- **COMMIT** - save work done
- **SAVEPOINT** - identify a point in a transaction to which you can later roll back
- **ROLLBACK** - restore database to original since the last COMMIT
- **SET TRANSACTION** - Change transaction options like isolation level and what rollback segment to use



46

© 2010 IBM Corporation



IBM DB2[®] 9.7

**Understanding SQL
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

1. INTRODUCTION TO SQL	3
2. OBJECTIVES OF THIS LAB	3
3. SETUP AND START DB2	4
3.1 Environment Setup Requirements	4
3.2 Login to the Virtual Machine	4
3.3 SAMPLE Database	5
3.4 Launching Data Studio	6
4. EXPLORING A DATABASE WITH IBM DATA STUDIO	7
4.1 Filtering by Schema	8
4.2 Viewing Data Object Properties	11
5. DATA MANIPULATION LANGUAGE	12
5.1 Querying Data	13
5.1.1 Retrieving All Rows from a Table	13
5.1.2 Retrieving Rows using SELECT Statement	15
5.1.3 Sorting the Results	18
5.1.4 Aggregating Information	19
5.1.5 Retrieving Data from Multiple Tables (Joins)	20
5.2 Insert, Update and Delete	24
5.2.1 INSERT	24
5.2.2 UPDATE	26
5.2.3 DELETE	27
6. ANSWERS	28

1. Introduction to SQL

Structured Query Language (SQL) is a standardized language used to work with database objects and the data they contain. Using SQL, you can define, alter, and delete database objects, as well as insert, update, delete, and retrieve data values stored in database tables.

SQL has a defined syntax and a set of language elements. Most SQL statements can be categorized according to the functions they perform; SQL statements fall under one of the following categories:

- **Data Definition Language (DDL)**
 - Defines properties of data objects
CREATE, ALTER, DROP
- **Data Manipulation Language (DML)**
 - Used to retrieve, add, edit and delete data
SELECT, INSERT, UPDATE, DELETE
- **Data Control Language (DCL)**
 - Used to control access to databases and data objects
GRANT, REVOKE
- **Transaction Control Language (TCL)**
 - Groups DML statements into transactions that can collectively be applied to a database or undone in the event of a failure
COMMIT, ROLLBACK, SAVEPOINT

2. Objectives of This Lab

By the end of this lab, you will be able to use **IBM Data Studio** environment to write and execute **DML** SQL statements (**SELECT**, **INSERT**, **UPDATE** and **DELETE**).

3. Setup and Start DB2

3.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in the **VMware Basics and Introduction** module.

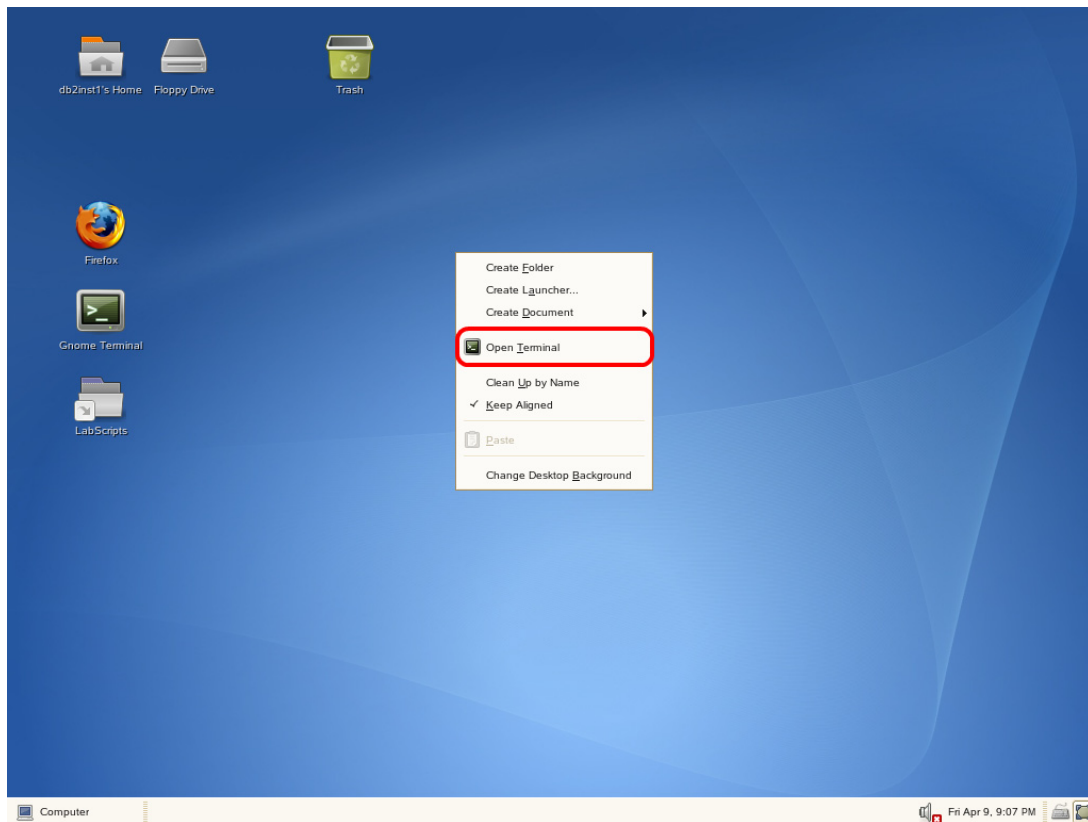
3.2 Login to the Virtual Machine

1. Start the VMware image by clicking the button  in VMware.

2. At the login prompt, login with the following credentials:

- Username: **db2inst1**
- Password: **password**

3. Open a terminal window as follows by right-clicking on the **Desktop** and choosing the **Open Terminal** item.



4. Start the Database Manager by entering the following command:

```
db2inst1@db2rules:~> db2start
```

3.3 SAMPLE Database

Throughout this lab, we will use **IBM Data Studio** to execute the exercises.

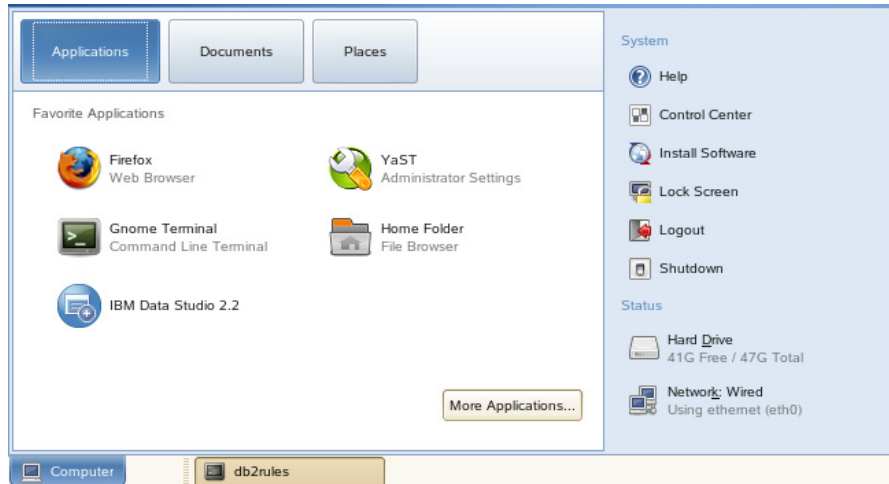
For executing this lab, you will need the DB2's sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the **SAMPLE** database:

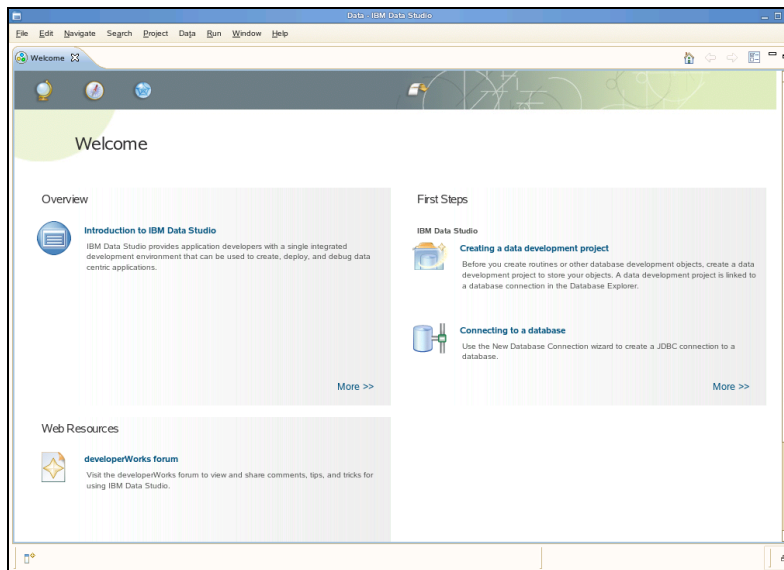
```
db2 force applications all
db2 drop db sample
db2 drop db mysample
db2samp1
```


3.4 Launching Data Studio

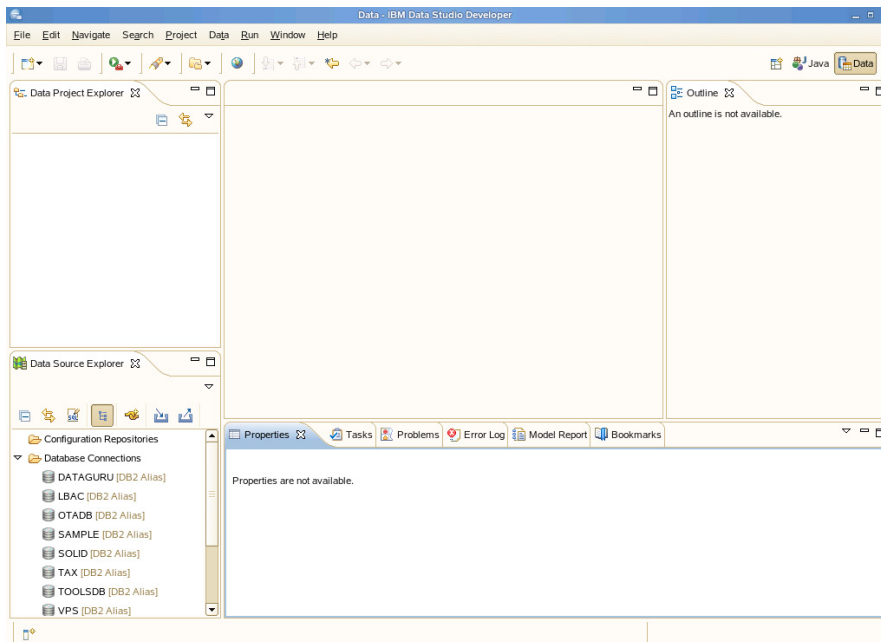
1. Click on the **Computer** button in the bottom left corner of the screen, and select **Data Studio 2.2**.



2. In the **Select a workspace** dialog, accept the default path. Click **OK**.
3. Data Studio will now start with the Welcome homepage.





4. Minimize this window by clicking the minimize button () located at the top right to bring you into the **Data** perspective as shown below.

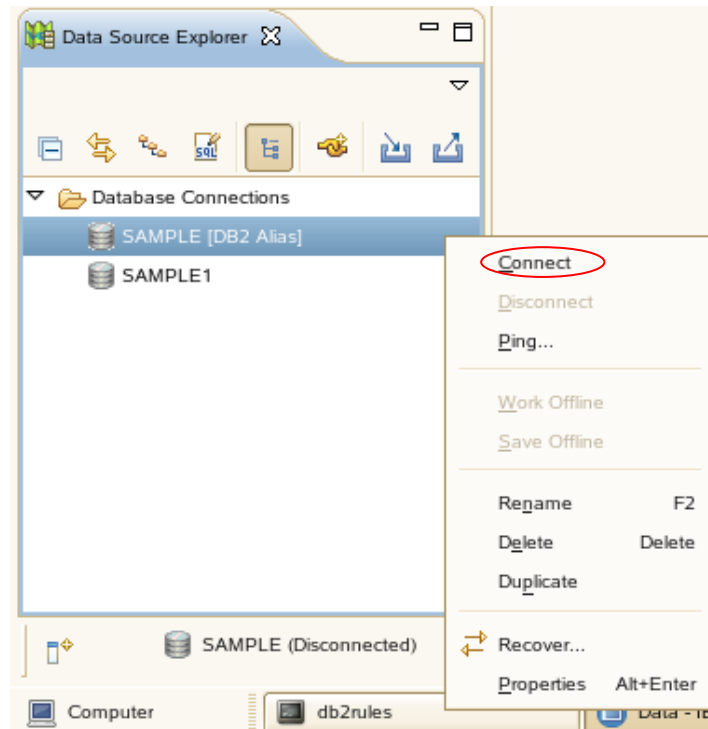


4. Exploring a Database with IBM Data Studio

Before you can do anything productive with **IBM Data Studio**, a connection must be established to a database. The **Data Source Explorer** view in Data Studio allows you to do this. From this view it is possible to interact with and manipulate database artifacts. Now let's connect to **SAMPLE** database.


Note: In the **Data Source Explorer** make sure the *Show the Data Source Explorer Contents in flat view* icon is selected (this icon  should change to this: ) , otherwise you may have problem directing to the right tables.

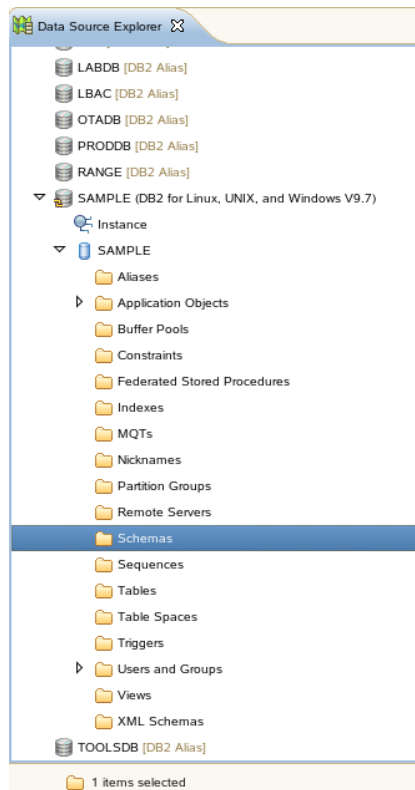
1. In the Data Source Explorer, right-click on the **SAMPLE [DB2 Alias]** node, and select **Connect**.
2. If the **Database Authorization** window appears, enter the following credentials:
 - Username: **db2inst1**
 - Password: **password**



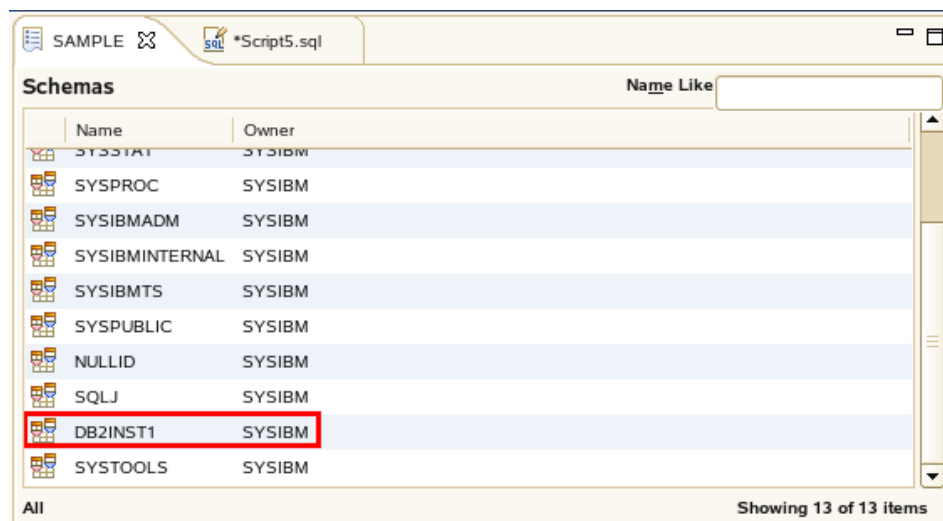
3. Click **OK**. Notice that the **SAMPLE [DB2 Alias]** connection icon now has a little yellow chain, signifying that the connection has been established.

4.1 Filtering by Schema

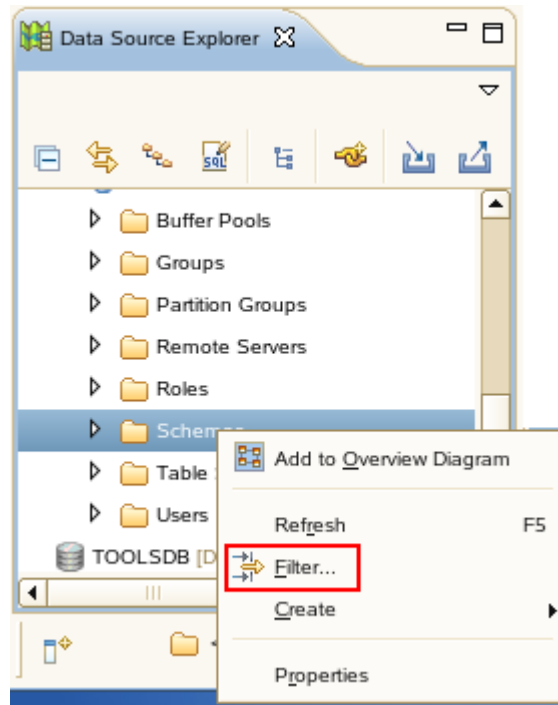
1. In the Data Source Explorer, expand the following nodes by clicking the  icons beside them: **SAMPLE [DB2 for Linux...]** > **SAMPLE** > **Schemas**.



Notice that once you click on **Schemas** there are many different schemas listed in the main view under SAMPLE > Schemas tab: DB2INST1, NULLID, SQLJ, etc. Because we will only be working with the **DB2INST1** schema, let's filter our list to show only this schema.

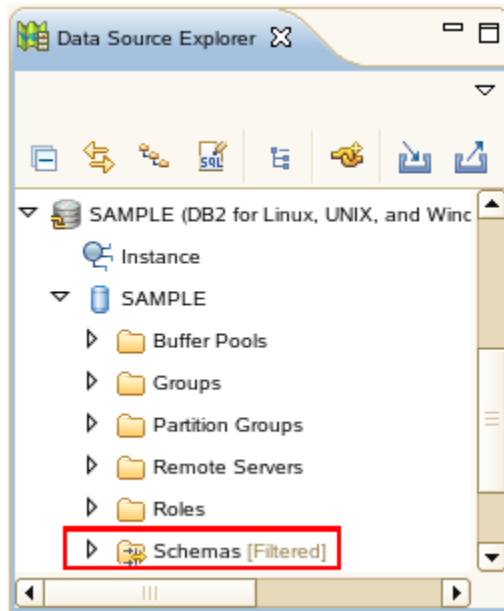


2. In the **Data Source Explorer**, right-click on the **Schemas** node and select **Filter**. The **Filter** dialog will appear, allowing you to filter either by typing in the name of a schema (or a portion thereof), or by selecting from a list of schemas. For this lab, we will filter by selection.



- Uncheck the **Disable filter** checkbox.
- Select the **Selection** radio button.
- Select **Include selected items** in the drop down list.
- Check the **DB2INST1** checkbox.
- Click **OK**.

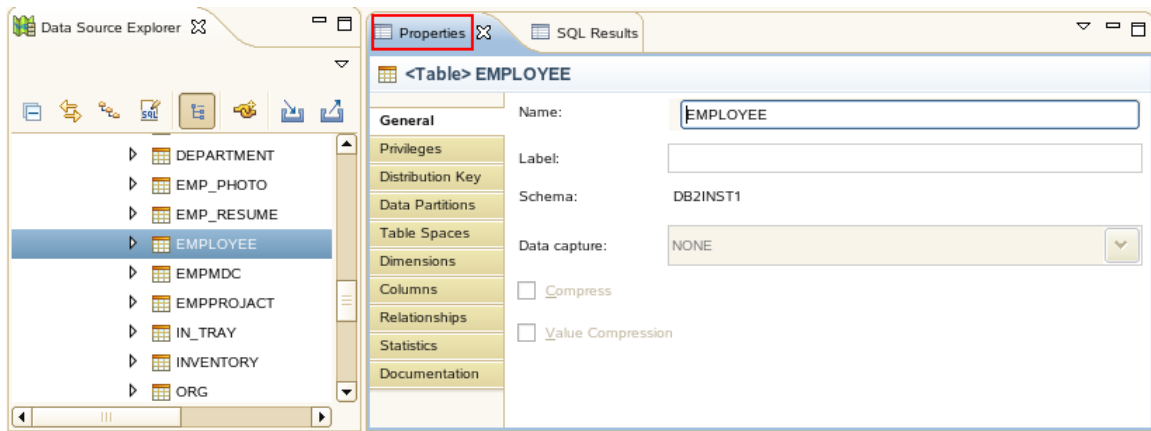
Repeat step 1 and notice that the previous **Schemas** node now reads **Schemas [Filtered]**. Note as well that **DB2INST1** is the only schema that appears in the list.



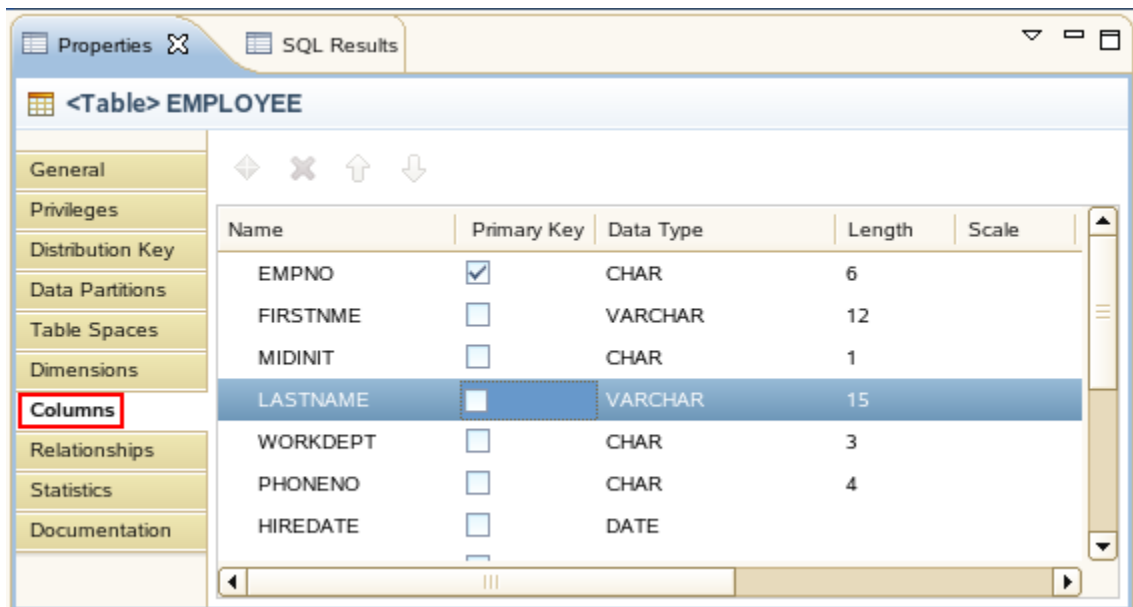
4.2 Viewing Data Object Properties

Data Source Explorer allows users to view most properties of nearly every single data object available such as tables, views, packages, sequences, and so on. Let's spend a moment viewing the properties of some table to see how the Properties view works.

1. In the **Data Source Explorer**, go to: **SAMPLE > Schemas [Filtered] > DB2INST1 > Tables**.
2. Select one of the listed tables, say, **EMPLOYEE** by clicking on it.
3. Select the **Properties** tab to the right of the **Data Source Explorer**.



4. Click the **Columns** tab to view a list of the columns in the **EMPLOYEE** table.



5. Data Manipulation Language

We will now look into the **DML** statements, using the **SAMPLE** database created in Section 3.3. In this section, we are going to explore some commonly used

DML statements by walking through examples, and then follow it up with some exercises to test your understanding.

Before we move on to executing the SQL statements, make sure you are connected to the SAMPLE database.

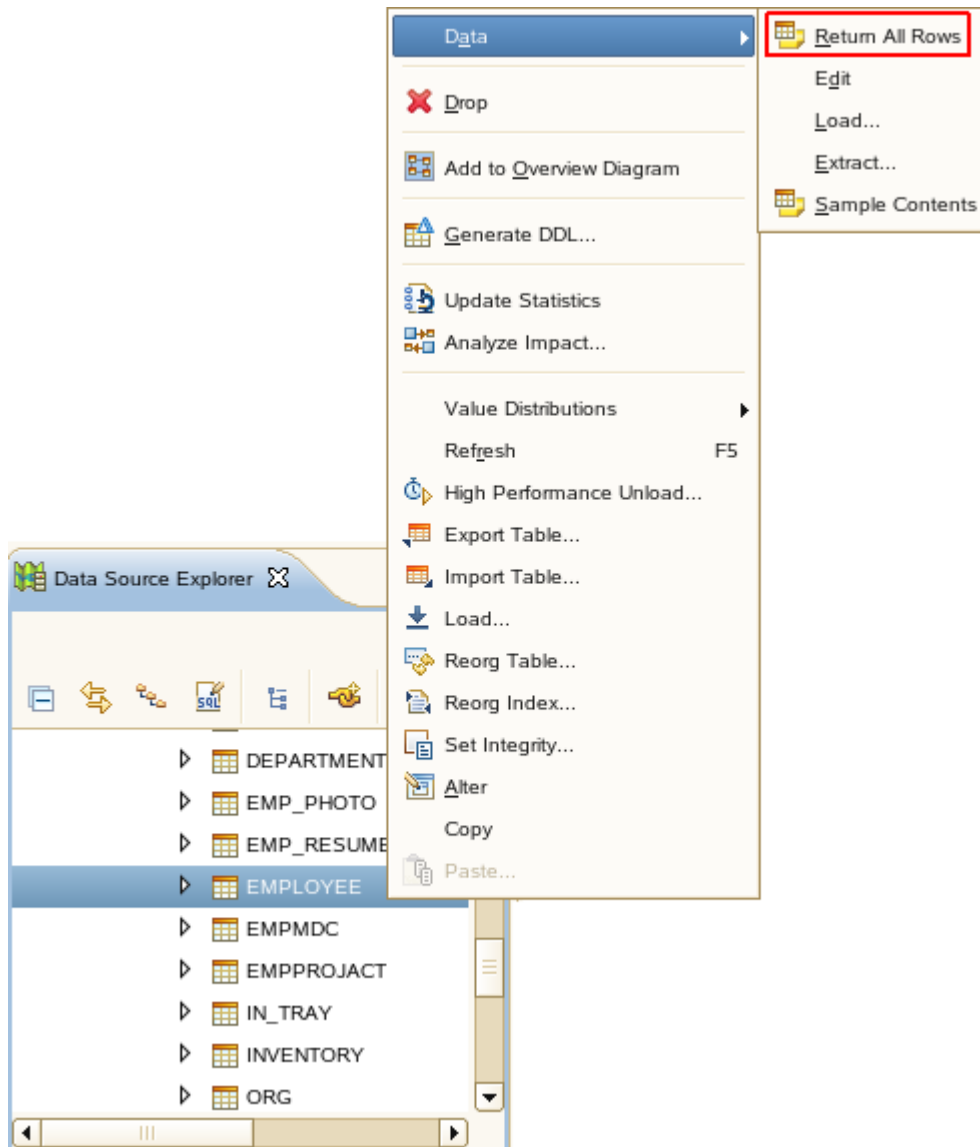
5.1 Querying Data

Because no database is worth much unless data can be obtained from it, it's important to understand how to use a SELECT SQL statement to retrieve data from your tables.

5.1.1 Retrieving All Rows from a Table

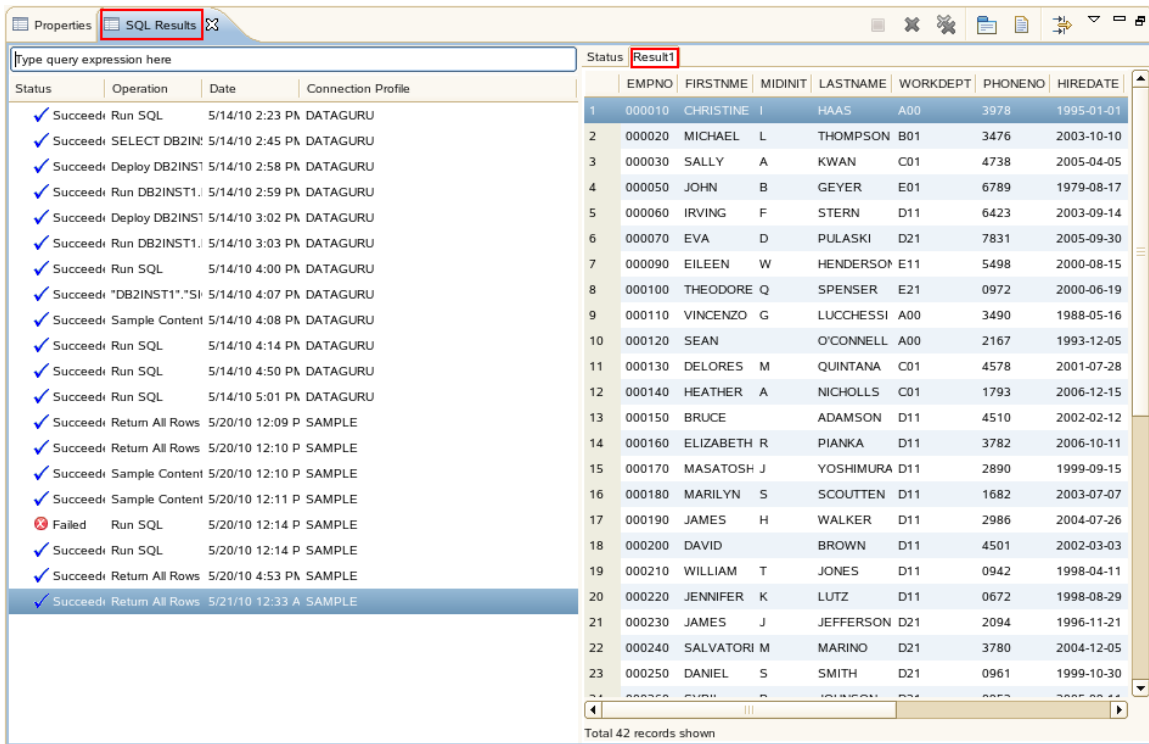
Before using SQL, we'll quickly show you how to retrieve rows from a table just using the Data Studio options, without the need to write SQL code.

1. In the **Data Source Explorer** view, direct to the table you want to return all the rows. For example, **SAMPLE [DB2 for Linux...] > SAMPLE > Schemas [Filtered] > DB2INST1 > Tables > EMPLOYEE**.
2. Right Click on the table **EMPLOYEE**, choose **Data**, and then click on **Return All Rows**.



3. As we can see under the **SQL Results** tab, the operation was successful. All rows from table **EMPLOYEE** are displayed under the **Result1** tab to the right.

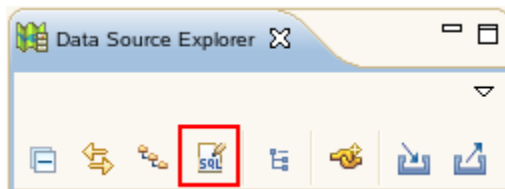
You can always expand or restore views by clicking on the corresponding icons in top right corner of each view.



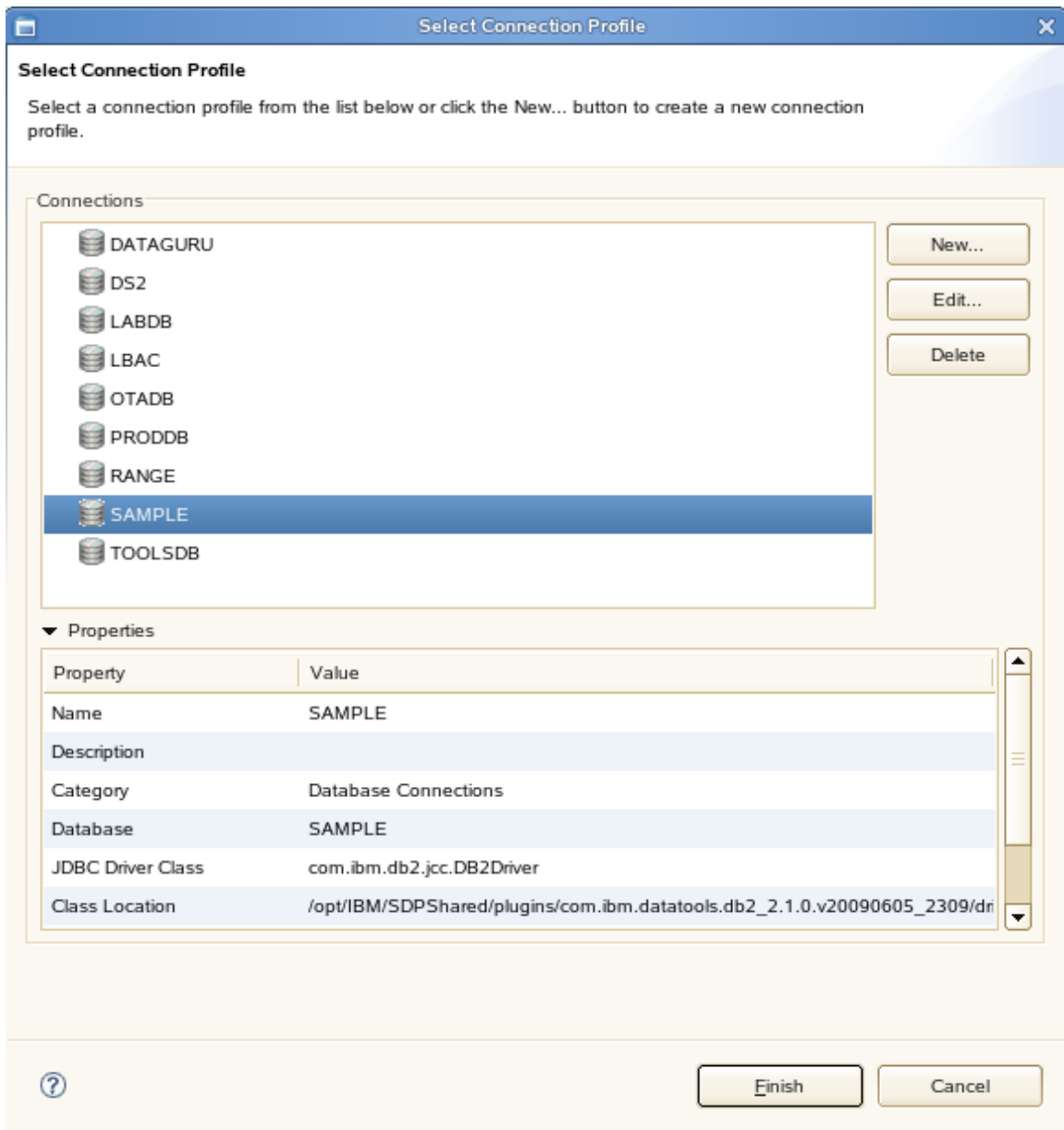
5.1.2 Retrieving Rows using SELECT Statement

Although the method showed in the previous section is handy when you need to quickly inspect the data from a table, in the real world you will be using SQL to retrieve data from a database. Follow the steps below to learn how to execute a SELECT statement in Data Studio.

1. In the **Data Source Explorer** toolbar, click the icon **New SQL Script**.



2. In the **Select Connection Profile** window that appears, select **SAMPLE** and click **Finish**.

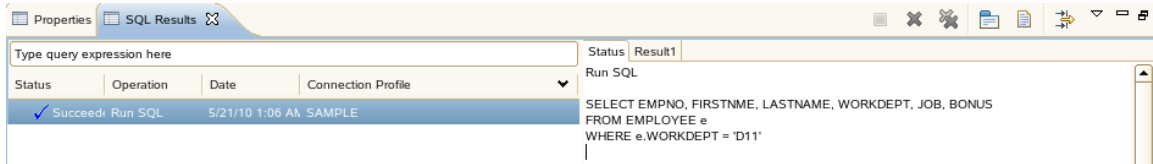


3. A new tab will appear in the main view. Now let's write a SQL query using a **WHERE** clause, say, we are curious about the BONUS money the managers in department D11 will get. Type in the query below in the newly-created tab:

```
SELECT EMPNO, FIRSTNME, LASTNAME, WORKDEPT, JOB, BONUS
FROM EMPLOYEE e
WHERE e. WORKDEPT = 'D11'
```

4. From the main menu, select **Run > Run SQL**

5. Notice that the **SQL Results** view is brought to the foreground at the bottom of the screen. Click the icon to maximize the view. The SQL Results view should indicate that the SQL Script was successful. In the **Status** tab to the right, a summary of the statements in the script file are listed.



6. To view the results of our SQL query statements, click the **Result1** tab on the right.

	EMPNO	FIRSTNME	LASTNAME	WORKDEPT	JOB	BONUS
1	000060	IRVING	STERN	D11	MANAGER	500.00
2	000150	BRUCE	ADAMSON	D11	DESIGNER	500.00
3	000160	ELIZABETH	PIANKA	D11	DESIGNER	400.00
4	000170	MASATOSHI	YOSHIMURA	D11	DESIGNER	500.00
5	000180	MARILYN	SCOUTTEN	D11	DESIGNER	500.00
6	000190	JAMES	WALKER	D11	DESIGNER	400.00
7	000200	DAVID	BROWN	D11	DESIGNER	600.00
8	000210	WILLIAM	JONES	D11	DESIGNER	400.00
9	000220	JENNIFER	LUTZ	D11	DESIGNER	600.00
10	200170	KIYOSHI	YAMAMOTO	D11	DESIGNER	500.00
11	200220	REBA	JOHN	D11	DESIGNER	600.00

7. Restore the **SQL Results** view to its original state, and close the **Script.sql** tab in the main view by clicking its **X** icon.

8. Now let's do some exercises. Create the SQL statements for the queries described below. You can then compare your answers with the suggested solutions in **Section 6**.

1. Find out all sales information from salesman called "**LEE**" in the "Ontario-South" region from the table **SALES**.

2. Find out the name of all the departments that have a manager assigned to it from table **DEPARTMENT**.

Tip: departments without a manager have NULL in the column **MGRNO**.

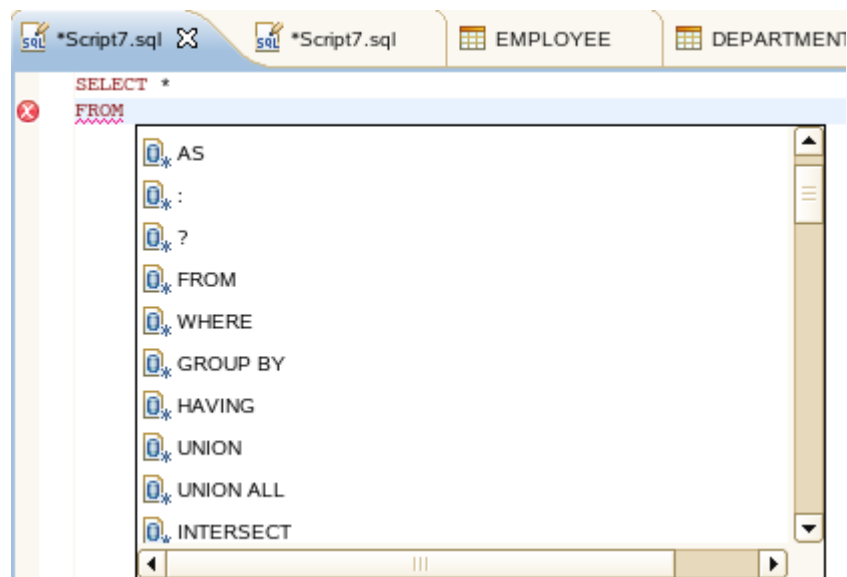
5.1.3 Sorting the Results

The **ORDER BY** statement sorts the result set by one or more columns. By default, the records returned by executing a SQL statement are sorted in ascending order, but we can change it to a descending order with the **DESC** keyword.

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

Let's now run an example on our **SAMPLE** database. In the table **STAFF**, rank all the people from department 66 by their salary, in descending order.

Tip: You can invoke the code assist function by pressing **Ctrl + Space**. This way instead of typing in the whole words, you can choose from a pre-defined list of keywords.



Now run the query below:

```
SELECT *
```

```

FROM STAFF
WHERE DEPT = '66'
ORDER BY SALARY DESC

```

Status	Result1	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
1		270	Lea	66	Mgr	9	88555.50	NULL
2		280	Wilson	66	Sales	9	78674.50	811.50
3		320	Gonzales	66	Sales	4	76858.20	844.00
4		310	Graham	66	Sales	13	71000.00	200.30
5		330	Burke	66	Clerk	1	49988.00	55.50

As you can see from the returned table above, manager “Lea” has the highest salary in department 66.

More exercises: (Suggested solutions in **Section 6**)

1. Using the same table **STAFF** as illustrated above, rank all the people by their years of experience in descending order. For people with same **YEARS**, rank again by their salary in ascending order.

5.1.4 Aggregating Information

The SQL **GROUP BY** statement is used in conjunction with the aggregate functions (e.g. **SUM**, **COUNT**, **MIN**, or **MAX**, etc.) to group the result-set by one or more columns.

```

SELECT column_name(s), aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name

```

Consider that you need to find out the average salary for each job position from table **STAFF**. Run the query below:

```

SELECT JOB, AVG(SALARY) as AVG_SALARY
FROM STAFF
GROUP BY JOB

```


- **RIGHT (OUTER) JOIN:** Returns all rows that have corresponding PKs and FKs plus the rows from right table that don't have any matches in the left table.
- **FULL (OUTER) JOIN:** Returns all rows that have corresponding PKs and FKs plus the rows from left table that don't have any matches in the right table, plus the rows from right table that don't have any matches in the left table.

Now, from tables **EMP_PHOTO** and **EMPLOYEE**, let's find out who uploaded a employee photo in bitmap format. Try the query below:

```
SELECT e.EMPNO, p.PHOTO_FORMAT, e.FIRSTNME, e.LASTNAME
FROM EMPLOYEE e, EMP_PHOTO p
WHERE e.EMPNO = p.EMPNO AND p.PHOTO_FORMAT = 'bitmap'
```

Note: *p.EMPNO* in **EMP_PHOTO** is actually **FK** referring to *e.EMPNO*, the **PK** in table **EMPLOYEE**.

Status	Result1	EMPNO	PHOTO_FORMAT	FIRSTNME	LASTNAME
1		000130	bitmap	DELORES	QUINTANA
2		000140	bitmap	HEATHER	NICHOLLS
3		000150	bitmap	BRUCE	ADAMSON
4		000190	bitmap	JAMES	WALKER

The rationale behind the SQL above is as follows: First, the tables in the **FROM** clause are combined together into a bigger one. The **WHERE** clause is then responsible for filtering rows from this bigger table. Finally the columns in the **SELECT** clause are returned for all matching rows. This is known as "implicit join notation" for **INNER JOIN**. The equivalent query with "explicit join notation" is shown below:

```
SELECT e.EMPNO, p.PHOTO_FORMAT, e.FIRSTNME, e.LASTNAME
FROM EMPLOYEE e INNER JOIN EMP_PHOTO p
ON e.EMPNO = p.EMPNO
AND p.PHOTO_FORMAT = 'bitmap'
```

Now let's run a similar query, but with **LEFT OUTER JOIN** instead of **INNER JOIN** above. Run the following query:

```
SELECT e.EMPNO, p.PHOTO_FORMAT, e.FIRSTNAME, e.LASTNAME
FROM   EMPLOYEE e LEFT OUTER JOIN EMP_PHOTO p
      ON e.EMPNO = p.EMPNO
      AND p.PHOTO_FORMAT = 'bitmap'
```

This time, the result is much longer. From what was explained before, an outer join does not require each record in the two joined tables to have a matching record. And the result of a left (outer) join for table **EMPLOYEE** and **EMP_PHOTO** always contains all records of the "left" table (**EMPLOYEE**), even if the join-condition does not find any matching record in the "right" table (**EMP_PHOTO**), and this is where all the *NULL* values under column *PHOTO_FORMAT* come from.

Status	Result1			
	EMPNO	PHOTO_FORMAT	FIRSTNME	LASTNAME
1	000130	bitmap	DELORES	QUINTANA
2	000140	bitmap	HEATHER	NICHOLLS
3	000150	bitmap	BRUCE	ADAMSON
4	000190	bitmap	JAMES	WALKER
5	000110	NULL	VINCENZO	LUCCHESSI
6	000180	NULL	MARILYN	SCOUTTEN
7	200010	NULL	DIAN	HEMMINGER
8	200330	NULL	HELENA	WONG
9	000200	NULL	DAVID	BROWN
10	000280	NULL	ETHEL	SCHNEIDER
11	000310	NULL	MAUDE	SETRIGHT
12	300001	NULL	Paul	Pierce
13	000070	NULL	EVA	PULASKI
14	000170	NULL	MASATOSHI	YOSHIMURA
15	000240	NULL	SALVATORE	MARINO
16	000260	NULL	SYBIL	JOHNSON
17	000270	NULL	MARIA	PEREZ
18	200280	NULL	EILEEN	SCHWARTZ
19	000100	NULL	THEODORE	SPENSER
20	000230	NULL	JAMES	JEFFERSON
21	000250	NULL	DANIEL	SMITH
22	000320	NULL	RAMLAL	MEHTA
23	200240	NULL	ROBERT	MONTEVERDE
24	000020	NULL	MICHAEL	THOMPSON
25	000050	NULL	JOHN	GEYER
26	000210	NULL	WILLIAM	JONES
27	000290	NULL	JOHN	PARKER
28	200120	NULL	GREG	ORLANDO
29	200310	NULL	MICHELLE	SPRINGER
30	000010	NULL	CHRISTINE	HAAS
31	000060	NULL	IRVING	STERN
32	000220	NULL	JENNIFER	LUTZ
33	000330	NULL	WING	LEE
34	200220	NULL	REBA	JOHN
35	200340	NULL	ROY	ALONZO
36	000030	NULL	SALLY	KWAN
37	000120	NULL	SEAN	O'CONNELL
38	000300	NULL	PHILIP	SMITH
39	000090	NULL	EILEEN	HENDERSON
40	000160	NULL	ELIZABETH	PIANKA
41	000340	NULL	JASON	GOUNOT
42	200170	NULL	KIYOSHI	YAMAMOTO
43	200140	NULL	KIM	NATZ

Exercises time! (Suggested solutions in **Section 6**)

1. Consider you are interested in the action information (with *ACTNO* greater than 100) information and designer names of each project action (from table **PROJACT**). List this information, sorting the results alphabetically according to designers' names.
2. Join tables **EMPLOYEE** and **DEPARTMENT**, considering *WORKDEPT* in **EMPLOYEE** is the **FK** referring to *DEPTNO* the **PK** in table **DEPARTMENT**. Save the results, and repeat this query, but use **LEFT OUTER JOIN**, **RIGHT OUTER JOIN** and **FULL OUTER JOIN** instead. Compare the results.

5.2 Insert, Update and Delete

Consider now that we are required to enter new product information into our database; or that Ms. Lee gets promoted so her *JOB* and *SALARY* need to be altered accordingly; or Mr. Bryant was not active enough over a certain project and got fired, should we still have him in our employee table?

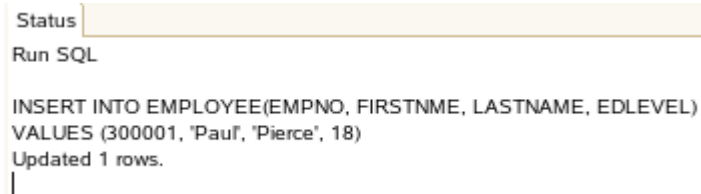
For these situations, we can use the SQL statements **INSERT**, **UPDATE** and **DELETE** to manipulate the table data.

5.2.1 INSERT

INSERT statement is used to add a new row to a table. For example, NBA player Paul Pierce has retired from his career of basketball player, and successfully locates himself in a position at your company. Now you should add his information to the **EMPLOYEE** table.

Run the query below:

```
INSERT INTO EMPLOYEE(EMPNO, FIRSTNME, LASTNAME, EDLEVEL)
VALUES (300001, 'Paul', 'Pierce', 18)
```



Status

Run SQL

```
INSERT INTO EMPLOYEE(EMPNO, FIRSTNME, LASTNAME, EDLEVEL)
VALUES (300001, 'Paul', 'Pierce', 18)
Updated 1 rows.
```

If we run `SELECT * FROM EMPLOYEE`, we can see that Paul Pierce is now successfully part of the **EMPLOYEE** table, with unspecified columns filled with the default value, which is *NULL* in this case.

	FIRSTNAME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM	
1	300001	Paul	NULL	Pierce	NULL	NULL	NULL	18	NULL	NULL	NULL	NULL	NULL	
2	200340	ROY	R	ALONZO	E21	5698	1997-07-05	FIELI	16	M	1956-05-17	31840.00	500.00	1907.00
3	200330	HELENA		WONG	E21	2103	2006-02-23	FIELI	14	F	1971-07-18	35370.00	500.00	2030.00
4	200310	MICHELLE	F	SPRINGER	E11	3332	1994-09-12	OPEFI	12	F	1961-04-21	35900.00	300.00	1272.00
5	200280	EILEEN	R	SCHWARTZ	E11	8997	1997-03-24	OPEFI	17	F	1966-03-28	46250.00	500.00	2100.00
6	200240	ROBERT	M	MONTEVERDI	D21	3780	2004-12-05	CLEFI	17	M	1984-03-31	37760.00	600.00	2301.00
7	200220	REBA	K	JOHN	D11	0672	2005-08-29	DESI	18	F	1978-03-19	69840.00	600.00	2387.00
8	200170	KIYOSHI		YAMAMOTO	D11	2890	2005-09-15	DESI	16	M	1981-01-05	64680.00	500.00	1974.00
9	200140	KIM	N	NATZ	C01	1793	2006-12-15	ANAL	18	F	1976-01-19	68420.00	600.00	2274.00
10	200120	GREG		ORLANDO	A00	2167	2002-05-05	CLEFI	14	M	1972-10-18	39250.00	600.00	2340.00
11	200010	DIAN	J	HEMMINGEF	A00	3978	1995-01-01	SALE	18	F	1973-08-14	46500.00	1000.00	4220.00
12	000340	JASON	R	GOUNOT	E21	5698	1977-05-05	FIELI	16	M	1956-05-17	43840.00	500.00	1907.00

The **INSERT** statement could also have sub-queries, for example, using a **SELECT** clause, which would allow us to insert multiple records at once. Let's try it out. First, execute the DDL below:

```
CREATE TABLE MNG_PEOPLE LIKE EMPLOYEE
```

It creates a new table called **MNG_PEOPLE** which inherits all the properties/column definitions from table **EMPLOYEE**.

Then we **SELECT** all the managers from table **EMPLOYEE** and insert them into the newly-created table **MNG_PEOPLE**.

```
INSERT INTO MNG_PEOPLE SELECT * FROM EMPLOYEE WHERE JOB = 'MANAGER'
```

To check if the operation was successful, retrieve all rows from table **MNG_PEOPLE**. You should see that 7 records were successfully inserted into the new table.

Status	Result1	EMPNO	FIRSTNAME	MI	LASTNAME	WORK	PHONE	HIRE	JOB	EDL	SEX	BIRTHDATE	SALARY	BONUS	COMM
1		000020	MICHAEL	L	THOMPSON	B01	3476	2003-	MANAGER	18	M	1978-02-02	94250.00	800.00	3300.00
2		000030	SALLY	A	KWAN	C01	4738	2005-	MANAGER	20	F	1971-05-11	98250.00	800.00	3060.00
3		000050	JOHN	B	GEYER	E01	6789	1979-	MANAGER	16	M	1955-09-15	80175.00	800.00	3214.00
4		000060	IRVING	F	STERN	D11	6423	2003-	MANAGER	16	M	1975-07-07	72250.00	500.00	2580.00
5		000070	EVA	D	PULASKI	D21	7831	2005-	MANAGER	16	F	2003-05-26	96170.00	700.00	2893.00
6		000090	EILEEN	W	HENDERSON	E11	5498	2000-	MANAGER	16	F	1971-05-15	89750.00	600.00	2380.00
7		000100	THEODORE	Q	SPENSER	E21	0972	2000-	MANAGER	14	M	1980-12-18	86150.00	500.00	2092.00

Now try to execute the exercises below (Suggested solutions can be found in **Section 6**):

1. Our company just started a new department called *FOO* with department number *K47*, and '*E01*' as ADMRDEPT. Please insert this record into table **DEPARTMENT**.
2. Create a new table called **D11_PROJ** with the same structure of table **PROJECT** and add to it data about all projects from department *D11*.

5.2.2 UPDATE

UPDATE statement is used to update existing records in a table. Its syntax looks like:

```
UPDATE table_name
SET column1=value, column2=value2,...,column = valueN
WHERE some_column=some_value
```

Note: The **WHERE** clause here indicates specifically which record(s) will be updated. Without **WHERE**, all records in this table will be modified!

The Human Resources lady just handed you a detailed information list about Paul Pierce, and asked you to update all the following columns in table **EMPLOYEE** with his data:

```
HIREDATE: 2010-01-01
JOB:      DESIGNER
SEX:      M
BIRTHDATE: 1977-10-13
```

We update his personal information by running the query below:

```
UPDATE EMPLOYEE
```

```

SET HIREDATE = '2010-01-01', JOB = 'DESIGNER', SEX = 'M', BIRTHDATE =
'1977-10-13'
WHERE EMPNO = 300001

```

As you can see in the status tab, one row has been updated.

```

Status
Run SQL

UPDATE EMPLOYEE
SET HIREDATE = '2010-01-01', JOB = 'DESIGNER', SEX = 'M', BIRTHDATE = '1977-10-13'
WHERE EMPNO = 300001
Updated 1 rows.

```

Again, if we run `SELECT * FROM EMPLOYEE`, we can see that Paul's data has been updated in those four columns.

EMP	FIRSTNAME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
1	300001	Paul	NULL	Pierce	NULL	2010-01-01	DESIGNER	18	M	1977-10-13	NULL	NULL	NULL

Now, let's see what else we can do with Paul's information.

(Suggested solutions in **Section 6**):

1. Try to update Paul's EDLEVEL to 'NULL', see what happens.
2. Try to update Paul's WORKDEPT to 'Z11', see what happens.

5.2.3 DELETE

DELETE statement deletes rows in a table. Its syntax looks like:

```

DELETE FROM table_name
WHERE some_column=some_value

```

IMPORTANT: Just like the UPDATE statement, if you omit the **WHERE** clause, all records will be deleted.

Now, let's delete Paul Pierce's record from our database, since he changed his mind and headed back to the basketball court. Run the following query:

```

DELETE FROM EMPLOYEE
WHERE EMPNO = 300001

```


Check the contents of table **EMPLOYEE** (by now, you should know at least two ways to do so). If you successfully executed the DELETE statement, Paul's record should not be in the result list.

More exercises (Suggested solutions in **Section 6. Answers**):

1. Try to delete department 'E21' from table **DEPARTMENT**

6. Answers

Section 5.1.2

Query 1

```
SELECT *  
FROM SALES  
WHERE SALES_PERSON = 'LEE'  
AND REGION = 'Ontario-South'
```

Status	Result1	SALES_DATE	SALES_PERSON	REGION	SALES
1		2005-12-31	LEE	Ontario-South	3
2		2006-03-29	LEE	Ontario-South	2
3		2006-03-30	LEE	Ontario-South	7
4		2006-03-31	LEE	Ontario-South	14
5		2006-04-01	LEE	Ontario-South	8

Query 2

```
SELECT DEPTNAME  
FROM DEPARTMENT  
WHERE MGRNO is not NULL
```

Status	Result1
	DEPTNAME
1	SPIFFY COMPUTER SERVICE DIV.
2	PLANNING
3	INFORMATION CENTER
4	MANUFACTURING SYSTEMS
5	ADMINISTRATION SYSTEMS
6	SUPPORT SERVICES
7	OPERATIONS
8	SOFTWARE SUPPORT

Section 5.1.3

Query 1

```
SELECT *  
FROM STAFF  
WHERE YEARS is not NULL  
ORDER BY YEARS DESC, SALARY ASC
```

Status	Result1	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
1		310	Graham	66	Sales	13	71000.00	200.30
2		260	Jones	10	Mgr	12	81234.00	NULL
3		50	Hanes	15	Mgr	10	80659.80	NULL
4		290	Quill	84	Mgr	10	89818.00	NULL
5		210	Lu	10	Mgr	10	90010.00	NULL
6		280	Wilson	66	Sales	9	78674.50	811.50
7		270	Lea	66	Mgr	9	88555.50	NULL
8		190	Sneider	20	Clerk	8	34252.75	126.50
9		20	Pernal	20	Sales	8	78171.25	612.45
10		340	Edwards	84	Sales	7	67844.00	1285.00
11		70	Rothman	15	Sales	7	76502.83	1152.00
12		100	Plotz	42	Mgr	7	78352.80	NULL
13		160	Molinare	10	Mgr	7	82959.20	NULL
14		220	Smith	51	Sales	7	87654.50	992.80
15		10	Sanders	20	Mgr	7	98357.50	NULL
16		90	Koonitz	42	Sales	6	38001.75	1386.70
17		130	Yamaguchi	42	Clerk	6	40505.90	75.60
18		250	Wheeler	51	Clerk	6	74460.00	513.30
19		40	O'Brien	38	Sales	6	78006.00	846.55
20		150	Williams	51	Sales	6	79456.50	637.65
21		140	Fraye	51	Mgr	6	91150.00	NULL
22		110	Ngan	15	Clerk	5	42508.20	206.60
23		350	Gafney	84	Clerk	5	43030.50	188.00
24		300	Davis	84	Sales	5	65454.50	806.10
25		30	Marenghi	38	Mgr	5	77506.75	NULL
26		240	Daniels	10	Mgr	5	79260.25	NULL
27		170	Kermisch	15	Clerk	4	42258.50	110.10
28		320	Gonzales	66	Sales	4	76858.20	844.00
29		180	Abrahams	38	Clerk	3	37009.75	236.50
30		230	Lundquist	51	Clerk	3	83369.80	189.65
31		330	Burke	66	Clerk	1	49988.00	55.50

Total 31 records shown

Section 5.1.4

Query 1

```
SELECT SALES_PERSON, SUM(SALES) AS total_sales
FROM SALES
GROUP BY SALES_PERSON
```

Status	Result1	
	SALES_PERSON	TOTAL_SALES
1	GOUNOT	50
2	LEE	91
3	LUCCHESI	14

Query 2

```
SELECT JOB, COUNT(*) as TOTAL_NUM
FROM EMPLOYEE
WHERE SEX = 'M'
GROUP BY JOB
```

Status	Result1	
	JOB	TOTAL_NUM
1	CLERK	6
2	DESIGNER	6
3	FIELDREP	4
4	MANAGER	4
5	OPERATOR	2
6	SALESREP	1

Section 5.1.5

Query 1

```
SELECT DISTINCT p.PROJNO, FIRSTNME, LASTNAME, p.ACSTDATE, ep.EMSTDATE
FROM EMPLOYEE e, EMPPROJECT ep, PROJECT p
WHERE e.EMPNO = ep.EMPNO
```

```

AND ep.PROJNO = p.PROJNO
AND e.JOB = 'DESIGNER'
AND p.ACTNO > 100
ORDER BY FIRSTNME, LASTNAME

```

Status	Result1	PROJNO	FIRSTNME	LASTNAME	ACSTDATE	EMSTDATE
1		MA2112	BRUCE	ADAMSON	2002-07-15	2002-01-01
2		MA2112	BRUCE	ADAMSON	2002-07-15	2002-07-15
3		MA2113	ELIZABETH	PIANKA	2002-10-01	2002-07-15
4		MA2112	JAMES	WALKER	2002-07-15	2002-01-01
5		MA2112	JAMES	WALKER	2002-07-15	2002-10-01
6		MA2113	MARILYN	SCOUTTEN	2002-10-01	2002-04-01
7		MA2112	MASATOSHI	YOSHIMURA	2002-07-15	2002-01-01
8		MA2113	MASATOSHI	YOSHIMURA	2002-10-01	2002-01-01
9		MA2112	MASATOSHI	YOSHIMURA	2002-07-15	2002-06-01
10		MA2113	WILLIAM	JONES	2002-10-01	2002-10-01

Query 2

```

SELECT *
FROM EMPLOYEE e
      INNER
      | LEFT (OUTER)
      | RIGHT (OUTER)
      | FULL (OUTER)
JOIN DEPARTMENT d
ON e.WORKDEPT = d.DEPTNO

```

Section 5.2.1

Query 1

```

INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
VALUES ('K47', 'FOO', 'E01')

```

Status		Result1			
	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
1	K47	FOO	NULL	E01	NULL

Query 2

```
CREATE TABLE D11_PROJ LIKE PROJECT
```

```
INSERT INTO D11_PROJ
SELECT *
FROM PROJECT
WHERE DEPTNO = 'D11'
```

Status		Result1						
	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
1	MA2110	W L PROGRAMMING	D11	000060	9.00	2002-01-01	2003-02-01	MA2100
2	MA2111	W L PROGRAM DESIGN	D11	000220	2.00	2002-01-01	1982-12-01	MA2110
3	MA2112	W L ROBOT DESIGN	D11	000150	3.00	2002-01-01	1982-12-01	MA2110
4	MA2113	W L PROD CONT PROGS	D11	000160	3.00	2002-02-15	1982-12-01	MA2110

Section 5.2.2

Query 1

```
UPDATE EMPLOYEE
SET EDLEVEL = NULL
WHERE EMPNO = 300001
```

Under the SQL Result tab, it says running of this query fails, and the Status tab to the right says it is because we were trying to assign a NULL value to a NOT NULL column, which is illegal.

Status	Operation	Date	Connection Profile
Failed	Run SQL	5/26/10 2:11 AM	SAMPLE

```
Status
Run SQL

UPDATE EMPLOYEE
SET EDLEVEL = NULL
WHERE EMPNO = 300001
Assignment of a NULL value to a NOT NULL column "TBSPACEID=2, TABLEID=6, COLNO=8" is not allowed.. SQLC
```

Query 2

```
UPDATE EMPLOYEE
SET WORKDEPT = 'Z11'
WHERE EMPNO = 300001
```

This query failed too, because it tried to update a FOREIGN KEY(FK) column, WORKDEPT in our case, with a value that did not exist in the PRIMARY/PARENT KEY column, column DEPTNO in table DEPARTMENT, that this FK was referring to. This is illegal too.

```
Status |
Run SQL

UPDATE EMPLOYEE
SET WORKDEPT = 'Z11'
WHERE EMPNO = 300001
The insert or update value of the FOREIGN KEY "DB2INST1.EMPLOYEE.REDE" is not equal to any value of the parent key of the parent table.. SQL
```

Section 5.2.3

Query 1

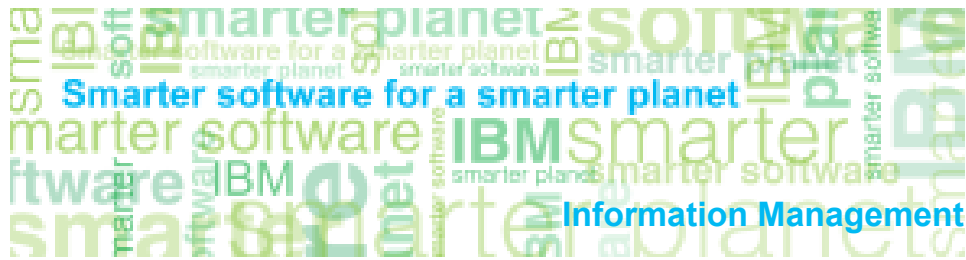
```
DELETE FROM DEPARTMENT
WHERE DEPTNO = 'E21'
```

The error message says we can not delete row(s) containing PRIMARY/PARENT KEY column(s) that some other columns are currently referring to, which is, again, illegal.

```
Status |
Run SQL

DELETE FROM DEPARTMENT
WHERE DEPTNO = 'E21'
A parent row cannot be deleted because the relationship "DB2INST1.PROJECT.FK_PROJECT_1" restricts the deletion..
```

Data Concurrency



© 2010 IBM Corporation



Agenda

- **Database Transactions**
- **Concurrency**
- **Concurrency Issues**
- **Concurrency Control**
 - Isolation Levels
 - Locking
 - Specifying Isolation Levels

Database Transactions

▪ Transaction

- sequence of one or more SQL operations, grouped together as a single unit that occurs in one atomic action
- also known as a unit of work

▪ After a transaction to the database is made it can be made permanent (**committed**) or backed out (**rolled back**)

- manual-commit: use COMMIT or ROLLBACK statements
- auto-commit: database manager performs a commit operation after every SQL statement

▪ Initiation and termination of a transaction defines the point of data consistency of data with the database

- committed data versus uncommitted data

3

© 2010 IBM Corporation

Database Transactions

▪ Committed Data

- data consistent with the database
- changes can always become committed data manually using a COMMIT statement
- committed data can only be reversed/removed with new SQL statements (within a new transaction)
- accessible to all users and applications

▪ Uncommitted Data

- data inconsistent with the database
- changes that occur during the transaction before a COMMIT statement is executed
- changes can be reversed with ROLLBACK
- inaccessible to other users and applications unless Uncommitted Read is used

4

© 2010 IBM Corporation

Database Transactions

Connection to a database defines first initiation

```
CONNECT TO DB employees
INSERT INTO employee VALUES (100, 'JOHN')
INSERT INTO employee VALUES (200, 'MANDY')
COMMIT
```

empID	name
100	JOHN
200	MANDY

```
DELETE FROM employee WHERE name='MANDY'
UPDATE employee SET empID=101 where name='JOHN'
ROLLBACK
```

No changes applied due to ROLLBACK

empID	name
100	JOHN
200	MANDY

```
UPDATE employee SET name='JACK' where empID=100
COMMIT ✗
ROLLBACK
```

empID	name
100	JACK
200	MANDY

5

© 2010 IBM Corporation

Concurrency

- **Concurrency**
 - Sharing of resources by multiple interactive users or application programs at the same time
- **Having multiple interactive users can lead to:**
 - Lost Update
 - Uncommitted Read
 - Non-repeatable Read
 - Phantom Read
- **Need to be able to control the degree of concurrency:**
 - With proper amount of data stability
 - Without loss of performance

6

© 2010 IBM Corporation

Concurrency Issues

▪ Lost Update

- Occurs when two transactions read and then attempt to update the same data, the second update will overwrite the first update before it is committed

- 1) Two applications, A and B, both read the same row and calculate new values for one of the columns based on the data that these applications read
- 2) A updates the row
- 3) Then B also updates the row
- 4) A's update lost

Concurrency Issues

▪ Uncommitted Read

- Occurs when uncommitted data is read during a transaction
- Also known as a Dirty Read

- 1) Application A updates a value
- 2) Application B reads that value before it is committed
- 3) A backs out of that update
- 4) Calculations performed by B are based on the uncommitted data

Concurrency Issues

▪ Non-repeatable Read

- Occurs when a transaction reads the same row of data twice and returns different data values with each read

- 1) Application A reads a row before processing other requests
- 2) Application B modifies or deletes the row and commits the change
- 3) A attempts to read the original row again
- 4) A sees the modified row or discovers that the original row has been deleted

Concurrency Issues

▪ Phantom Read

- Occurs when a search based on some criterion returns additional rows after consecutive searches during a transaction

- 1) Application A executes a query that reads a set of rows based on some search criterion
- 2) Application B inserts new data that would satisfy application A's query
- 3) Application A executes its query again, within the same unit of work, and some additional phantom values are returned

Concurrency Control

- **Isolation Levels**

- determine **how data is locked** or isolated from other concurrently executing processes while the data is being accessed
- are in **effect** while the **transaction is in progress**

- **There are four levels of isolation in DB2:**

- Repeatable read
- Read stability
- Cursor stability
 - **Currently Committed**
- Uncommitted read

Locking

- **Isolation levels are enforced by locks**

- locks **limit or prevent** data **access** by **concurrent users** or application processes

- **Locking Attributes**

- resource** being locked is called **object**
- objects which can be **explicitly** locked are **databases, tables** and **table spaces**
- objects which can be **implicitly** locked are **rows, index keys,** and **tables**
- implicit locks are acquired by DB2 according to isolation level and processing situations
- object** being locked represents **granularity** of lock
- length of time** a lock is held is **called duration** and is affected by isolation level

Types of Locks

- **Share (S)**
 - concurrent transactions are limited to **read-only** operations
- **Update (U)**
 - concurrent transactions are limited to **read-only** operations
 - if the transactions have **not declared** that they might **update** a row, the database manager **assumes** that transaction currently looking at a row **might update** it
- **Exclusive (X)**
 - concurrent transactions are **prevented** from **accessing** the data in any way
 - does **not apply** to transactions with an isolation level of **UR**
- **Database manager places exclusive locks on every row that is inserted, updated, or deleted**

13

© 2010 IBM Corporation

Deadlock

- **Deadlock**
 - Occurs when 2 (or more) competing operations are waiting for each other to free some resource, but neither does, thus the operations will never finish. Eg:

- App1 modifies row 1 on Table A – it holds an X lock on it
- App2 modifies row 5 on Table B – It holds an X lock on it
- App2 tries to modify row 1 on Table A but it can't since App1 has the lock. It goes into WAIT mode.
- App1 tries to modify row 5 on Table B but it can't since App2 has the lock. It goes into WAIT mode.
- **DEADLOCK** as both operations can not complete

- **Deadlock Detector**
 - discovers **deadlock cycles**
 - **randomly** selects one of the transactions involved to roll back and **terminate**
 - transaction chosen is then sent an SQL error code, and every **lock** it had acquired is **released**

14

© 2010 IBM Corporation

Repeatable Read

- **Highest level of isolation**
 - No dirty reads, non-repeatable reads or phantom reads
- **Locks the entire table or view being scanned for a query**
 - Provides **minimum concurrency**
- **When to use Repeatable Read:**
 - Changes** to the result set are **unacceptable**
 - Data stability is more important than performance

Read Stability

- **Similar to Repeatable Read but not as strict**
 - No dirty reads or non-repeatable reads
 - Phantom reads can occur
- **Locks only the retrieved or modified rows in a table or view**
- **When to use Read Stability:**
 - Application needs to operate in a **concurrent environment**
 - Qualifying rows must remain stable for the duration of the unit of work
 - Only** issue **unique queries** during a unit of work
 - If the same query is issued more than once during a unit of work, the same result set should not be required

Cursor Stability

- **Default isolation level**
 - No dirty reads
 - Non-repeatable reads and phantom reads can occur
- **Locks only the row currently referenced by the cursor**
- **When to use Cursor Stability:**
 - Want **maximum concurrency** while seeing only **committed data**


17

© 2010 IBM Corporation

Currently Committed

- **Currently Committed is a variation on Cursor Stability**
 - Avoids timeouts** and **deadlocks**
 - Log based:
 - No management overhead

Situation	Result
Reader blocks Reader	No
Reader blocks Writer	Maybe
Writer blocks Reader	Yes
Writer blocks Writer	Yes



Situation	Result
Reader blocks Reader	No
Reader blocks Writer	No
Writer blocks Reader	No
Writer blocks Writer	Yes

18

© 2010 IBM Corporation

Currently Committed

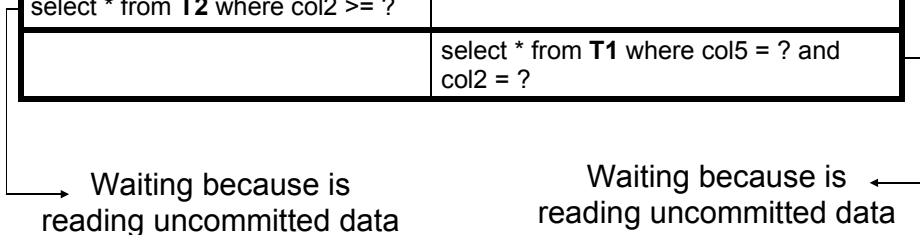
- **Up to DB2 9.5**
 - Cursor Stability is the default isolation level
- **Now in DB2 9.7**
 - Currently Committed is the default for **NEW** databases
 - Currently Committed is disabled for upgraded databases, i.e., Cursor Stability semantics are used
- **Applications that depend on the old behavior (writers blocking readers) will need to update their logic or disable the Currently Committed semantics**

19

© 2010 IBM Corporation

Example – Cursor Stability Semantics

Transaction A	Transaction B
update T1 set col1 = ? where col2 = 2	
	update T2 set col1 = ? where col2 = ?
select * from T2 where col2 >= ?	
	select * from T1 where col5 = ? and col2 = ?

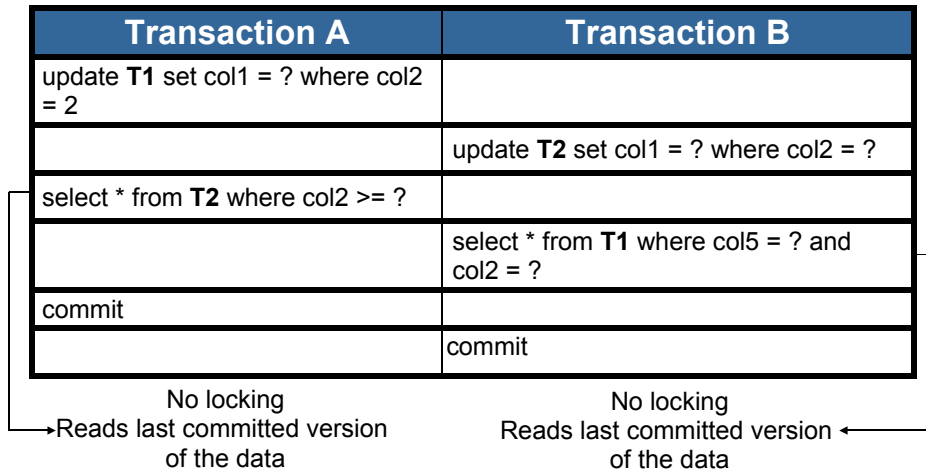


DEADLOCK!! 

20

© 2010 IBM Corporation

Example – Currently Committed Semantics



No deadlocks, no timeouts in this scenario!

21

© 2010 IBM Corporation

Currently Committed – How to use it?

- **cur_commit** – DB config parameter
 - **ON**: default for new DBs created in DB2 9.7 – CC semantics in place
 - **DISABLED**: default value for existing DBs – old CS semantics in place
- **PRECOMPILE/BIND**
 - **CONCURRENTACCESSRESOLUTION**: Specifies the concurrent access resolution to use for statements in the package.
 - USE CURRENTLY COMMITTED
 - WAIT FOR OUTCOME

22

© 2010 IBM Corporation

Uncommitted Read

- **Lowest level of isolation**
 - Dirty reads, non-repeatable reads and phantom reads can occur
- **Locks only rows being modified in a transaction involving DROP or ALTER TABLE**
 - Provides **maximum concurrency**
- **When to use Uncommitted Read:**
 - Querying **read-only** tables
 - Using only **SELECT** statements
 - Retrieving **uncommitted data** is acceptable
- **Uncommitted Read with Read-Write tables**
 - UR behaves like CS with updateable cursors

23

© 2010 IBM Corporation

Isolation Levels

▪ Summary

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
Repeatable Read (RR)	-	-	-
Read Stability (RS)	-	-	Possible
Cursor Stability (CS)	-	Possible	Possible ←
Uncommitted read (UR)	Possible	Possible	Possible

DEFAULT

Application Type	High data stability required	High data stability not required
Read-write transactions	Read Stability (RS)	Cursor Stability (CS)
Read-only transactions	Repeatable Read (RR) or Read Stability (RS)	Uncommitted Read (UR)

24

© 2010 IBM Corporation

Specifying Isolation Levels

▪ Precompile / Bind

- ISOLATION option of PREP or BIND command
- Can determine isolation level of a package by executing the following query

```
SELECT ISOLATION FROM syscat.packages
WHERE pkgname = 'pkgname'
AND pkgschema = 'pkgschema'
```

▪ Statement Level

- Use the WITH {RR, RS, CS, UR} clause
- The WITH UR option applies only to read-only operations
 - ensure that a result table is read-only by specifying FOR READ ONLY in the SQL statement
- Overrides the isolation level specified for the package

```
SELECT * FROM tb1 WITH RR
```

Specifying Isolation Levels

▪ Dynamic SQL within the current session

- SET CURRENT ISOLATION
- For all subsequent dynamic SQL statements within the same session

▪ JDBC or SQLJ at run time

- SQLJ profile customizer (db2sqljcustomize command)

▪ CLI or ODBC at run time

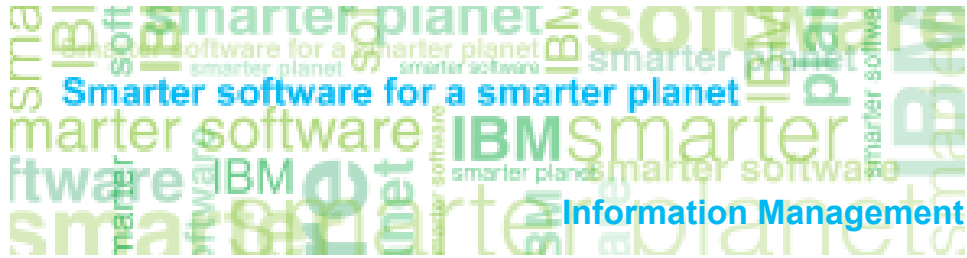
- CHANGE ISOLATION LEVEL command specified during the program preparation process

```
CHANGE ISOLATION LEVEL TO RR
```

Questions?

Summer/Fall 2010

E-mail: imschool@us.ibm.com
Subject: “DB2 Academic Workshop”





IBM DB2[®] 9.7

**Data Concurrency
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

1. INTRODUCTION TO DATA CONCURRENCY	4
1.1 CURSOR STABILITY.....	4
1.2 CURRENTLY COMMITTED	5
1.2.1 Cursor Stability x Currently Committed.....	6
1.2.2 Repeatable Read.....	6
1.2.3 Read Stability.....	7
1.2.4 Uncommitted Read	7
2. OBJECTIVES OF THIS LAB.....	7
3. SETUP AND START DB2	8
3.1 ENVIRONMENT SETUP REQUIREMENTS	8
3.2 LOGIN TO THE VIRTUAL MACHINE.....	8
3.3 SAMPLE DATABASE	9
3.4 CREATE AND POPULATE THE TABLE.....	9
4. CURSOR STABILITY WITH CURRENTLY COMMITTED.....	9
4.1 THE “BEFORE” SCENARIO: WITHOUT CURRENTLY COMMITTED	10
4.1.1 Turning off Currently Committed.....	10
4.1.2 Execute a write query in Terminal A	11
4.1.3 Execute a read query in Terminal B	12
4.1.4 Releasing the lock	13
4.2 THE “AFTER” SCENARIO: WITH CURRENTLY COMMITTED	14
4.2.1 Turning on Currently Committed.....	15
4.2.2 Execute a write query in Terminal A	15
4.2.3 Execute a read query in Terminal B	15
5. REPEATABLE READ.....	17
5.1 THE “PHANTOM READ” SCENARIO: REPEATABLE READ.....	18
5.1.1 Execute a read query in Terminal A	18
5.1.2 Execute a write query in Terminal B	18
5.1.3 Releasing the lock	19
6. READ STABILITY	21
6.1 THE “PHANTOM READ” SCENARIO: READ STABILITY	21
6.1.1 Execute a read query in Terminal A	21
6.1.2 Execute a write query in Terminal B.....	22
6.1.3 Execute another read query in Terminal A	23
7. UNCOMMITTED READ	24
7.1 THE “UNCOMMITTED READ” SCENARIO: CURSOR STABILITY.....	25
7.1.1 Execute an update query in Terminal A.....	25
7.1.2 Execute a read query in Terminal B	25

7.1.3	Releasing the lock	26
7.2	THE "UNCOMMITTED READ" SCENARIO: UNCOMMITTED READ.....	28
7.2.1	Execute an update query in Terminal A.....	28
7.2.2	Execute a read query in Terminal B	28

1. Introduction to Data Concurrency

This section provides a brief introduction about data concurrency and concurrency control in DB2. It explains in detail the available isolation levels in DB2, with a special focus on Currently Committed, which has been introduced in DB2 9.7. *If you are comfortable with these concepts and wish to start the lab exercises right away, please proceed to Section 2.*

In most database environments, many users must access and change the data within the database at the same time. It is important that the database manager allow these multiple users to make concurrent changes while ensuring that data integrity is preserved.

Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager must control this access in order to prevent undesirable effects, such as Lost Updates, Uncommitted Read, Non-repeatable Read and Phantom Read.

The level of concurrency control in a database is determined by the isolation level that is associated with an application process. This determines the degree to which the data that is being accessed by that process, is locked or isolated from other concurrently executing processes. The isolation level is in effect for the duration of a unit of work.

With DB2 there are four levels of isolation available:

- Repeatable read
- Read stability
- Cursor stability (default)
- Uncommitted read

In DB2 9.7 an additional parameter called Currently Committed has been added to the cursor stability (CS) isolation level. Therefore, one could say DB2 has a 5th isolation level: “Cursor Stability with Currently Committed semantics”. In fact, this is the new default for new databases created in DB2 9.7.

1.1 Cursor Stability

The *cursor stability* isolation level locks any row being accessed during a transaction while the cursor is positioned on that row. This lock remains in effect until the next row is fetched or the transaction terminates. However, if any data in the row was changed, the lock is held until the change is committed.

Under this isolation level, no other application can update or delete a row while an updatable cursor is positioned on that row. Under CS, access to the uncommitted data of other applications is not possible. However, non-repeatable reads and phantom reads are possible.

CS is the default isolation level. It is suitable when you want maximum concurrency and need to see only committed data.

1.2 Currently Committed

Lock timeouts and deadlocks can occur under the CS isolation level with row-level locking, especially with applications that are not designed to prevent such problems. Some high throughput database applications cannot tolerate waiting on locks that are held during transaction processing, and some applications cannot tolerate processing uncommitted data, but still require non-blocking behavior for read transactions.

Under the new *currently committed* semantics, only committed data is returned, as was the case previously, but now readers do not wait for writers to release row locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write operation.

Currently committed semantics are turned on by default for new databases. This allows any application to take advantage of the new behavior, and no changes to the application itself are required. The new database configuration parameter **cur_commit** can be used to override this behavior. This might be useful, for example, in the case of applications that require blocking on writers to synchronize internal logic.

Similarly, upgraded databases have **cur_commit** disabled by default in case applications require blocking writers to synchronize their internal logic, and this parameter can be turned on later, if so desired.

Currently committed semantics apply only to read-only scans that do not involve catalog tables or the internal scans that are used to evaluate constraints. Note that, because currently committed is decided at the scan level, a writer's access plan might include currently committed scans. For example, the scan for a read-only subquery can involve currently committed semantics. Because currently committed semantics obey isolation level semantics, applications running under currently committed semantics continue to respect isolation levels.

Currently committed semantics require increased log space. Additional space is required for logging the first update of a data row during a transaction. This data is required for retrieving the currently committed image of the row. Depending on

the workload, this can have an insignificant or substantial impact on the total log space used. The requirement for additional log space does not apply when `cur_commit` is disabled.

1.2.1 Cursor Stability x Currently Committed

Consider the following scenario, in which deadlocks are avoided under the currently committed semantics. In this scenario, two applications update two separate tables, but do not yet commit. Each application then attempts to read (with a read-only cursor) from the table that the other application has updated.

Time	Transaction A	Transaction B
1	<code>update T1 set col1 = ? where col2 = ?</code>	
2		<code>update T2 set col1 = ? where col2 = ?</code>
3		<code>select col1, col5, from T1 where col5 = ? and col2 = ?</code> <i>waiting for A to commit</i>
4	<code>select col1, col3, col4 from T2 where col2 >= ?</code> <i>waiting for B to commit</i>	

Without currently committed semantics, these transactions running under the cursor stability isolation level might create a deadlock, causing one of the transactions to fail. This happens when each transactions needs to read data that is being updated by the other transaction.

Under currently committed semantics, if the query (of either application) happens to require the data currently being updated by the other transaction, that transaction does not wait for the lock to be released, making a deadlock impossible. The previously committed version of the data is located and used instead.

1.2.2 Repeatable Read

Under Repeatable Read, lost updates, uncommitted read, non-repeatable reads, and phantom reads are not possible. In this scenario we will simulate how a phantom read would occur and observe how repeatable read isolates the transactions in order to prevent concurrency problems.

Application A will execute a query that reads a set of rows based on some search criterion. Application B will try to insert new data that would satisfy application A's query.

1.2.3 Read Stability

Read Stability is similar to Repeatable Read, however, since Read Stability only locks the rows being accessed, phantom reads can occur. We will simulate a scenario to show how read stability differs from repeatable read in terms of isolation transactions.

Application A will execute a query that reads a set of rows based on some search criterion. Application B will insert new data that would satisfy application A's query.

1.2.4 Uncommitted Read

The uncommitted read isolation level can be useful when using read-only tables or only select statements. When using uncommitted read we do not have to worry about wait times because uncommitted read does not wait for a transaction to commit. Instead, it will read the uncommitted changes of other transactions. Updatable cursors operating under UR behave as though the isolation level were CS. Under UR, access to uncommitted data, non-repeatable reads, and phantom reads are possible.

Application A will execute a query that updates a row using RR. Application B will attempt to read the same row using CS and UR.

2. Objectives of This Lab

After completion of this lab, the student should be able to:

- Understand the semantic differences between Cursor Stability and Currently Committed.
- Be able to enable and disable the Currently Committed semantics for a database.
- Understand the differences between Repeatable Read, Read Stability, Cursor Stability and Uncommitted Read.
- Be able to specify different isolation levels for a database at run time using CLP.

3. Setup and Start DB2

3.1 Environment Setup Requirements

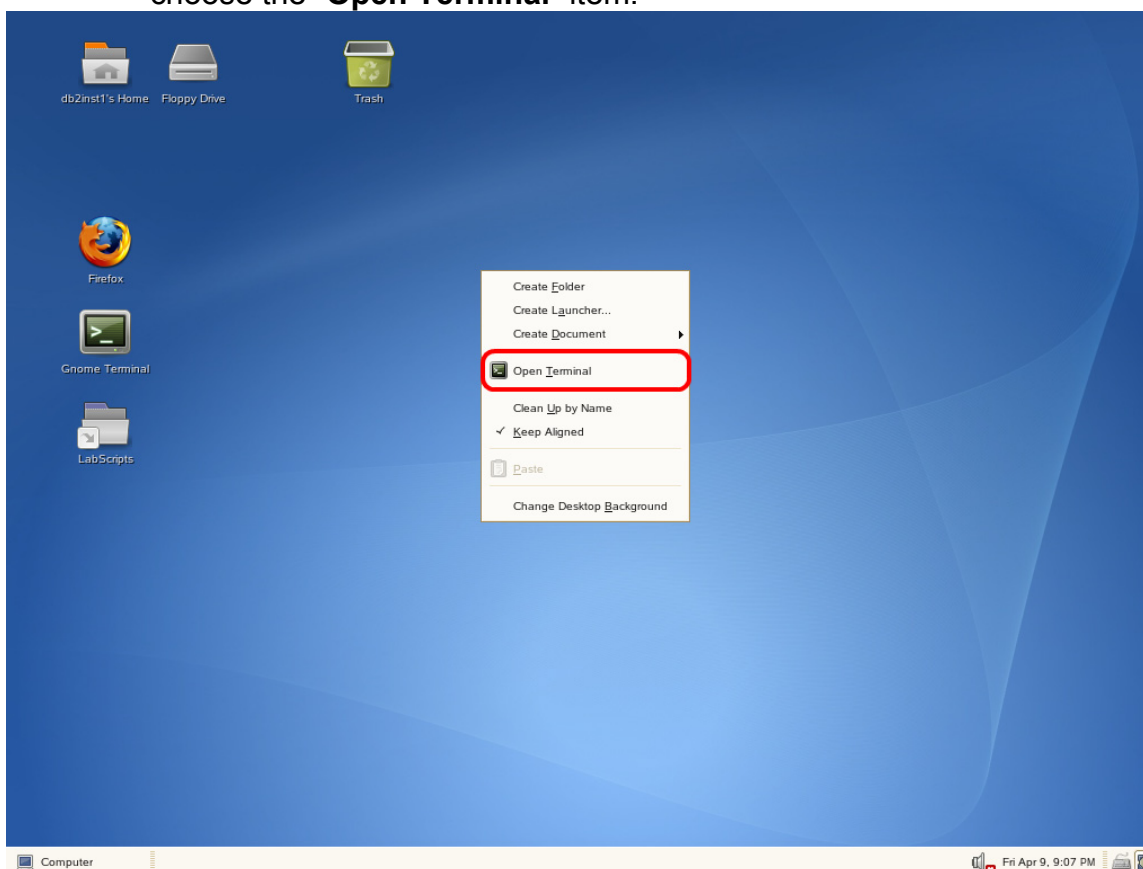
To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in the **VMware Basics and Introduction** module.

3.2 Login to the Virtual Machine

1. Login to the VMware virtual machine using the following information:
User: **db2inst1**
Password: **password**
2. Open a terminal window as by right-clicking on the **Desktop** area and choose the **“Open Terminal”** item.



3. Start up DB2 Server by typing “**db2start**” in the terminal window.

```
db2start
```

3.3 SAMPLE Database

For executing this lab, you will need the DB2’s sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the **SAMPLE** database:

```
db2 force applications all
db2 drop db sample
db2samp1
```

3.4 Create and populate the table

We will create a simple table that will be updated during this lab session. The table named “**tb1**” will be created with a single column named “**column1**”. We will then populate it with 9 rows with the same value “**10**”.

1. Run the following commands.

```
db2 connect to SAMPLE
db2 "create table TB1 (COLUMN1 integer)"
db2 "insert into TB1 (select 10 from syscat.tables fetch first 9 rows
only)"
db2 terminate
```

4. Cursor Stability with Currently Committed

We will now demonstrate the effect of the currently committed feature. To do so, we will simulate a scenario where a potential read / write block can happen when 2 queries are running concurrently. Then, we compare the difference in results and execution time when we toggle the parameter `cur_commit`.

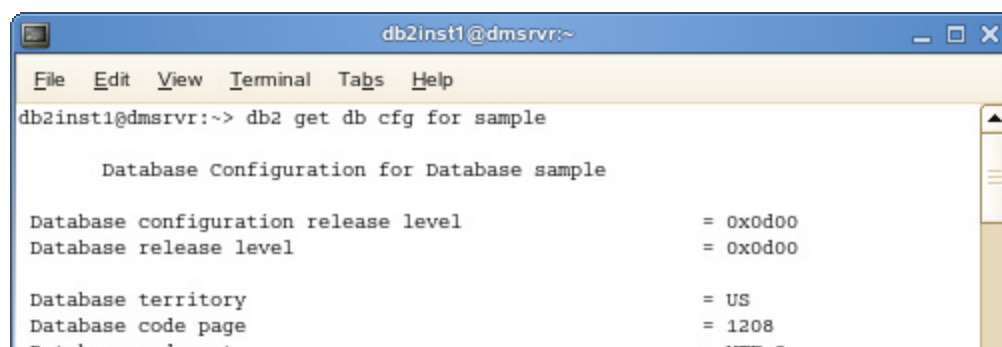
We will use DB2’s command line processor (CLP) to simulate the applications accessing the database at the same time.

4.1 The “Before” scenario: without Currently Committed

4.1.1 Turning off Currently Committed

1. First, we will examine the existing setting for currently committed. Using the terminal, type in the following command. Since we will be using more than one terminal, we'll refer to this terminal as Terminal A.

```
db2 get db cfg for sample
```



```

db2inst1@dmsrvr:~
File Edit View Terminal Tabs Help
db2inst1@dmsrvr:~> db2 get db cfg for sample

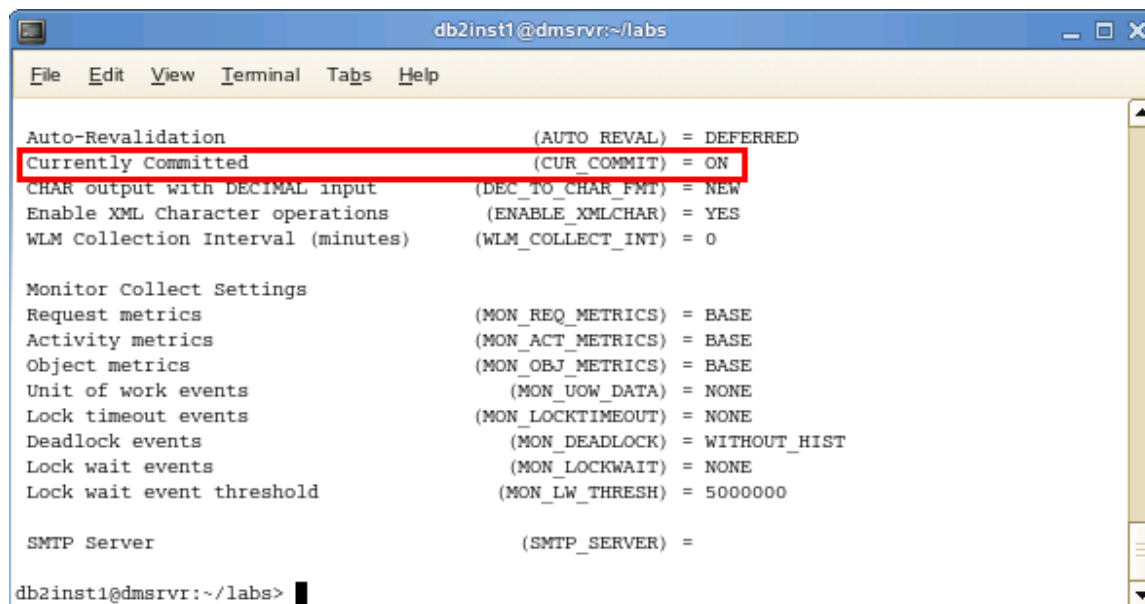
Database Configuration for Database sample

Database configuration release level          = 0x0d00
Database release level                      = 0x0d00

Database territory                          = US
Database code page                          = 1208
Database code set                            - TIME_8

```

The `cur_commit` parameter is located near the end of the list. It should display as ON for now, as this is the default for new databases in DB2 9.7.



```

db2inst1@dmsrvr:~/labs
File Edit View Terminal Tabs Help
Auto-Revalidation (AUTO REVAL) = DEFERRED
Currently Committed (CUR_COMMIT) = ON
CHAR output with DECIMAL input (DEC_TO_CHAR_FMT) = NEW
Enable XML Character operations (ENABLE_XMLCHAR) = YES
WLM Collection Interval (minutes) (WLM_COLLECT_INT) = 0

Monitor Collect Settings
Request metrics (MON_REQ_METRICS) = BASE
Activity metrics (MON_ACT_METRICS) = BASE
Object metrics (MON_OBJ_METRICS) = BASE
Unit of work events (MON_UOW_DATA) = NONE
Lock timeout events (MON_LOCKTIMEOUT) = NONE
Deadlock events (MON_DEADLOCK) = WITHOUT_HIST
Lock wait events (MON_LOCKWAIT) = NONE
Lock wait event threshold (MON_LW_THRESH) = 5000000

SMTP Server (SMTP_SERVER) =

db2inst1@dmsrvr:~/labs>

```

2. The next step is to disable the Currently Committed semantics. For that, change the value of `cur_commit` to DISABLED using the following command:

```
db2 update db cfg for sample using cur_commit disabled
```

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 update db cfg for sample using cur_commit disabled
DB200001 The UPDATE DATABASE CONFIGURATION command completed successfully.
db2inst1@db2rules:~> █

```

4.1.2 Execute a write query in Terminal A

1. In order to mimic the behaviour of a long running transaction, we first need to disable the auto-commit feature, which is ON by default in CLP. When auto-commit is active, CLP automatically issues a COMMIT after every executed SQL statement. Therefore, we need to disable it so we are able to specify when the transaction will be committed. Enter the CLP prompt by typing the command below. The “+c” option will disable the auto-commit feature for this session.

```
db2 +c
```

2. You can check that the auto-commit feature is off by executing the command below. Since auto-commit is OFF, from now on all SQL statements that you execute will be part of the same transaction until you issue a “commit” or “rollback”.

```
list command options
```

```

db2inst1@dmsrvr:~/labs
File Edit View Terminal Tabs Help
-----
Option  Description                               Current Setting
-----
-a      Display SQLCA                               OFF
-c      Auto-Commit                                 OFF
-d      Retrieve and display XML declarations        OFF
-e      Display SQLCODE/SQLSTATE                     OFF
-f      Read from input file                         OFF
-i      Display XML data with indentation           OFF
-l      Log commands in history file                 OFF
-m      Display the number of rows affected          OFF
-n      Remove new line character                    OFF
-o      Display output                              ON
-p      Display interactive input prompt             ON
-q      Preserve whitespaces & linefeeds            OFF
-r      Save output to report file                   OFF
-s      Stop execution on command error              OFF
-t      Set statement termination character          OFF
-v      Echo current command                        OFF
-w      Display FETCH/SELECT warning messages       ON
-x      Suppress printing of column headings        OFF
-z      Save all output to output file               OFF
-----
db2 => █

```

3. Connect to database “sample”.

```
connect to sample
```



```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2 =>

```

- Before we make any updates to the table, we will do a quick query to observe the current values for column "column1".

```
select * from tb1
```

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1

COLUMN1
-----
          10
          10
          10
          10
          10
          10
          10
          10
          10
          10

  9 record(s) selected.

db2 =>

```

- We will then execute an update query which will put a lock on the rows for as long as the transaction is not committed. We will execute a simple update query which will change all the values to 20.

```
update tb1 set column1 = 20
```

```

db2 => update tb1 set column1 = 20
DB20000I  The SQL command completed successfully.
db2 =>

```

4.1.3 Execute a read query in Terminal B

- We will open up another terminal window that will act as the second application trying to access the table. Open a terminal window as by right-clicking on the **Desktop** area and choose the "Open Terminal" item. This new terminal will be designated as Terminal B.

- Similar to the first terminal, we will connect to the database “sample” as user “db2inst1” with password “password” by typing in the command

```
db2 connect to sample
```

- Next, we will launch a query that will read the data locked by Terminal A.

```
time db2 "select * from tb1"
```

The **time** command will allow us to quantify the wait time. We can see that the query waits and does not return any result. In fact, it is being blocked by Terminal A’s query.

The image shows two terminal windows side-by-side. The top window, labeled 'Terminal A', shows a terminal session where a query 'select * from tb1' is executed, returning 9 records with 'COLUMN1' values of 10. Then, an update 'update tb1 set column1 = 20' is executed successfully. The bottom window, labeled 'Terminal B', shows a terminal session where the user attempts to connect to a database named 'sampe' (misspelled), receiving an error: 'SQL1013N The database alias name or database name "SAMPE" could not be found. SQLSTATE=42705'. After correcting the name to 'sample', the connection is successful. Then, the user runs 'time db2 "select * from tb1"', which is blocked by Terminal A's update.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
COLUMN1
-----
          10
          10
          10
          10
          10
          10
          10
          10
          10
          10
          10

  9 record(s) selected.

db2 => update tb1 set column1 = 20
DB20000I The SQL command completed successfully.
db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sampe
SQL1013N The database alias name or database name "SAMPE" could not be found.
SQLSTATE=42705
db2inst1@db2rules:~> db2 connect to sample

  Database Connection Information

Database server      = DB2/LINUX 9.7.1
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

db2inst1@db2rules:~> time db2 "select * from tb1"

```

4.1.4 Releasing the lock

- With the 2 terminals open beside each other, we will observe the effect of committing the query in Terminal A. In Terminal A, commit the transaction by executing the following command

```
commit
```

The image shows two terminal windows. The top window, labeled 'Terminal A', shows a user named 'db2inst1' at 'db2rules' running a series of SQL commands. The first command is 'select * from tb1', which returns 9 records with the value '10'. The second command is 'update tb1 set column1 = 20', which is confirmed as successful. The third command is 'commit', also confirmed as successful. The bottom window, labeled 'Terminal B', shows the same user running 'time db2 "select * from tb1"'. The output shows 9 records with the value '20', indicating that the update from Terminal A has been applied. Below the query results, a timing summary is shown: 'real 2m3.613s', 'user 0m0.008s', and 'sys 0m0.192s'.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
10
10
10
10
10
10
10
10
10
10
9 record(s) selected.
db2 => update tb1 set column1 = 20
DB20000I The SQL command completed successfully.
db2 => commit
DB20000I The SQL command completed successfully.
db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

db2inst1@db2rules:~> time db2 "select * from tb1"

COLUMN1
-----
20
20
20
20
20
20
20
20
20
9 record(s) selected.

real    2m3.613s
user    0m0.008s
sys     0m0.192s
db2inst1@db2rules:~>

```

We can see that terminal B's query instantly returned with the updated values. The block by terminal A has been released and the transaction on terminal B was allowed to continue and access the values.

4.2 The "After" scenario: With Currently Committed

We will repeat the procedure again but this time with the Currently Committed feature turned on. The objective is to see the difference in the time it took for the second query to return and the actual values being returned.

4.2.1 Turning on Currently Committed

1. In Terminal A, we will use the command to turn on currently committed:

```
update db cfg for sample using cur_commit on
```

```
db2 => update db cfg for sample using cur_commit on
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W One or more of the parameters submitted for immediate modification
were not changed dynamically. For these configuration parameters, all
applications must disconnect from this database before the changes become
effective.
db2 =>
```

2. After changing the value, we need to disconnect the database connection for the new value to take effect. In terminal A, execute:

```
connect reset
```

3. In terminal B, execute:

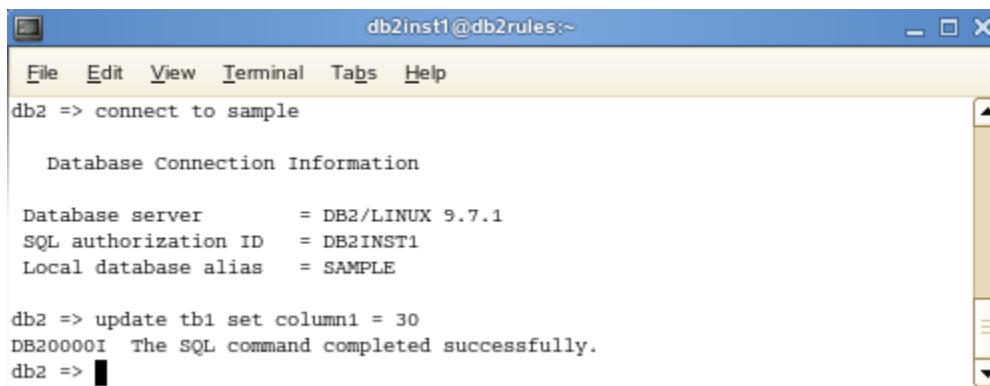
```
db2 connect reset
```

4.2.2 Execute a write query in Terminal A

1. Similar to the previous section, we will update the values in the table from 20 to 30.

```
connect to sample
update tb1 set column1 = 30
```

You should see that the query has been executed successfully.



```
db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2 => update tb1 set column1 = 30
DB20000I The SQL command completed successfully.
db2 =>
```

4.2.3 Execute a read query in Terminal B

1. In Terminal B, reconnect to the database and try to retrieve the values from table tb1.

```
db2 connect to sample
time db2 "select * from tb1"
```

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> time db2 "select * from tb1"

COLUMN1
-----
          20
          20
          20
          20
          20
          20
          20
          20
          20
          20

  9 record(s) selected.

real    0m0.183s
user    0m0.008s
sys     0m0.144s
db2inst1@db2rules:~>

```

Notice the amount of time the query took to return this time. The query returned instantly because there was no access block to the data. Also, notice the values returned were not from the most recent update since we have not committed it yet.

2. In Terminal A, commit the update by typing in the command

```
commit
```

3. Switch the focus back to Terminal B. We want to execute the selection query again by pressing the up arrow button once to retrieve the last executed command, and then press Enter. If you cannot find the last command, type in

```
time db2 "select * from tb1"
```

Notice the values returned this time reflects our last update since the transaction in terminal A has ended and the updates committed to the database.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> time db2 "select * from tb1"

COLUMN1
-----
30
30
30
30
30
30
30
30
30
30

9 record(s) selected.

real    0m0.188s
user    0m0.008s
sys     0m0.144s
db2inst1@db2rules:~>

```

4. Terminate the database connection in terminal A:

```
connect reset
```

5. Then, terminate the database connection in terminal B:

```
db2 connect reset
```

5. Repeatable Read

Now that we have demonstrated the effect of cursor stability and the currently committed feature, we will take a look at repeatable read. To do so, we will simulate a scenario to show how repeatable read isolates each transaction to prevent phantom read concurrency issues.

Application A will execute a query that reads a set of rows based on some search criterion. Application B will try to insert new data that would satisfy application A's query.

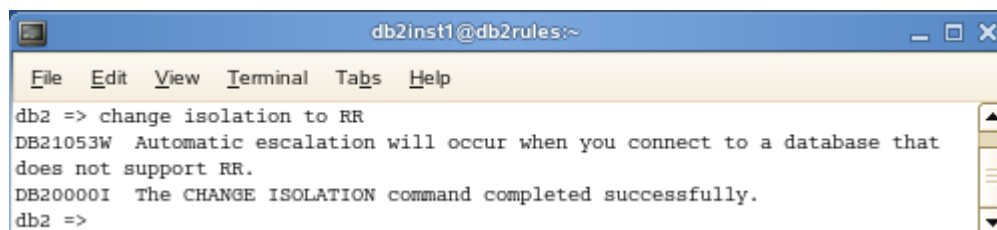
We will use DB2's command line processor (CLP) to simulate the applications accessing the database at the same time.

5.1 The “Phantom Read” scenario: Repeatable Read

5.1.1 Execute a read query in Terminal A

1. We need to change the isolation of the current CLP session of Terminal A to repeatable read. This must be done before connecting to a database.

```
change isolation to RR
```



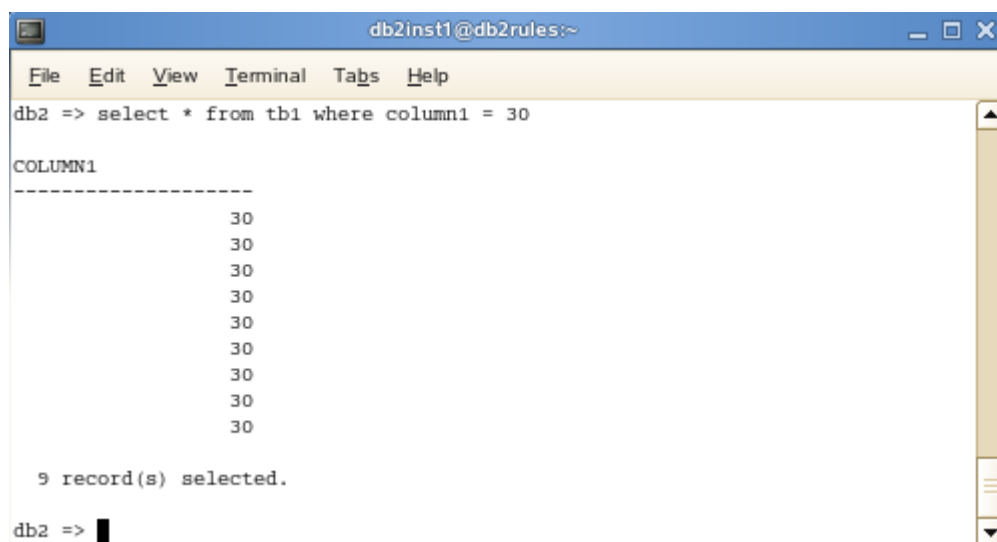
```
db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => change isolation to RR
DB21053W Automatic escalation will occur when you connect to a database that
does not support RR.
DB20000I The CHANGE ISOLATION command completed successfully.
db2 =>
```

2. Connect to database “sample”.

```
connect to sample
```

3. Now we can perform a quick query to observe the current values for column “column1” based on some criteria.

```
select * from tb1 where column1 = 30
```



```
db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1 where column1 = 30

COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30

  9 record(s) selected.
db2 =>
```

5.1.2 Execute a write query in Terminal B

1. We will launch a query that will attempt to insert data into tb1 which is locked by Terminal A.

```
db2 connect to sample
db2 "insert into tb1 values (30)"
```

We can see that the operation waits and does not return any result. In fact, it is being blocked by Terminal A's query.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1 where column1 = 30

COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30

  9 record(s) selected.

db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2inst1@db2rules:~> db2 "insert into tb1 values (30)"

```

5.1.3 Releasing the lock

1. With the 2 terminals open beside each other, we will observe the effect of committing the query in Terminal A. In Terminal A, commit the transaction by executing the following command

```
commit
```



```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30

  9 record(s) selected.

db2 => commit
DB20000I The SQL command completed successfully.
db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sample

  Database Connection Information

Database server      = DB2/LINUX 9.7.1
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

db2inst1@db2rules:~> db2 "insert into tb1 values (30)"
DB20000I The SQL command completed successfully.
db2inst1@db2rules:~> ||

```

We can see that terminal B's query instantly completed. The block by Terminal A has been released and the transaction on Terminal B was allowed to insert the new values.

Here we can see that with the Repeatable Read isolation level, phantom read scenarios do not occur because the rows read by the application are locked and cannot be updated by other transactions.

What if we perform the same scenario with the read stability isolation level instead?

2. Terminate the database connection in terminal A:

```
connect reset
```

3. Then, terminate the database connection in terminal B:

```
db2 connect reset
```

6. Read Stability

We have previously determined that phantom reads cannot occur with the repeatable read isolation level. They are possible, however, when using the read stability isolation level. We will simulate a scenario to show how read stability differs from repeatable read in terms of isolating transactions.

Application A will execute a query that reads a set of rows based on some search criterion. Application B will insert new data that would satisfy application A's query.

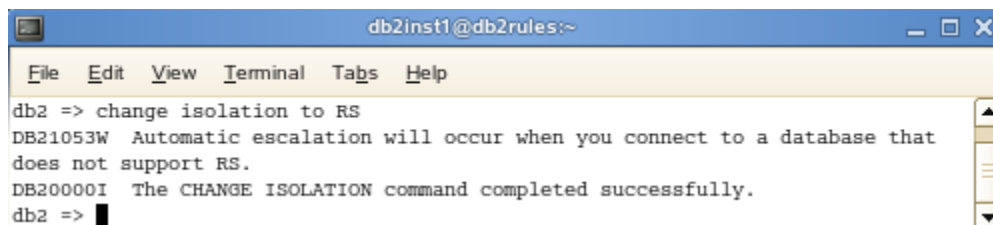
We will use DB2's command line processor (CLP) to simulate the applications accessing the database at the same time.

6.1 The “Phantom Read” scenario: Read Stability

6.1.1 Execute a read query in Terminal A

1. We need to change the isolation of the current CLP session of Terminal A to read stability. This must be done before connecting to a database.

```
change isolation to RS
```



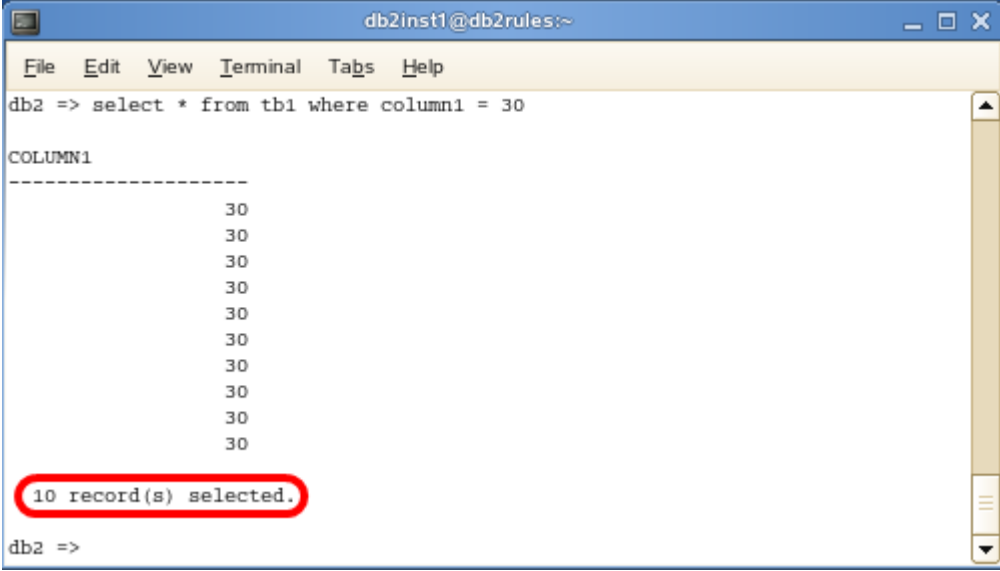
```
db2inst1@db2rules:~  
File Edit View Terminal Tabs Help  
db2 => change isolation to RS  
DB21053W Automatic escalation will occur when you connect to a database that  
does not support RS.  
DB20000I The CHANGE ISOLATION command completed successfully.  
db2 => █
```

2. Connect to database “sample”.

```
connect to sample
```

3. Now we can perform a quick query to observe the current values for column “column1” using some criteria.

```
select * from tb1 where column1 = 30
```

A terminal window titled 'db2inst1@db2rules:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt 'db2 =>' is followed by the SQL query 'select * from tb1 where column1 = 30'. The output shows a table with one column 'COLUMN1' and ten rows of the value '30'. A red circle highlights the text '10 record(s) selected.' at the bottom of the output. The prompt 'db2 =>' is visible at the bottom of the terminal.

```
db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1 where column1 = 30
COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
10 record(s) selected.
db2 =>
```

The number of record(s) selected is currently 10.

6.1.2 Execute a write query in Terminal B

1. Terminal B will insert data matching the criteria of the query by Terminal A.

```
db2 connect to sample
db2 "insert into tb1 values (30)"
```

We can see that the query does not wait for Terminal A to commit and inserts data into tb1.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1 where column1 = 30

COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30

10 record(s) selected.

db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

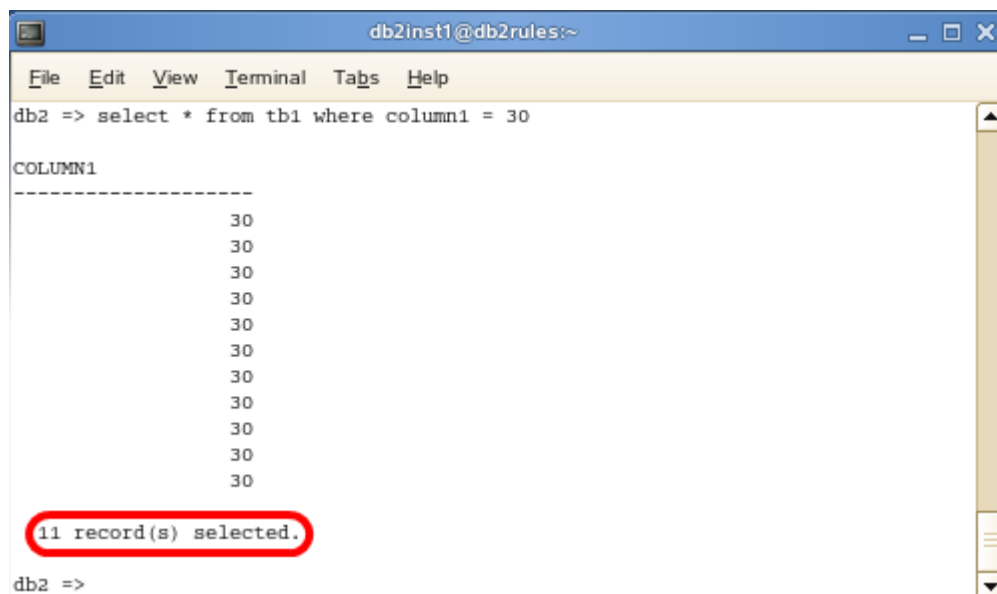
db2inst1@db2rules:~> db2 "insert into tb1 values (30)"
DB20000I The SQL command completed successfully.
db2inst1@db2rules:~>

```

6.1.3 Execute another read query in Terminal A

1. Now we can perform another quick query to observe the current values for column "column1" before committing.

```
select * from tb1 where column1 = 30
```



```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => select * from tb1 where column1 = 30
COLUMN1
-----
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
          30
11 record(s) selected.
db2 =>

```

Notice the query now returns 11 rows of data instead of 10. One additional row has appeared even though we executed the same SQL query inside the same transaction. This is because the Read Stability isolation level does not prevent the appearance of phantom rows.

2. In Terminal A, commit the update by typing in the command

```
commit
```

3. Terminate the database connection in terminal A:

```
connect reset
```

4. Then, terminate the database connection in terminal B:

```
db2 connect reset
```

7. Uncommitted Read

Now that we know what the difference between repeatable read and read stability is, we can see how the lowest isolation level functions. The uncommitted read isolation level can be useful when using read-only tables or only select statements. When using uncommitted read, uncommitted data from other transactions is read.

Application A will execute a query that updates a row using RR. Application B will attempt to read the same row using CS and UR.

7.1 The “Uncommitted Read” scenario: Cursor Stability

7.1.1 Execute an update query in Terminal A

1. We need to change the isolation of the current CLP session of Terminal A to repeatable read. This must be done before connecting to a database.

```
change isolation to RR
```

2. Connect to database “sample”.

```
connect to sample
```

3. Now we can perform a quick query to update the current values for column “column1”.

```
update tb1 set column1 = 40
```



```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => change isolation to RR
DB21053W Automatic escalation will occur when you connect to a database that
does not support RR.
DB20000I The CHANGE ISOLATION command completed successfully.
db2 => connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2 => update tb1 set column1 = 40
DB20000I The SQL command completed successfully.
db2 =>

```

7.1.2 Execute a read query in Terminal B

1. Using CS, Terminal B will attempt to read the data being locked by Terminal A.

```
db2 connect to sample
db2 "select * from tb1"
```

We can see that the select query waits for Terminal A to commit before reading the data.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2 => change isolation to RR
DB21053W Automatic escalation will occur when you connect to a database that
does not support RR.
DB20000I The CHANGE ISOLATION command completed successfully.
db2 => connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2 => update tb1 set column1 = 40
DB20000I The SQL command completed successfully.
db2 =>

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2inst1@db2rules:~> db2 "select * from tb1"

```

7.1.3 Releasing the lock

1. With the 2 terminals open beside each other, we will observe the effect of committing the query in Terminal A. In Terminal A, commit the transaction by executing the following command

```
commit
```

```
db2inst1@db2rules:~  
File Edit View Terminal Tabs Help  
does not support RR.  
DB20000I The CHANGE ISOLATION command completed successfully.  
db2 => connect to sample  
  
Database Connection Information  
  
Database server          = DB2/LINUX 9.7.1  
SQL authorization ID    = DB2INST1  
Local database alias    = SAMPLE  
  
db2 => update tb1 set column1 = 40  
DB20000I The SQL command completed successfully.  
db2 => commit  
DB20000I The SQL command completed successfully.  
db2 => █
```

```
db2inst1@db2rules:~  
File Edit View Terminal Tabs Help  
db2inst1@db2rules:~> db2 "select * from tb1"  
  
COLUMN1  
-----  
40  
40  
40  
40  
40  
40  
40  
40  
40  
40  
40  
40  
  
11 record(s) selected.  
db2inst1@db2rules:~> █
```

We can see that terminal B's query instantly completed. The block by Terminal A has been released and the transaction on Terminal B was allowed to read the committed data.

2. Terminate the database connection in terminal B:

```
db2 connect reset
```


7.2 The “Uncommitted Read” scenario: Uncommitted Read

7.2.1 Execute an update query in Terminal A

1. We will perform a quick query to update the current values for column “column1”.

```
update tb1 set column1 = 50
```

7.2.2 Execute a read query in Terminal B

1. Terminal B will attempt to read the data being locked by Terminal A using UR.

```
db2 change isolation to UR  
db2 connect to sample  
db2 "select * from tb1"
```

We can see that the select query under the uncommitted read isolation level does not wait for Terminal A to commit before reading the data. Instead the values returned are from the uncommitted transaction from Terminal A.

If the transaction from Terminal A executes a rollback, the data listed in Terminal B does not reflect the actual data in TB1. This phenomenon is called a “dirty read”.

```

db2inst1@db2rules:~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 change isolation to UR
DB21053W Automatic escalation will occur when you connect to a database that
does not support UR.
DB20000I The CHANGE ISOLATION command completed successfully.
db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server           = DB2/LINUX 9.7.1
SQL authorization ID     = DB2INST1
Local database alias     = SAMPLE

db2inst1@db2rules:~> db2 "select * from tb1"

COLUMN1
-----
50
50
50
50
50
50
50
50
50
50
50
50
50
50

11 record(s) selected.

```

- In Terminal A, commit the update by typing in the command:

```
commit
```

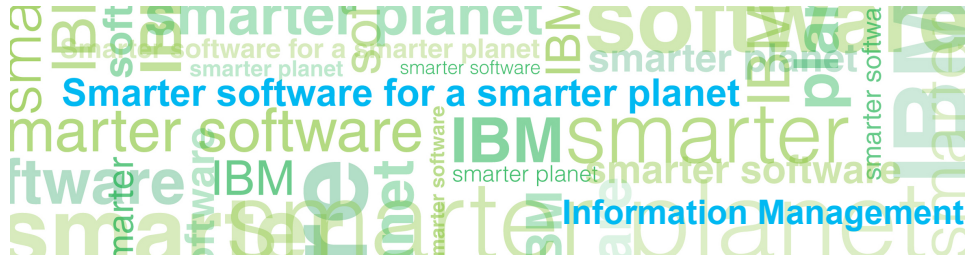
- Terminate the database connection in terminal A:

```
connect reset
```

- Then, terminate the database connection in terminal B:

```
db2 connect reset
```

DB2® Security



© 2010 IBM Corporation



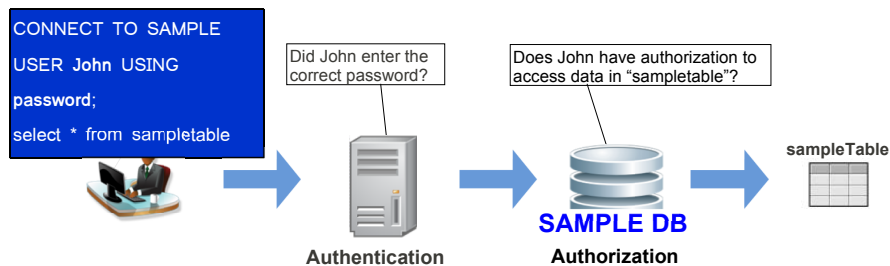
Agenda

- Authentication
- Trusted Context
- Authorization
- Authorities
- Privileges
- Label-Based Access Control (LBAC)
- Roles



DB2 Security Overview

- There are two main mechanisms (and subcategories) within DB2 that allow you to implement a security plan
- **Authentication**
- **Authorization**
 - Authorities
 - Privileges

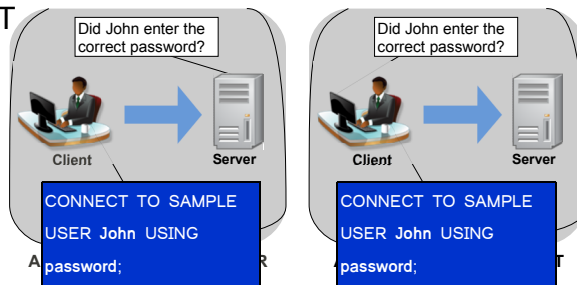
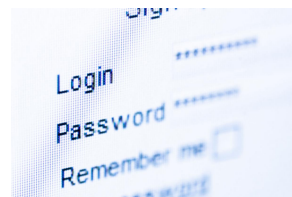


3

© 2010 IBM Corporation

Authentication

- Determining that you are who you say you are
- Can rely on the operating system's authentication mechanism
- Can rely on a separate product
- **Where and how DB2 authenticates users**
 - SERVER
 - SERVER_ENCRYPT
 - CLIENT
 - KERBEROS
 - etc...



4

© 2010 IBM Corporation

Configuration of Authentication on DB2 Server

- Authentication type is defined in the Database Manager configuration file (DBM CFG)

db2 "GET DBM CFG"

```

DB2 CLP - DB2COPY1 - db2
Database manager authentication      <AUTHENTICATION> = SERVER
Alternate authentication            <ALTERNATE_AUTH_ENG> = NOT_SPECIFIED
Cataloging allowed without authority <CATALOG_NOAUTH> = NO
Trust all clients                   <TRUST_ALLCLNTS> = YES
Trusted client authentication       <TRUST_CLNTAUTH> = CLIENT
Bypass federated authentication     <FED_NOAUTH> = NO
    
```

- To configure how and where DB2 authenticates users, set the authentication parameter at the DB2 server

db2 "UPDATE DBM CFG USING AUTHENTICATION CLIENT"

```

DB2 CLP - DB2COPY1 - db2
Database manager authentication      <AUTHENTICATION> = CLIENT
Alternate authentication            <ALTERNATE_AUTH_ENG> = NOT_SPECIFIED
Cataloging allowed without authority <CATALOG_NOAUTH> = NO
Trust all clients                   <TRUST_ALLCLNTS> = YES
Trusted client authentication       <TRUST_CLNTAUTH> = CLIENT
Bypass federated authentication     <FED_NOAUTH> = NO
    
```

5

© 2010 IBM Corporation

Trusted Context

- Provide a means whereby the end-user identity in a three-tier environment can be easily and efficiently propagated to the database server
- Introduce the concept of a trusted context between a database server and a specific application tier
- Why not just keep one common user ID?
 - Loss of user identity for auditing purposes
 - Hard to distinguish actions needed by app vs needed by user
 - Middle tier is “over granted” privileges
 - If ID is compromised, high risk of security exposure

6

© 2010 IBM Corporation

Trusted Context

- **Implementation Considerations**

- Users need to be identified individually but do not want expensive new connections
- How do we identify a trusted source?

- **Solution: Create a “Trusted Context”**

- A trusted relationship between the DB and the application
 - Switch current user ID
 - Acquire additional privileges via role inheritance
- Relationship identified by connection attributes
 - IP Address, Domain Name, Authorization ID, Data Encryption used

```
CREATE TRUSTED CONTEXT ctxt
BASED UPON CONNECTION USING SYSTEM AUTHID smith
ATTRIBUTES (ADDRESS '192.168.2.27')
DEFAULT ROLE managerRole ENABLE
```

7

© 2010 IBM Corporation

Authorization

- **Verifies if an authorization ID has sufficient privileges to perform the desired database operation**

- Authorities
 - Provide a way to group privileges and to control maintenance and utility operations (SYSADM, DBADM, SECADM, SYSMANT, SYSCTRL, ...)
- Privileges
 - Allow a certain action to be taken on a database object (SELECT, UPDATE, DELETE, etc...)
 - LBAC provides a more granular approach, granting read/write access to individual rows/columns

8

© 2010 IBM Corporation

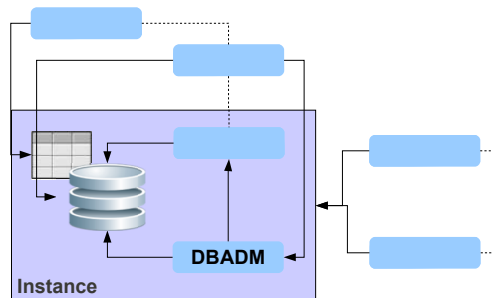
Authorities

- **Instance-level Authorities**

- SYSADM, SYSCTRL, SYSMANT, SYSMON
- Eg: SYSADM - control over all resources created and maintained by the Database Manager (instance)

- **Database-level Authorities**

- DBADM, SECADM, SQLADM, WLMADM, EXPLAIN, ACCESSCTRL, DATAACCESS, etc



9

© 2010 IBM Corporation

System Administrator (SYSADM) Authority

- **Highest level of administrative authority at the instance level**
- **Only a user with SYSADM authority can perform the following functions:**
 - Upgrade and restore a database
 - Change the database manager configuration file including specifying the groups having SYSADM, SYSCTRL, SYSMANT, or SYSMON authority
- **Does not implicit get DBADM authority, so does not automatically have access to data**
- **Specified by the `sysadm_group` parameter in the DBM CFG**
- **Example: Granting SYSADM authority to the group 'grp':**

```
UPDATE DBM CFG USING SYSADM_GROUP grp
```

10

© 2010 IBM Corporation

Database Administrator (DBADM) Authority

- **Administrative authority over a single database**
- **Does not automatically included the ability to access data**
 - Ability to create objects and issue database commands
 - Create, alter, and drop non-security related database objects
 - Read log files
 - Create, activate, and drop event monitors
 - Query the state of a table space
 - Update log history files
 - Quiesce a table space
 - Reorganize a table
 - Collect catalog statistics using the RUNSTATS utility
- **DBADM authority can only be granted or revoked by the SECADM**
- **Can be granted to a user, a group, or a role**

Security Administrator (SECADM) Authority

- **Creates and manages security related database objects over a single database:**
 - Grant and revoke database privileges and authorities
 - Create and drop:
 - Security label components
 - Security policies
 - Security labels
 - Trusted contexts
 - Audit policies
 - Roles
 - Execute audit routines
- **Has no inherent ability to access data stored in user tables**
- **Can only be granted by a user with SECADM authority**

Privileges

- **Schema Privilege**
 - CREATEIN allows the user to create objects within the schema
 - ALTERIN allows the user to alter objects within the schema
 - DROPIN allows the user to drop objects from within the schema
- **Tablespace Privilege**
 - USE allows the user to create tables within the tablespace
- **Table and View Privilege**
 - CONTROL provides the user with all privileges for a table or view including the ability to drop it, and to grant and revoke individual table privileges
 - DELETE allows the user to delete rows from a table or view.
 - INSERT allows the user to insert a row into a table or view, and to run the IMPORT utility.
 - SELECT allows the user to retrieve rows from a table or view, to create a view on a table, and to run the EXPORT utility.
 - UPDATE allows the user to change an entry in a table, a view, or for one or more specific columns in a table or view
 - Table Only Privileges
 - ALTER allows the user to modify on a table
 - INDEX allows the user to create an index on a table
 - REFERENCES allows the user to create and drop a foreign key, specifying the table as the parent in a relationship

13

© 2010 IBM Corporation

Privileges

- **Package Privilege**
 - CONTROL provides the user with the ability to rebind, drop, or execute a package
 - BIND allows the user to rebind or bind that package and to add new package versions of the same package name and creator
 - EXECUTE allows the user to execute or run a package
- **Index Privileges**
 - CONTROL allows the user to drop the index
- **Sequence Privilege**
 - USAGE allows the user to use NEXT VALUE and PREVIOUS VALUE expressions for the sequence
 - ALTER allows the user to perform tasks such as restarting the sequence or changing the increment for future sequence values
- **Routine Privilege**
 - EXECUTE allows the users to invoke a routine, create a function that is sourced from that routine, and reference the routine in any DDL statement such as CREATE VIEW or CREATE TRIGGER

14

© 2010 IBM Corporation

Granting Privileges

- **Explicit**

- Privileges can be explicitly given to users or groups via the GRANT and REVOKE commands

```
db2 grant select on table db2inst1.person to user employee
```

- **Implicit**

- DB2 may grant privileges automatically when certain commands are issued

```
db2 create table mytable
```

User automatically gains full access to the table

- **Indirect**

- Packages contain SQL statements in an executable format. The user only requires EXECUTE privilege to run them
- Example: package1 contains the following static SQL statements

```
select * from test  
insert into test values (1,2,3)
```

- In this case a user with EXECUTE privilege on package1 is indirectly granted SELECT and INSERT privilege on table TEST

15

© 2010 IBM Corporation

Granular Privileges

- **Why granular privileges?**

- The need to restrict access to specific portion of data in a table

- **How to implement?**

- **Views**

- 1) Simulate a new table
- 2) Create a view (subset of the data from the base table)
- 3) Authorize the user to access the view
- 4) Revoke access from the user to the base table

- **LBAC (Label Based Access Control)**

- Can restrict read/write access to row(s) and/or column(s) of a table

16

© 2010 IBM Corporation

Granular Privileges – Views

- Provides a different way of looking at data in one or more tables; it is a named specification of a result table.

EMPLOYEE TABLE				
LASTNAME	WORKDIV	OFFICE	SALARY	BONUS
Smith	A0	Toronto	60000	2500
Cmic	A0	Vancouver	65000	1500
Johnson	B1	Calgary	55000	1000
Carlson	C2	Ottawa	70000	2200
Pogue	B1	Toronto	50000	2800
Ring	B1	Victoria	52000	3000
Barisic	A0	Ottawa	67000	1200

- Allows multiple users to see different presentations of the same data

EMPLOYEE_INFO VIEW		
LASTNAME	WORKDIV	OFFICE
Smith	A0	Toronto
Cmic	A0	Vancouver
Johnson	B1	Calgary
Carlson	C2	Ottawa
Pogue	B1	Toronto
Ring	B1	Victoria
Barisic	A0	Ottawa

```
CREATE VIEW EMPLOYEE_INFO AS (
SELECT LASTNAME, WORKDIV, OFFICE
FROM EMPLOYEE);
```

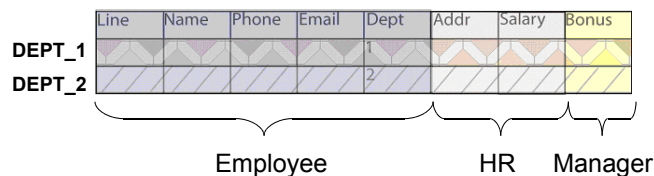
- Nice for simple security policy, but complicated to manage in large settings

17

© 2010 IBM Corporation

Granular Privileges – Label Based Access Control (LBAC)

- **Access Control at the table level via traditional privileges**
 - Does the user hold the required privilege to perform the requested operation on the table?
- **Label Based Access Control**
 - Sets security labels at the row level, column level or both
- **How does LBAC work?**
 - Users and Objects (rows/columns) are assigned labels that are later compared to authorize access

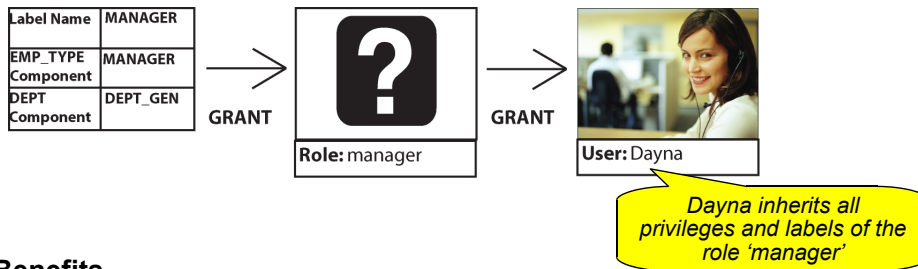


18

© 2010 IBM Corporation

Roles

- Database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC or to other roles via a GRANT statement.



Benefits

- SECADMs control access at a level of abstraction that is close to the structure of the organization. (Eg. Manager, HR, Employee)
- The assignment and maintenance of privileges is simplified.
 - User roles change □ Revoke old role and grant new role – not specific privileges
 - Role has more responsibility □ All users inherit the new privileges

19

© 2010 IBM Corporation

Roles – Implementation

The Basics

- Step 1 – Create Role

```
CREATE ROLE DEVELOPER
```

- Step 2 – Assign Privileges to a Role

```
GRANT SELECT ON TABLE
SERVER TO ROLE
DEVELOPER
```

- Step 3 – Grant Role to Users

```
GRANT ROLE DEVELOPER TO
USER BOB, USER ALICE
```

- Step 4 – Revoke Role as Necessary

```
REVOKE ROLE DEVELOPER
FROM USER BOB
```

Extra Features

- Role Admin Option

– Allows the specified user to grant or revoke the role to or from others

```
GRANT ROLE DEVELOPER TO USER
BOB WITH ADMIN OPTION
```

- Role Hierarchies

– A role hierarchy is formed when one role is granted membership in another role.

```
CREATE ROLE DOCTOR
CREATE ROLE SPECIALIST
CREATE ROLE SURGEON
```

```
GRANT ROLE DOCTOR TO ROLE
SPECIALIST
GRANT ROLE SPECIALIST TO
ROLE SURGEON
```



20

© 2010 IBM Corporation

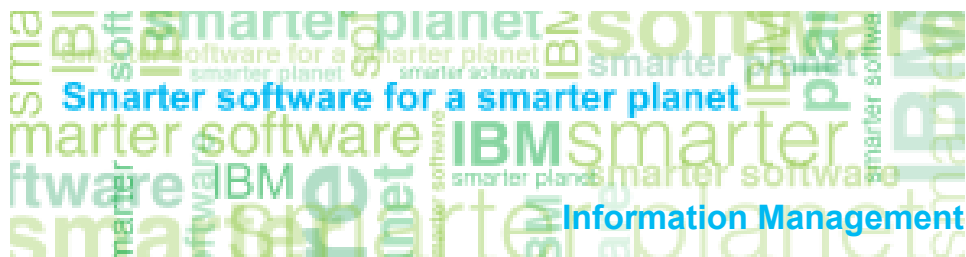
Summary

- **Authentication**
 - Verifies the user are who they say they are using the underlying operating system or other security protocols
- **Trusted Context**
 - Solves the problems associated with loss of user identity in a 3-tiered environment
- **Authorization**
 - Controls the access to database objects
- **Granular Privileges**
 - Access to specific portion of data in a table can be restricted using views and LBAC
- **Roles**
 - Allows easy management of privileges

Questions?

Summer/Fall 2010

E-mail: imschool@us.ibm.com
Subject: “DB2 Academic Workshop”





IBM DB2[®] 9.7

**DB2 Security
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

CONTENTS	2
1. INTRODUCTION	3
2. SUGGESTED READING	3
3. BASIC SETUP	3
3.1 Environment Setup Requirements.....	3
3.2 Preparation Steps.....	4
4. AUTHENTICATION	4
4.1 Where Does Authentication Take Place?.....	4
4.2 Specifying Authentication Type on the Server.....	6
4.3 Specifying Authentication Type on the Client.....	7
4.4 Using Data Studio to Manage Authentication Parameters	7
5. AUTHORIZATION	10
5.1 Authorities.....	10
5.1.1 INSTANCE-LEVEL AUTHORITIES	10
5.1.2 DATABASE-LEVEL AUTHORITIES	11
5.2 Privileges	12
5.3 Exercise - Granting and Revoking Authorities and Privileges	12
5.4 Granular Privileges - Views	14
6. ROLE	16
6.1 Example - Roles	16

1. Introduction

Your database system may contain confidential and sensitive data so it is important to safeguard your information. In order to prevent identity theft, it is crucial to control who has access to your database and limit the operations that the user can perform on the data.

In this lab, you will learn how to control access to the instance, how to control access to the database itself, and finally how to control access to the data and data objects within the database.

By the end of this lab, you will be able to:

- ▶ Grant and revoke authorities to/from users
- ▶ Grant and revoke privileges to/from users
- ▶ Create roles
- ▶ Grant and revoke roles to/from users

2. Suggested reading

Understanding DB2 9 Security

by Rebecca Bond (Author), Kevin Yeung-Kuen See (Author), Carmen Ka Man Wong (Author), Yuk-Kuen Henry Chan (Author)

3. Basic Setup

3.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 9.7 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in **VMware Basics and Introduction** from module 1.

3.2 Preparation Steps

1. Start the VMware® image. Once loaded and prompted for login credentials, use the user “db2inst1” to provide DBADM authority:

User: **db2inst1**

Password: **password**

2. Open a terminal window by right-clicking on the Desktop and choosing the “Open Terminal” item:

3. Start the Database Manager by issuing the following command:

```
db2start
```

Note: Disregard the warning message if the database manager is already active.

For executing this lab, you will need the DB2’s sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the **SAMPLE** database:

```
db2 force applications all
db2 drop db sample
db2sAMPL
```

4. Authentication

When you first attempt to access an instance or database, the authentication system will try to determine if you are who you say you are. DB2 authentication works closely with the authentication mechanism of the underlying operating system to verify your user IDs and passwords. DB2 can also use third-party authentication facilities such as Kerberos to authenticate users.

By using an external authentication system outside of DB2, DB2 does not need to store a redundant set of passwords and sensitive credentials. This minimizes security vulnerabilities and hacker attacks.

4.1 Where Does Authentication Take Place?

Authentication type defines where and how authentication will take place. This is specified by the AUTHENTICATION parameter in the database manager configuration

file on the server, thus all the databases in an instance will have the same authentication type. On the client, the authentication type is specified when a remote database is cataloged.

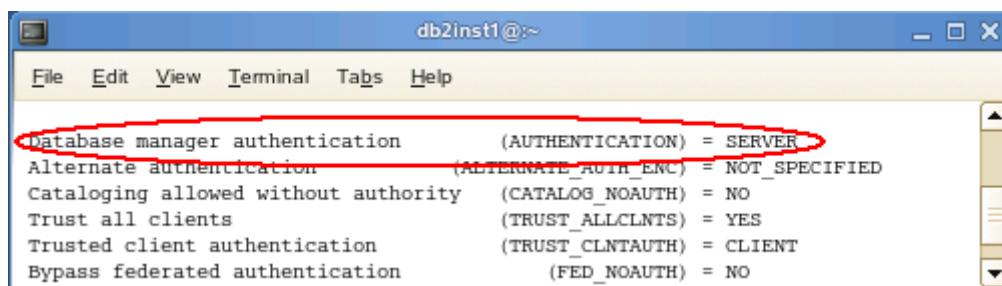
Authentication Type	Description
SERVER	All authentications take place at the server. When you connect to a database, you will have to include your user ID and password. This information will then be verified against the credentials at the server's operating system.
SERVER_ENCRYPT	This is similar to SERVER authentication type where authentication occurs at the server, but the password is encrypted by DB2 at the client before it is sent to the server for authentication.
CLIENT	Authentication occurs at the client's operating system.
KERBEROS	Authentication occurs at the server and is handled by Kerberos security software. The KERBEROS authentication type is available if both the DB2 server and client operating systems support Kerberos. The Kerberos security protocol uses conventional cryptography to create a shared secret key which becomes the credentials used to verify the identity of the user. This eliminates the need to pass a user ID and password across the network.
KRB_SERVER_ENCRYPT	This authentication type is the same as KEBREOS, except it will use SERVER_ENCRYPT if the client does not support Kerberos security system. If none of these options are available, the client will receive a connection error and will not be able to connect.
DATA_ENCRYPT	Authentication occurs at the server and its behaviour is similar to SERVER_ENCRYPT. In this type of authentication, not only is the password encrypted, but all user data is also encrypted during transmission between the client and the server.
DATA_ENCRYPT_CMP	This type of authentication is identical to DATA_ENCRYPT. However, this setting provides compatibility to those clients who do not support DATA_ENCRYPT authentication and will instead connect using SERVER_ENCRYPT so user data will not be encrypted.
GSSPLUGIN	Authentication occurs at the server using an external GSS-API plug-in. If the client's authentication type is not specified, the server will send a list of server-supported plug-ins to the client. These plug-ins are listed in the <code>srvcon_gssplugin_list</code> database manager configuration parameter. The client then selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list, the client is authenticated using the KERBEROS authentication method.
GSS_SERVER_ENCRYPT	Authentication occurs at the server using either the GSSPLUGIN or the SERVER_ENCRYPT authentication

method. Authentication uses a GSS-API plug-in and if the client does not support any of the plug-ins found in the server-supported plug-ins list, the client is authenticated using KERBEROS. If the client does not support the Kerberos security protocol, the client is authenticated using the SERVER_ENCRYPT authentication method.

4.2 Specifying Authentication Type on the Server

1. To check the current authentication type, issue the following command. In this case, the current authentication method is SERVER.

```
db2 GET DATABASE MANAGER CONFIGURATION
```

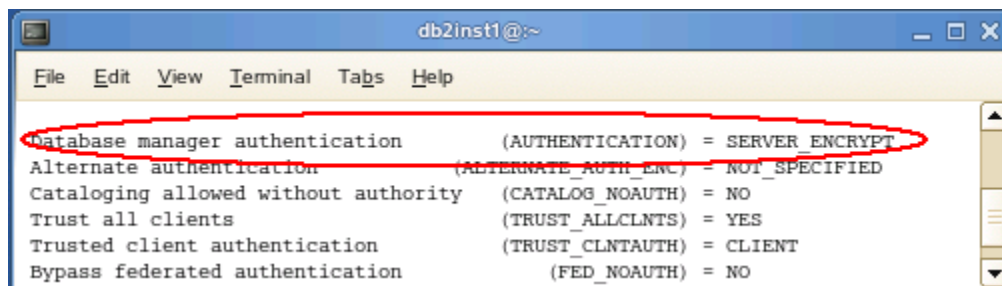


```
db2inst1@~
File Edit View Terminal Tabs Help
Database manager authentication (AUTHENTICATION) = SERVER
Alternate authentication (ALTERNATE_AUTH_ENC) = NOT_SPECIFIED
Cataloging allowed without authority (CATALOG_NOAUTH) = NO
Trust all clients (TRUST_ALLCLNTS) = YES
Trusted client authentication (TRUST_CLNTAUTH) = CLIENT
Bypass federated authentication (FED_NOAUTH) = NO
```

2. Change the authentication setting to SERVER_ENCRYPT by executing the following command. You must be a member of the SYSADM group to make changes to security-related configuration parameters for an instance.

```
db2 UPDATE DBM CFG USING AUTHENTICATION SERVER_ENCRYPT
```

3. Re-issue the command from step 1 to check the current authentication setting.



```
db2inst1@~
File Edit View Terminal Tabs Help
Database manager authentication (AUTHENTICATION) = SERVER_ENCRYPT
Alternate authentication (ALTERNATE_AUTH_ENC) = NOT_SPECIFIED
Cataloging allowed without authority (CATALOG_NOAUTH) = NO
Trust all clients (TRUST_ALLCLNTS) = YES
Trusted client authentication (TRUST_CLNTAUTH) = CLIENT
Bypass federated authentication (FED_NOAUTH) = NO
```

4. Change the authentication setting back to SERVER by executing the following command.

```
db2 UPDATE DBM CFG USING AUTHENTICATION SERVER
```

4.3 Specifying Authentication Type on the Client

The client authentication type is stored in the client's database directory. To see the list of databases known to the system, use the following command:

```
db2 LIST DATABASE DIRECTORY
```

To change the authentication type for a connection, the database needs to be re-cataloged from the database directory with the new authentication type.

The specification of the authentication type when cataloging the remote client is optional. If an authentication type is specified, it must match or be compatible with the value specified at the data server. If they do not match, the connection will fail.

To catalog a database connection using the SERVER_ENCRYPT authentication you can enter the following command:

```
db2 CATALOG DATABASE sample AT NODE mynode AUTHENTICATION
SERVER_ENCRYPT
```

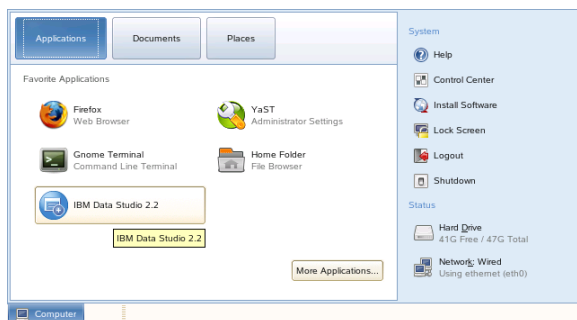
Note: since the database could be already cataloged, you may receive the error message:

SQL1005N The database alias "sample" already exists in either the local database directory or system database directory.

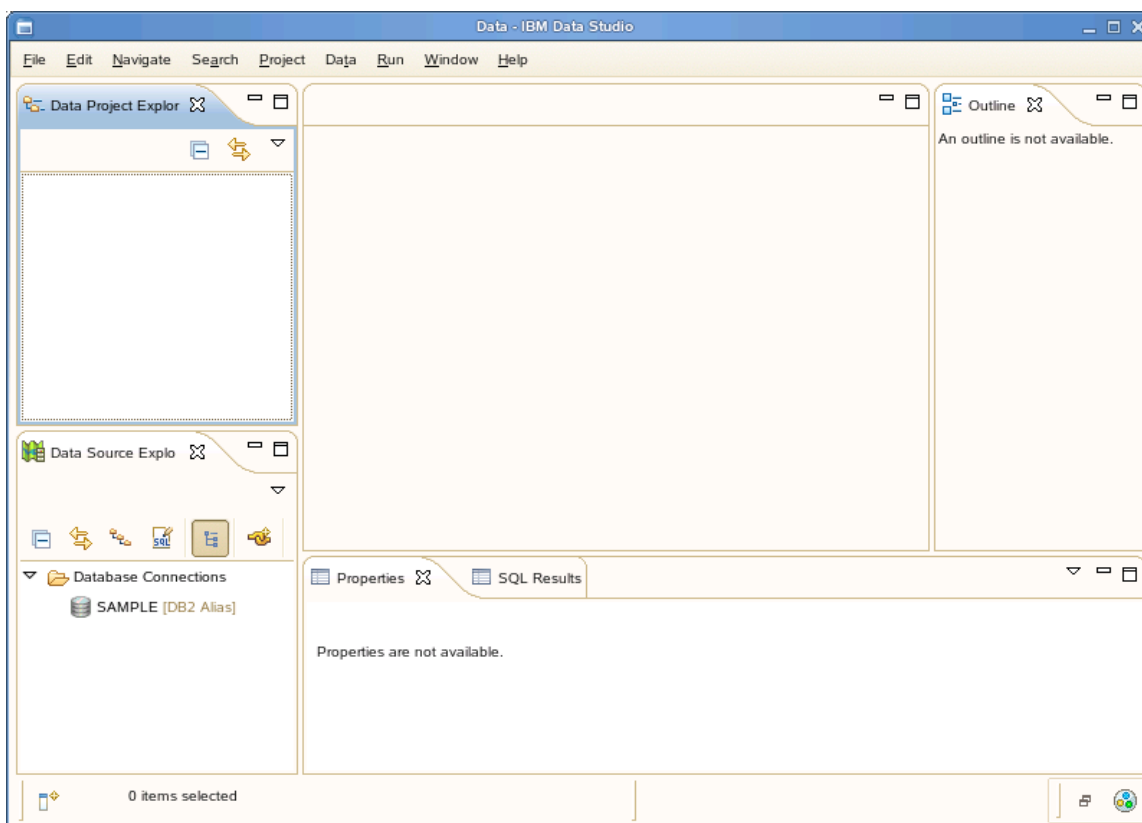
4.4 Using Data Studio to Manage Authentication Parameters

Configuration of the authentication type can also be easily managed through the Data Studio.

1. Launch Data Studio by clicking on the **Computer** button in the bottom left corner of the screen, and select **IBM Data Studio 2.2**.



2. In the **Select a workspace** dialog, accept the default path and check the **Use this as the default** and **do not ask again** checkbox. Click **OK**.
3. Minimize the Welcome window to bring you into the Data perspective as shown below.



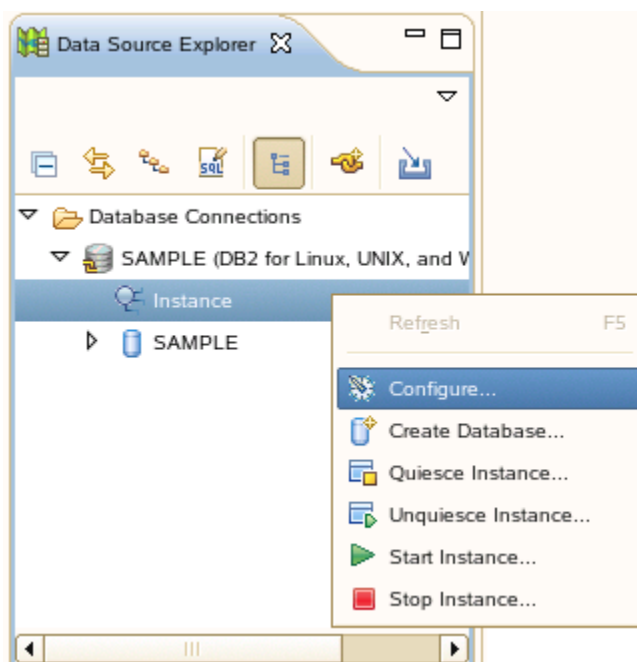
4. Connect to the database Sample.

From the **Database Source Explorer** panel (bottom left panel), expand Connections. Right-click on the SAMPLE database and select **Connect**. Login with the following credentials:

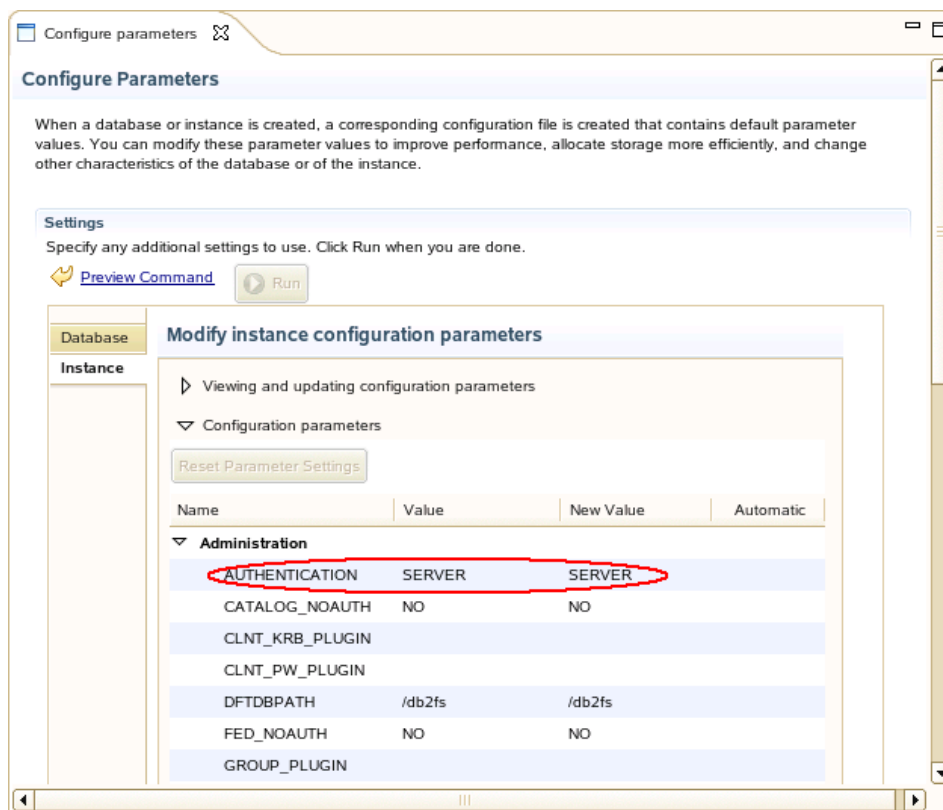
User: db2inst1

Password: password

5. After a connection has been established, right click on **Instance** and select **Configure** to see the instance configuration parameters.



The authentication parameter is shown at the top of the configuration parameters dialog. To change the current setting, simply double click on the parameter and specify a new value for authentication.



5. Authorization

After a user has been authenticated, authorization serves as the second security mechanism which determines what operations a user can perform within a database or instance. Authorization consists of the privileges, authorities, roles, and label-based access control (LBAC) credentials.

A user's authorities determine his/her ability to perform high-level database and instance management operations.

Privileges provide more granular control than authorities. Privileges define the objects that a user can create or drop and commands that a user can use to access objects like tables, views, indexes, and packages.

Roles are a way of collecting users together, so that privileges can be managed together instead of individually.

LBAC uses security labels to control who has read access and who has write access to individual rows and/or columns in a table. LBAC is not included in DB2 Express-C and the implementation of LBAC is beyond the scope of this lab.

5.1 Authorities

Authorities are needed for managing databases and instances and can be divided into two groups:

- Instance-level authorities
- Database-level authorities

5.1.1 Instance-level Authorities

Instance level authorities enable you to perform instance-wide functions, such as creating and upgrading databases, managing table spaces, and monitoring activity and performance on your instance. No instance-level authority provides access to data in database tables.

Database-level Authorities	Descriptions
SYSADM	for users managing the instance as a whole
SYSCTRL	for users administering a database manager instance
SYSMAINT	for users maintaining databases within an instance

SYSMON	for users monitoring the instance and its databases
--------	---

Instance-level authorities are granted through the database manager configuration and can only be assigned to groups. Groups are defined at the operating system level and individual users are assigned to these groups. To grant SYSADM, SYSCTRL, SYSMANT or SYSMON authority to a group, set the database manager configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, SYSMANT_GROUP and SYSMON_GROUP to an operating system group.

By default, on UNIX systems, the SYSADM group is set to the primary group of the instance owner DB2GRP1. Any users that belong to this group have SYSADM authority. On Windows, members of the local Administrators group are all granted SYSADM authority.

From the command below, you can see that DB2GRP1 is defined as SYSADM group.

```
db2 get dbm cfg | grep SYSADM_GROUP
```

```

db2inst1@db2rules:~> db2 get dbm cfg | grep SYSADM_GROUP
SYSADM group name (SYSADM_GROUP) = DB2GRP1
db2inst1@db2rules:~>

```

5.1.2 Database-level Authorities

Database authorities enable users to perform activities at the database level, thus allowing the users to perform such functions as granting and revoking privileges, inserting, selecting, deleting and updating data, and managing workloads.

Database-level Authorities	Descriptions
SECADM	for users managing security within a database
DBADM	for users administering a database
ACCESSCTRL	for users who need to grant and revoke authorities and privileges (except for SECADM, DBADM, ACCESSCTRL, and DATAACCESS authority, SECADM authority is required to grant and revoke these authorities)
DATAACCESS	for users who need to access data
SQLADM	for users who monitor and tune SQL queries
WLMADM	for users who manage workloads
EXPLAIN	for users who need to explain query plans

5.2 Privileges

Privileges are more granular than authorities. They define the objects that a user or group can create, alter, or drop, and access database objects. Privileges can be obtained in three different ways:

Explicit: Privileges can be explicitly be given or taken away by users with ACCESSCTRL authority, SECADM authority or CONTROL privilege on that object using the GRANT or REVOKE command. A user who has been assigned privilege with the WITH GRANT OPTION on an object can also explicitly grant privileges.

Implicit: When a user creates a database object, that user will implicitly receive all privileges for that object. For example, when a user creates a database, that user implicitly receives DBADM authority for that database.

Indirect: An indirect privilege is usually associated with a package. When a user executes a package, it may require privileges that the user does not have. The user will be indirectly given these privileges temporarily, in order to execute the package.

5.3 Exercise - Granting and Revoking Authorities and Privileges

Thus far in the lab, you have been issuing all database commands as the instance administrator (db2inst1) which has privileges to access all the utilities, data, and database objects within DB2. It is important that users be only given privileges that are necessary to complete their tasks.

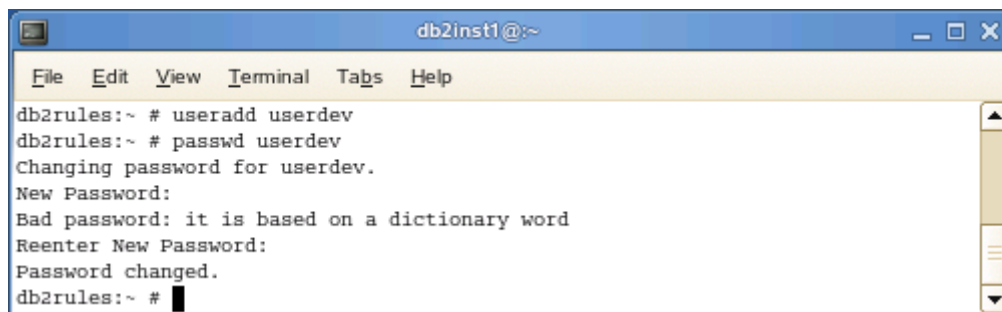
In the following scenario a new member has joined your team. We will look at how to assign specific authorities and privileges to him to safeguard the security of the database.

1. Open a terminal window by right-clicking on the Desktop and choosing the “Open Terminal” item.
2. DB2 uses the underlying operating system’s security to manage users and passwords. Thus we need to create the new users and groups at the operating system level.

Login to the operating system as the root user and add a new user USERDEV. Change his password to ‘password’

```
su -  
    Password: password  
  
useradd userdev  
passwd userdev  
    New password: password
```

```
exit
```



```
db2rules:~ # useradd userdev
db2rules:~ # passwd userdev
Changing password for userdev.
New Password:
Bad password: it is based on a dictionary word
Reenter New Password:
Password changed.
db2rules:~ # █
```

3. Authorities and privileges are implicitly denied if not granted. When the new user is added, he has no authorities or privileges other than those defined in the PUBLIC group.

Try querying the 'EMPLOYEE' table of sample database as user USERDEV and you will see that the operation will be denied because USERDEV doesn't have the required authorization or privilege.

```
db2
```

```
CONNECT TO SAMPLE USER userdev USING password
SELECT * FROM DB2INST1.EMPLOYEE
```



```
db2 => CONNECT TO SAMPLE USER userdev USING password

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = USERDEV
Local database alias    = SAMPLE

db2 => SELECT * from DB2INST1.EMPLOYEE
SQL0551N  "USERDEV" does not have the required authorization or privilege to
perform operation "SELECT" on object "DB2INST1.EMPLOYEE".  SQLSTATE=42501
db2 => █
```

4. USERDEV is an application developer within your team and he will develop and test programs. He needs to have SELECT, INSERT, UPDATE and DELETE access to the various tables in the database. He also needs to be able to add new packages to the database and execute the application to test it; therefore, he needs to be granted the BINDADD authority.

To grant these privileges to USERDEV, you must be a SYSADM. **Log in to your machine as the instance owner for DB2 (db2inst1)**, and issue the GRANT command.

```
CONNECT TO SAMPLE USER db2inst1 USING password
GRANT CREATETAB, BINDADD, CONNECT ON DATABASE TO USER userdev
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE employee TO USER userdev
```

5. USERDEV now has the privilege to query and modify the table EMPLOYEE. Try re-running the commands from step 3.

```

db2inst1@:~
File Edit View Terminal Tags Help
db2 => CONNECT TO SAMPLE USER userdev USING password

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = USERDEV
Local database alias    = SAMPLE

db2 => SELECT * FROM DB2INST1.EMPLOYEE

EMPNO  FIRSTNME  MIDINIT  LASTNAME  WORKDEPT  PHONENO  HIREDATE  JOB  EDLEVEL  SEX  BIRTHDATE  SALARY  BONUS  COMM
-----
000010 CHRISTINE I  HAAS     A00      3978     01/01/1995 PRES   18 F  08/24/1963 152750.00 1000.00 4220.00
000020 MICHAEL  L  THOMPSON B01      3476     10/10/2003 MANAGER 18 M  02/02/1978 94250.00 800.00 3300.00
000030 SALLY    A  KWAN     C01      4738     04/05/2005 MANAGER 20 F  05/11/1971 98250.00 800.00 3060.00
000050 JOHN     B  GEYER    E01      6789     08/17/1979 MANAGER 16 M  09/15/1955 80175.00 800.00 3214.00
000060 IRVING   F  STERN    D11      6423     09/14/2003 MANAGER 16 M  07/07/1975 72250.00 500.00 2580.00
000070 EVA      D  PULASKI D21      7831     09/30/2005 MANAGER 16 F  05/26/2003 96170.00 700.00 2893.00
000090 EILEEN  W  HENDERSON E11      5498     08/15/2000 MANAGER 16 F  05/15/1971 89750.00 600.00 2380.00
000100 THEODORE Q  SPENSER  E21      0972     06/19/2000 MANAGER 14 M  12/18/1980 86150.00 500.00 2092.00
000110 VINCENZO G  LUCCHESI A00      3490     05/16/1988 SALESREP 19 M  11/05/1959 66500.00 900.00 3720.00
000120 SEAN    O'CONNELL A00      2167     12/05/1993 CLERK   14 M  10/18/1972 49250.00 600.00 2340.00
000130 DELORES M  QUINTANA C01      4578     07/28/2001 ANALYST 16 F  09/15/1955 73800.00 500.00 1904.00
000140 HEATHER A  NICHOLLS C01      1793     12/15/2006 ANALYST 18 F  01/19/1976 68420.00 600.00 2274.00
000150 BRUCE   ADAMSON  D11      4510     02/12/2002 DESIGNER 16 M  05/17/1977 55280.00 500.00 2022.00
000160 ELIZABETH R  PIANKA  D11      3782     10/11/2006 DESIGNER 17 F  04/12/1980 62250.00 400.00 1780.00
000170 MASATOSHI J  YOSHIMURA D11      2890     09/15/1999 DESIGNER 16 M  01/05/1981 44680.00 500.00 1974.00
000180 MARILYN S  SCOTTEN  D11      1682     07/07/2003 DESIGNER 17 F  02/21/1979 51340.00 500.00 1707.00
000190 JAMES   H  WALKER  D11      2986     07/26/2004 DESIGNER 16 M  06/25/1982 50450.00 400.00 1636.00
000200 DAVID   BROWN   D11      4501     03/03/2002 DESIGNER 16 M  05/29/1971 57740.00 600.00 2217.00
000210 WILLIAM T  JONES   D11      0942     04/11/1998 DESIGNER 17 M  02/23/2003 68270.00 400.00 1462.00
000220 JENNIFER K  LUTZ    D11      0672     08/29/1998 DESIGNER 18 F  03/19/1978 49840.00 600.00 2387.00
000230 JAMES   J  JEFFERSON D21      2094     11/21/1996 CLERK   14 M  05/30/1980 42180.00 400.00 1774.00
000240 SALVATORE M  MARINO  D21      3780     12/05/2004 CLERK   17 M  03/31/2002 48760.00 600.00 2301.00
000250 DANIEL  S  SMITH   D21      0961     10/30/1999 CLERK   15 M  11/12/1969 49180.00 400.00 1534.00
000260 SYBIL   P  JOHNSON D21      8953     09/11/2005 CLERK   16 F  10/05/1976 47250.00 300.00 1380.00
000270 MARIA   L  PEREZ   D21      9001     09/30/2006 CLERK   15 F  05/26/2003 37380.00 500.00 2190.00
000280 ETHEL   R  SCHNEIDER E11      8997     03/24/1997 OPERATOR 17 F  03/28/1976 36250.00 500.00 2100.00
000290 JOHN    R  PARKER  E11      4502     05/30/2006 OPERATOR 12 M  07/09/1985 35340.00 300.00 1227.00
000300 PHILIP  X  SMITH   E11      2095     06/19/2002 OPERATOR 14 M  10/27/1976 37750.00 400.00 1420.00
000310 MAUDE   F  SETRIGHT E11      3332     09/12/1994 OPERATOR 12 F  04/21/1961 35900.00 300.00 1272.00
000320 RAMLAL  V  MEHTA  E21      9990     07/07/1995 FIELDREP 16 M  08/11/1962 39950.00 400.00 1596.00
000330 WING    L  LEE     E21      2103     02/23/2006 FIELDREP 14 M  07/18/1971 45370.00 500.00 2030.00
000340 JASON   R  GOUNOT  E21      5698     05/05/1977 FIELDREP 16 M  05/17/1956 43840.00 500.00 1907.00
200010 DIAN    J  HEMMINGER A00      3978     01/01/1995 SALESREP 18 F  08/14/1973 46500.00 1000.00 4220.00
200120 GREG    ORLANDO A00      2167     05/05/2002 CLERK   14 M  10/18/1972 39250.00 600.00 2340.00
200140 KIM     N  NATZ    C01      1793     12/15/2006 ANALYST 18 F  01/19/1976 68420.00 600.00 2274.00
200170 KIYOSHI YAMAMOTO D11      2890     09/15/2005 DESIGNER 16 M  01/05/1981 64680.00 500.00 1974.00
200220 REBA    K  JOHN    D11      0672     08/29/2005 DESIGNER 18 F  03/19/1978 69840.00 600.00 2387.00
200240 ROBERT  M  MONTEVERDE D21      3780     12/05/2004 CLERK   17 M  03/31/1984 37760.00 600.00 2301.00
200280 EILEEN  R  SCHWARTZ E11      8997     03/24/1999 OPERATOR 17 F  03/28/1966 46250.00 500.00 2100.00
200310 MICHELLE F  SPRINGER E11      3332     09/12/1994 OPERATOR 12 F  04/21/1961 35900.00 300.00 1272.00
200330 HELENA  WONG    E21      2103     02/23/2006 FIELDREP 14 F  07/18/1971 35370.00 500.00 2030.00
200340 ROY     R  ALONZO  E21      5698     07/05/1997 FIELDREP 16 M  05/17/1956 31840.00 500.00 1907.00

42 record(s) selected.

db2 =>

```

5.4 Granular Privileges - Views

There are two ways in which access to specific portions of data in a table can be restricted: views or label based access control (LBAC). LBAC is not included in DB2 Express-C and the implementation of LBAC is beyond the scope of this lab. We will instead focus on the implementation of views.

Views are virtual tables (computed dynamically and not stored explicitly) that are derived from one or more tables or views. They can be used to provide a customized subset of data to the users, allowing them to see different presentations of the same set of data or hide data to which a user should not have access. Views can perform delete, insert and

update operations, or be read-only. The classification indicates the kind of SQL operations allowed against the view.

Using the employee table from sample database, we will demonstrate how to implement views.

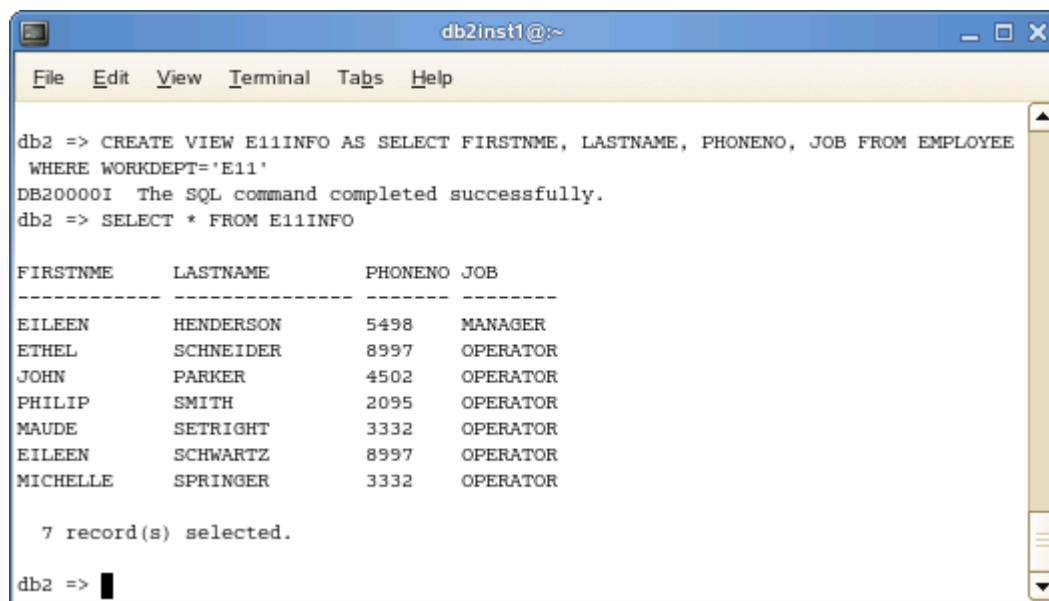
The employee table contains confidential information such as employees' salaries and bonuses. This information should not be seen by everyone. In order to protect this confidential information, a view can be created based on the employee table to restrict users from seeing certain columns. This will grant users access to the view rather than to the base table.

1. We would like to create a view that contains a directory of those who are in department E11. This directory will contain only first name, last name, phone number and job role.

```
CONNECT TO SAMPLE
CREATE VIEW E11INFO AS SELECT FIRSTNME, LASTNAME, PHONENO, JOB FROM
EMPLOYEE WHERE WORKDEPT='E11'
```

2. A user issuing a select statement against the view will see only four columns:

```
SELECT * FROM E11INFO
```



```
db2inst1@~
File Edit View Terminal Tabs Help

db2 => CREATE VIEW E11INFO AS SELECT FIRSTNME, LASTNAME, PHONENO, JOB FROM EMPLOYEE
WHERE WORKDEPT='E11'
DB200001 The SQL command completed successfully.
db2 => SELECT * FROM E11INFO

FIRSTNME      LASTNAME      PHONENO      JOB
-----
EILEEN        HENDERSON     5498         MANAGER
ETHEL         SCHNEIDER     8997         OPERATOR
JOHN          PARKER        4502         OPERATOR
PHILIP        SMITH         2095         OPERATOR
MAUDE         SETRIGHT      3332         OPERATOR
EILEEN        SCHWARTZ      8997         OPERATOR
MICHELLE      SPRINGER      3332         OPERATOR

7 record(s) selected.

db2 => █
```

3. The last step includes revoking access to the base table and granting access to the view instead:

```
REVOKE ALL ON employee FROM USER userdev
GRANT SELECT ON e11info TO USER userdev
```

6. Role

A role is a database object that may group together one or more privileges and can be assigned to users, groups, PUBLIC or to other roles via a GRANT statement. Roles simplify the administration and management of privileges.

Roles can be modeled after the structure of an organization. They can be created to map directly to specific job functions within the organizations. Instead of granting the same set of privileges to each individual user in a particular job function, this set of privileges can be granted to a role and then users are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily granted and revoked.

6.1 Example - Roles

Continuing the scenario from the previous section, your team is expanding and more application developers have joined your team. Instead of managing each of these individuals' privileges, it is easier to administer and manage if roles are used.

The security administrator holds the authority to create, drop, grant, revoke and comment on a role.

1. Connect to the sample database and create a new role called 'developer'.

```
CONNECT TO SAMPLE
CREATE ROLE DEVELOPER
```

2. After a role has been defined, use the GRANT statement to assign authorities and privileges to the role.

```
GRANT CREATETAB, BINDADD, CONNECT ON DATABASE TO ROLE developer
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE db2inst1.employee TO ROLE
developer
```

3. The role DEVELOPER is granted to user USERDEV:

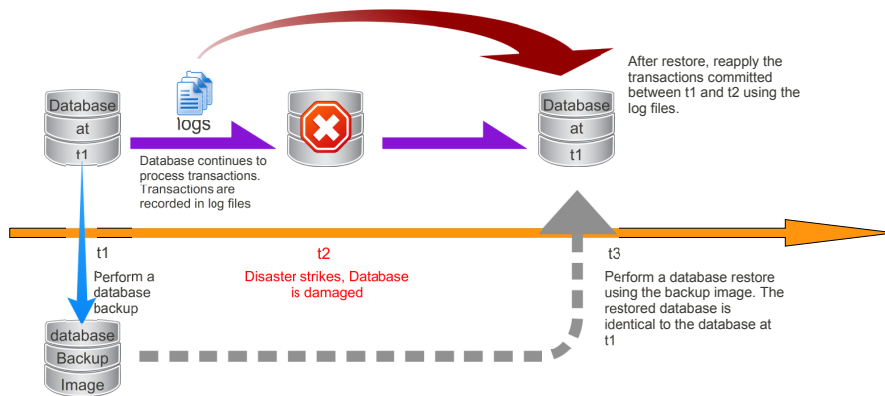
```
GRANT ROLE DEVELOPER TO USER USERDEV
```

4. When USERDEV changes his role and is no longer a developer, his role can be revoked from the database.

```
REVOKE ROLE DEVELOPER FROM USER USERDEV
```


Basic Concept of Database Backup and Recovery

- At t1, a database backup operation is performed
- At t2, a problem that damages the database occurs
- At t3, all committed data is recovered



3

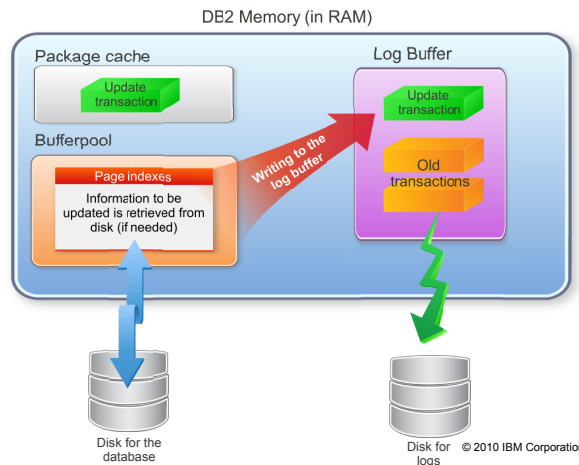
© 2010 IBM Corporation

DB2 Transaction Logs

- Keep track of changes made to database objects and their data
- During the recovery process, DB2 examines these logs and decides which changes to redo or undo
- Can be stored in files or on raw devices

- The transactions in the log buffer are recorded in the log device upon one of the following events:

- Log buffer is full
- Number of commits reach **MINCOMMIT** value
- One second has lapsed

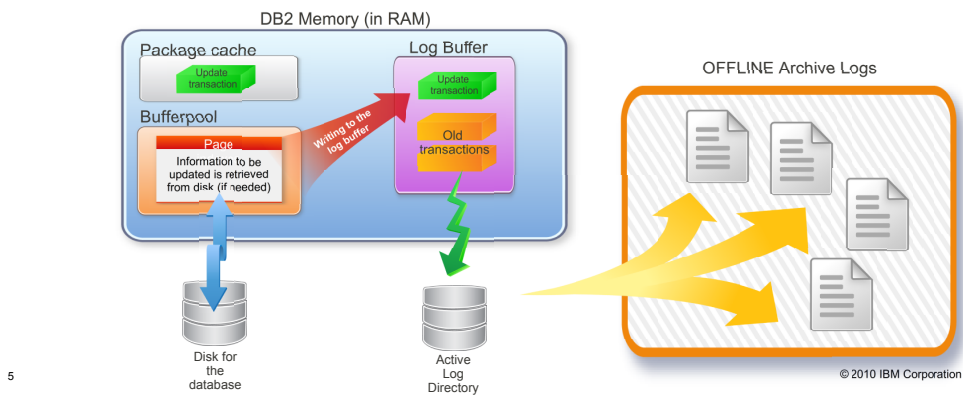


4

© 2010 IBM Corporation

Log File States

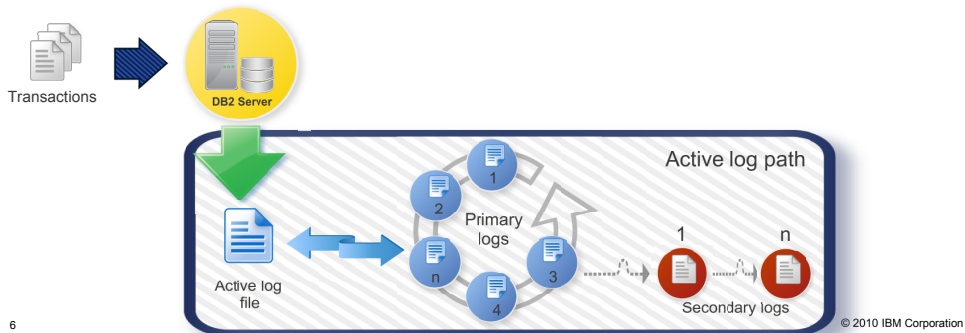
- **Active logs**
 - Transactions that have not been committed or rolled back
- **Online archive logs**
 - Committed and externalized logs in the active log directory
- **Offline archive logs**
 - Committed and externalized logs in a separate repository



5

Circular Logging

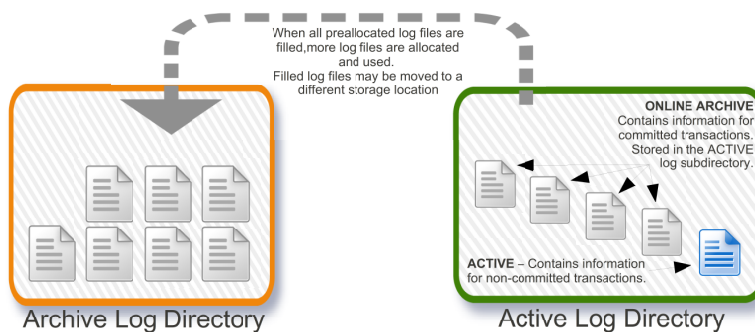
- **Primary log files used to record all transactions; reused when transactions are committed**
- **Secondary log files allocated when next primary log file is not available due to active transactions**
- **If both primary and secondary log limit are full and can not be reused, a log full condition occurs and SQL0964C error message is returned**
- **Only full, offline backups of the database are allowed**
- **Cannot have roll-forward recovery**



6

Archival Logging

- Enable with **LOGARCHMETH1** database configuration parameter
- History of log files is maintained, in order to allow roll forward recovery and online backup
- Logs can be optionally archived to an archive location when no longer active to avoid exhaustion of log directory



7

© 2010 IBM Corporation

Infinite Logging

- Infinite logging provides infinite active log space
 - Enabled by setting **LOGSECOND to -1**
- Secondary log files are allocated until the unit of work commits or storage is exhausted
- Archived logs can hinder performance for rollback and crash recovery
- Database must be configured to use **archival logging**
- Up to 256 log files (primary + secondary)
- Control parameters
 - **NUM_LOG_SPAN** – number of log files an active transaction can span
 - **MAX_LOG** – Percentage of active primary log file space that a single transaction could consume

8

© 2010 IBM Corporation

Database Backup

- **Copy of a database or table space**

- User data
- DB2 catalogs
- All control files, e.g. buffer pool files, table space file, database configuration file



- **Backup modes:**

- **Offline Backup**

- Does not allow other applications or processes to access the database
- Only option when using circular logging

- **Online Backup**

- Allows other applications or processes to access the database
- Available to users during backup
- Can backup to disk, tape, TSM and other storage vendors

9

© 2010 IBM Corporation

Database Backup – Syntax

```
db2 backup database <db_name> <online> to <dest_path>
```

Online backup example

```
db2 backup database mydb online to /home/db2inst1/backups
```

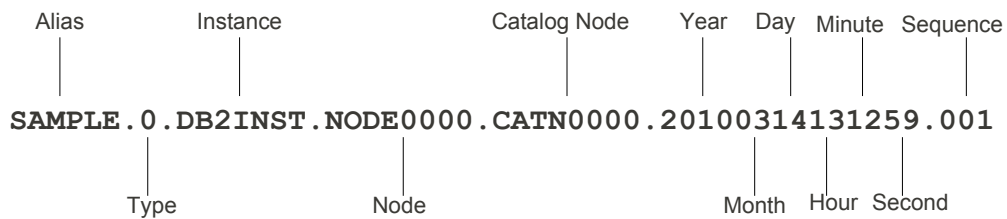
Offline backup example

```
db2 backup database mydb to /home/db2inst1/backups
```

10

© 2010 IBM Corporation

Database Backup – File Naming Convention



Backup Type:

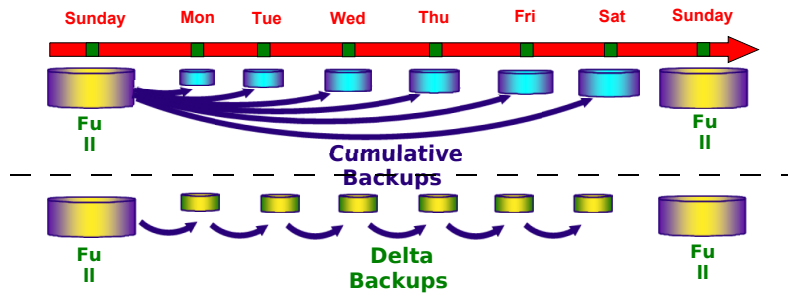
- 0 = Full Backup
- 3 = Tablespace Backup

Table space Backup

- Enables user to backup a subset of database
- **Multiple** table spaces can be specified
- Database must be using **archival logging**
- Table space backup can run in both **online** and **offline** backup
- Table space can be restored from either a database backup or table space backup of the given table space
- Use the keyword **TABLESPACE** to specify table spaces

```
db2 backup database mydb1 TABLESPACE (TBSP1) ONLINE to
/home/db2inst1/backup
```

Incremental Backups



- **Incremental (a.k.a. cumulative)** - Backup of all database data that has changed since the most recent, successful, full backup operation
- **Incremental Delta** - Backup of all database data that has changed since the last successful backup (full, incremental, or delta) operation.
- Need to have **TRACKMOD** database configuration parameter ON
- Supports both database and table space backups.
- Suitable for large databases, considerable savings by only backing up incremental changes.

13

© 2010 IBM Corporation

Database Backup – Compression

- **DB2 backups can now be automatically compressed**
 - Significantly reduce backup storage costs
- **Performance characteristics**
 - CPU costs typically increased (due to compression computation)
 - Media I/O time typically decreased (due to decreased image size)
 - Overall backup/restore performance can increase or decrease; depending on whether CPU or media I/O is a bottleneck

Example:

```
db2 backup database DS2 to /home/db2inst1/backups compress
```



14

© 2010 IBM Corporation

Automatic Database Backup

- **Simplifies database backup management tasks for the DBA by always ensuring that a recent full backup of the database is performed as needed**

- **To configure automatic backup**

- **Graphical user interface tools**

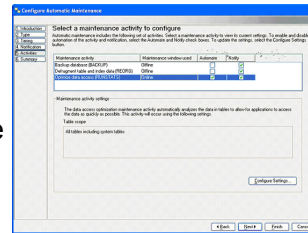
- Configure Automatic Maintenance wizard

- **Command line interface**

- **auto_db_backup**
- **auto_maint**

- **Stored procedure**

- **AUTOMAINT_SET_POLICY** system stored procedure



15

© 2010 IBM Corporation

Optimizing Backup Performance

- **DB2 automatically configures these parameters for performance**

- **Parallelism**

- Number of table spaces backed up in parallel

- **num_buffers**

- Number of buffers used
- Use at least **twice as many buffers** as backup targets (or sessions) to ensure that the backup target devices do not have to wait for data.

- **Buffer**

- Backup buffer size

- **Allocate more memory to backup utility by increasing utility heap size (**UTIL_HEAP_SZ**) configuration parameter.**

- **Backup subset of data where possible:**

- Table space backups
- Incremental backups

- **Use multiple target devices**

16

© 2010 IBM Corporation

Database Recovery

- Recovery is the rebuilding of a database or table space after a problem such as media or storage failure, power interruption, or application failure.



Types of Recovery

– Crash or restart recovery

- Protects the database from being left inconsistent (power failure)

– Version recovery

- Restores a snapshot of the database

– Roll forward recovery

- Extends version recovery by using full database and table space backup in conjunction with the database log files

- Crash recovery and version recovery are enabled in DB2 by default

17

© 2010 IBM Corporation

DB2 Restore Utility

- Restore utility is the complement of backup utility
- Restores database or table space from a previously taken backup
- **TAKEN AT** - Specify the time stamp of the database backup image. Backup image timestamp is displayed after successful completion of a backup
- **Without prompting** – Overrides any warnings.

Example:

```
SAMPLE .0 .DB2INST.NODE0000.CATN0000.20080718131210.001
```

```
RESTORE DATABASE dbalias FROM <db_path> TAKEN AT 20080718131210
```



18

© 2010 IBM Corporation

Table space Restore Operation

- Restored table space is in **Roll Forward Pending** state and can be either rolled forward to **End of Logs** or a **Point In Time**.
 - In case of Point in Time roll forward, table space must be rolled forward to at least the **minimum Point in Time**
- Minimum recovery time can be checked using
 - **db2 list tablespaces show detail**
- User table space must be in line with catalog table space
 - e.g if catalog indicates table T1 exists in table space TSP1, table T1 must exist in the TSP1 table space, otherwise database becomes inconsistent
- Every time there is a DDL changed, minimum recovery time for the table space is revised to indicate the last DDL change.
- Recommended to take a table space backup after a table space has been restore to a point in time.
- Transactions that came after the point in time are lost, therefore take a table space backup as new point of reference for future recoveries.

Incremental Restore

- Restore a database with incremental backup images
- **AUTOMATIC (recommended)** - All required backup images will be applied automatically by restore utility
- **MANUAL** – User applies the required backups manually
 - **db2ckrst** can provide the sequence for applying backups
- **ABORT** - aborts an in-progress manual cumulative restore
- **RESTORE DATABASE sample INCREMENTAL AUTOMATIC FROM /db2backup/dir1;**
- **ROLLFORWARD DATABASE sample TO END OF LOGS AND COMPLETE;**



IBM DB2[®] 9.7

**Backup and
Recovery
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

CONTENTS	2
1. INTRODUCTION	3
2. BASIC SETUP	3
2.1 Environment Setup Requirements.....	3
2.2 Preparation Steps.....	3
3. DB2 LOGGING	4
3.1 Logging Parameters	4
3.1.1 LOGFILSIZ.....	5
3.1.2 LOGPRIMARY AND LOGSECOND.....	5
3.1.3 LOGBUFSZ.....	6
3.2 Types of Logging	6
3.2.1 CIRCULAR LOGGING.....	6
3.2.2 ARCHIVAL LOGGING	6
4. RECOVERY SCENARIO	8
4.1 Scenario 1 - Entire database is accidentally dropped or becomes corrupted	8
4.2 Scenario 2 - Database Roll forward to a Point in Time.....	9
4.3 Scenario 3 – Incremental Backup and Restore	11
4.3.1 INCREMENTAL BACKUP.....	11
4.3.2 INCREMENTAL RESTORE	13

1. Introduction

Various situations may threaten the integrity of the database including system outage, hardware failure, transaction failure, and disaster. With DB2's backup and recovery, it prevents you from losing data.

By the end of this lab, you will be able to:

- ▶ Perform a full backup and restore
- ▶ Restore a database to a point in time
- ▶ Perform an incremental backup and restore

2. Basic Setup

2.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in **VMware Basics and Introduction** from module 1.

2.2 Preparation Steps

1. Start the VMware image. Once loaded and prompted for login credentials, use the user "db2inst1" to provide DBADM authority:

User: **db2inst1**

Password: **password**

2. Type in the command "**startx**" to bring up the graphical environment.

3. Open a terminal window by right-clicking on the Desktop and choosing the "Open Terminal" item:

4. If the Database Manager is not yet started, issue the following command:

```
db2start
```

For executing this lab, you will need the DB2's sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the **SAMPLE** database:

```
db2 force applications all
db2 drop db sample
db2sampl
```

3. DB2 Logging

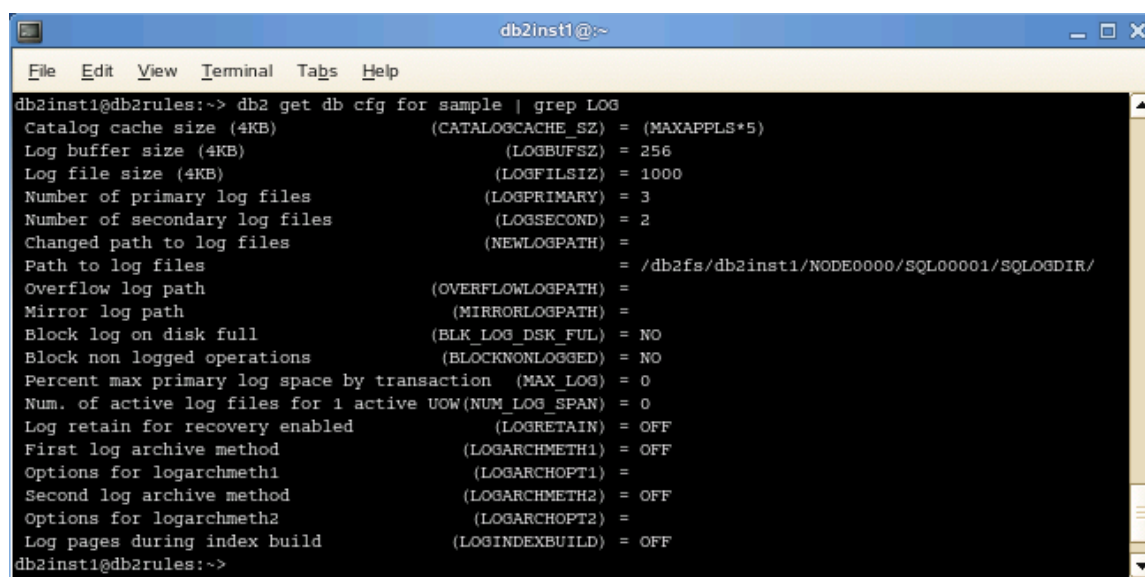
A transaction is a logical unit of work. Every transaction performed by DB2 is first written to the log and is then performed against the data. DB2 relies on these log files for backup and recovery.

Before we can go into the different types of DB2 logging, we first have to understand some logging parameters.

3.1 Logging Parameters

To see the database configuration that is related to logging, run the following command:

```
db2 get db cfg for sample | grep LOG
```



```
db2inst1@db2rules:~> db2 get db cfg for sample | grep LOG
Catalog cache size (4KB)          (CATALOGCACHE_SZ) = (MAXAPPLS*5)
Log buffer size (4KB)             (LOGBUFSZ) = 256
Log file size (4KB)              (LOGFILSIZ) = 1000
Number of primary log files       (LOGPRIMARY) = 3
Number of secondary log files     (LOGSECOND) = 2
Changed path to log files        (NEWLOGPATH) =
Path to log files                 = /db2fs/db2inst1/NODE0000/SQL00001/SQLLOGDIR/
Overflow log path                 (OVERFLOWLOGPATH) =
Mirror log path                  (MIRRORLOGPATH) =
Block log on disk full           (BLK_LOG_DSK_FUL) = NO
Block non logged operations       (BLOCKNONLOGGED) = NO
Percent max primary log space by transaction (MAX_LOG) = 0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0
Log retain for recovery enabled   (LOGRETAIN) = OFF
First log archive method         (LOGARCHMETH1) = OFF
Options for logarchmeth1        (LOGARCHOPT1) =
Second log archive method        (LOGARCHMETH2) = OFF
Options for logarchmeth2        (LOGARCHOPT2) =
Log pages during index build     (LOGINDEXBUILD) = OFF
db2inst1@db2rules:~>
```

3.1.1 LOGFILSIZ

It is the size of each transactional log file measured in 4KB pages. The default size is 1000 pages or 4 MB, which implies that it can hold up to 4 MB of transactional data. You can configure it to be larger if it is going to be a high-transaction OLTP type of environment. In an OLTP environment, small-sized log files would fill up very quickly and new log files would have to be created frequently.

3.1.2 LOGPRIMARY and LOGSECOND

LOGPRIMARY is the number of primary log files. At any given time, there might be some uncommitted transactions in the database that make up the active log space. Active log space refers to the total sum of log space taken up by uncommitted transactions. By default it is 3; therefore if you have 3 log files worth of uncommitted transactions, any new transactions would start utilizing the secondary log files.

LOGSECOND is the number of secondary log files. These are allocated only when a transaction exhausts all the space configured for the primary log, to accommodate spikes in transactional activity. Once the transactions using the secondary log files commit or roll back, DB2 returns to using primary log. You can have this configured. By default LOGSECOND is 2, meaning if primary log files filled up with uncommitted transactions, 2 more log files will be allocated temporarily to handle the spike. If all the primary and secondary log files have been used, then an error will be returned:

SQL0964C The transaction log for the database is full.

Set LOGPRIMARY to 5 and LOGSECOND to 3; issue the following command from the terminal window:

```
db2 update database configuration for sample using LOGPRIMARY 5
db2 update database configuration for sample using LOGSECOND 3
```

A warning message may be returned:

SQL1363W One or more of the parameters submitted for immediate modification were not changed dynamically. For these configuration parameters, all applications must disconnect from this database before the changes become effective.

In order for the change of configuration to take effect, simply disconnect and reconnect to it, since this is the only connection to the database at this moment.

```
db2 terminate
db2 connect to sample
```

3.1.3 LOGBUFSZ

All log records are written in memory before getting flushed to disk. LOGBUFSZ specifies the size of this area in memory. The default of 8 4KB pages is small for most scenarios. This parameter is critical for OLTP performance. Set the LOGBUFSZ to 256, which is a good starting number. In a real environment, take an OLTP workload and benchmark with higher LOGBUFSZ to find the optimal value.

3.2 Types of Logging

DB2 databases support two different logging modes: Circular and Archival.

3.2.1 Circular Logging

This is DB2's default logging technique for a newly created database. It uses primary log files in rotation up to the number of log files indicated by the LOGPRIMARY parameter. If a long-running transaction exhausts all the primary log files before completing, the transaction spills over to the secondary log files. When the work is committed, DB2 returns to the first log file and continues in a circular fashion.

Roll-forward recovery is not possible with this logging method because log files are not kept as they are constantly being overwritten. Only crash recovery and version recovery are available. If a database is using circular logging, the database can be backed up only through an offline backup.

To enable circular logging, set both LOGARCHMETH1 and LOGARCHMETH2 database configuration parameters to OFF.

3.2.2 Archival Logging

In archival logging, all log files are kept; they are never overwritten. To have online backups and the ability to perform roll forward recovery, the database needs to be enabled for archival logging.

To enable archival logging, you will need to specify the value of LOGARCHMETH1 to something other than OFF. If both LOGARCHMETH1 and LOGARCHMETH2 have been specified, then archive logs are archived twice.

Infinite logging is a variation of archival logging where LOGARCHMETH2 is set to -1. With this type of logging, secondary log files are allocated until the unit of work commits.

1. We will now change the logging method to archival logging and set the archival location:

```
mkdir /home/db2inst1/logarch
db2 update db cfg for sample using LOGARCHMETH1
disk:/home/db2inst1/logarch
```

2. Terminate the connection to the database and reconnect to the sample database:

```
db2 terminate
db2 connect to sample
```

However, when you try to reconnect to the sample database, you will receive the following error:

SQL1116N A connection to or activation of database "SAMPLE" cannot be made because of BACKUP PENDING. SQLSTATE=57019

This message is received because archival logging has just been enabled for this database so it is put into backup pending state. Recall that once archival logging is enabled for the database, roll forward recoveries can be performed. However, roll forward recovery can only be performed once a backup image has been restored and database is placed in Roll Forward Pending status. Therefore, a full database backup must be made before the database can be used.

3. Create a directory to store the backup and take a full database backup by issuing the following command:

```
mkdir /home/db2inst1/backups
db2 backup database sample to /home/db2inst1/backups
```

If no error has occurred, you will see a similar message as the following but with a different timestamp:

Backup successful. The timestamp for this backup image is: 20100509163937

When a backup image is created, the timestamp at which the backup image is created is returned in the format of **yyyymmddhhmmss**. This is useful information because the **Restore** utility uses this timestamp to differentiate between multiple available backup images.

Write down the timestamp returned by your backup command, it will be referred to as T1 in following exercises.

```
T1:
```

4. Try to connect to the database again. This time it should succeed.

```
db2 connect to sample
```

4. Recovery Scenario

In this section of the lab, we will explore various scenarios in which DB2 Recovery utility can be used to recover from failure.

4.1 Scenario 1 - Entire database is accidentally dropped or becomes corrupted

If a database was accidentally dropped or is corrupted, you can recover the database by restoring a full backup

In this example, we will restore from the offline backup image taken at the end of Exercise 3.2.2. If you had not noted down the timestamp (T1) at which the backup was taken, you can always check the Recovery history file to find the backup time stamp by issuing the following command:

```
db2 list history backup all for database sample
```

The timestamp is indicated within the circle in the screenshot below:

```

db2inst1@~
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 list history backup all for database sample

List History File for sample

Number of matching file entries = 1

Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log Backup ID
-----
B D 20100509163937.01 F D S0000000.LOG S0000000.LOG
-----

Contains 4 tablespace(s):

00001 SYSCATSPACE
00002 USERSPACE1
00003 IBMDB2SAMPLEREL
00004 IBMDB2SAMPLEXML
-----

Comment: DB2 BACKUP SAMPLE OFFLINE
Start Time: 20100509163937
End Time: 20100509164022
Status: A
-----

EID: 3 Location: /home/db2inst1/backups

```


1. To simulate this scenario, disconnect and drop the database sample:

```
db2 force applications all
db2 drop database sample
```

If you try to connect to the sample database now, you will receive the following error

```
db2 connect to sample
```

SQL1013N Database alias name or database name "sample" could not found. SQLSTATE=43705

2. To recover from this failure, you can restore a previously created full database backup.

Restore the database backup image that was created earlier in the previous exercise. You will need to substitute the timestamp T1 noted earlier into the command:

```
db2 restore database sample from /home/db2inst1/backups taken at <T1>
without rolling forward
```

Note that there is the **without rolling forward** clause in the restore command. Since restore is from an offline backup, it is not mandatory to do a roll forward after the restore. This is useful when a roll forward is not needed and restore can finish in just one step.

After restore finishes, you should be able to connect to the sample database without having to do a roll forward explicitly.

4.2 Scenario 2 - Database Roll forward to a Point in Time

Roll forward is the process of applying transaction log files after a restore has been performed. For example, the last backup was taken Sunday, and the database was lost on the following Tuesday. Once the backup from Sunday is restored, transactions in log files need to be applied in order to recover transactions that were executed after the backup was taken. This is achieved by rolling forward to **END OF LOGS**.

There might be a situation where it is not desired to apply all the transactions. For example, a large set of records are deleted from the database mistakenly by the user. In such a case, in order to recover all the deleted records, rolling forward to a **POINT IN TIME** before the deletions took place would be more appropriate.

1. To simulate this scenario, we will delete some rows from tables.

Before we began, check the number of rows in the original STAFF table within the sample database:

```
db2 connect to sample
db2 "select count(*) from staff"
```

The number of rows in the STAFF table should be 35.

Now run the following commands to delete some of the data from the STAFF table:

```
db2 "delete from staff where dept=10"
```

Check the count of the STAFF table after the delete statement:

```
db2 "select count(*) from staff"
```

There should now be 31 rows in the STAFF table.

2. We will run another delete statement on the EMPLOYEE table. However, imagine that these rows were deleted accidentally.

Run the "date" command and note the timestamp before we "accidentally" issue a delete statement.

```
date +%F-%H.%M.%S
```

This timestamp will be referred to as T2, write it down as a record as this is needed for the point in time recovery:

```
T2:
```

Now check the number of rows in the original EMPLOYEE table:

```
db2 "select count(*) from employee"
```

The number of rows in the EMPLOYEE table should be 42.

Now we will accidentally delete some data from the EMPLOYEE table:

```
db2 "delete from employee where edlevel=12"
```

Check the count of the EMPLOYEE table after the delete statement:

```
db2 "select count(*) from employee"
```

There should now be 39 rows in the EMPLOYEE table.

3. The rows that you have just deleted from the EMPLOYEE table were not supposed to be removed. If we restore the database to the last full backup, then the deletion of rows to the STAFF table will also be undone. In this case, we can recover to the point in time just before the delete statement was issued against the EMPLOYEE, which in our case is T2.

- Restore the database to the last backup image which we have taken from exercise 3.2.2 at T1:

```
db2 restore database sample from /home/db2inst1/backups taken at <T1>
without prompting
```

- Now that the database is restored, roll forward to a point in time before the delete on table EMPLOYEE was issued which is T2.

```
db2 rollforward db sample to <T2> using local time
```

Note that the timestamp for roll forward has to be provided in this format: **yyyy-mm-dd-hh.mm.ss**.

- Lastly, take the database out of the roll forward pending status by executing:

```
db2 rollforward database sample stop
```

- Connect to the sample database and check the number of rows of the STAFF table and the EMPLOYEE table.

```
db2 connect to sample
db2 "select count(*) from staff"
db2 "select count(*) from employee"
```

You will notice that the number of rows returned from the STAFF table is 31 and the number of rows in the EMPLOYEE table is 42.

The “accidentally” deleted rows from the EMPLOYEE table have been recovered by performing a point in time recovery. Roll forward was done up to a time before the delete statement was issued. The delete statement was issued after this point in time; therefore, it was not replayed.

If an END OF LOGS roll forward was done in this case, it would have also replayed the delete statement of the EMPLOYEE table, thereby deleting the rows again. The END OF LOGS option is useful when the database has been lost, and a recovery is needed through all available logs to ensure that all transactions have been recovered.

4.3 Scenario 3 – Incremental Backup and Restore

4.3.1 Incremental Backup

As database sizes grow larger, it can be quite costly to run full backups, both in terms of storage for the backup images and time required to execute the backups. This is where incremental backups come in. They allow the user to only backup the changes that have

been made since the last backup, instead of having to backup the entire database every time.

In order to use incremental backups, the database has to be enabled for it. This is done by turning the **TRACKMOD** database configuration parameter on. When TRACKMOD is turned on, the database keeps track of table spaces that have been modified. When an incremental backup command is issued, it will skip the table spaces that have not been modified since the last backup.

1. Turn the TRACKMOD database configuration parameter to ON:

```
db2 connect to sample
db2 update db cfg for sample using TRACKMOD ON
```

A warning message will be returned:

SQL1363W One or more of the parameters submitted for immediate modification were not changed dynamically. For these configuration parameters, all applications must disconnect from this database before the changes become effective.

2. In order for the change of configuration to take effect, reconnect to it.

```
db2 terminate
db2 connect to sample
```

3. Incremental backups require a full backup to act as a reference point for incremental changes. Create a backup of the database using the online mode:

```
db2 backup database sample online to /home/db2inst1/backups
```

Write down the timestamp of this backup, it will be referred to as T3.

T3:

4. Make some changes to the STAFF table by decreasing the salary of everyone:

```
db2 connect to sample
db2 "update staff set salary=salary*0.9"
```

5. After the database has enabled incremental backups by modifying TRACKMOD to ON and after creating a full backup of the database, an incremental backup can be now taken to just include the changes made.

```
db2 backup db sample incremental to /home/db2inst1/backups
```

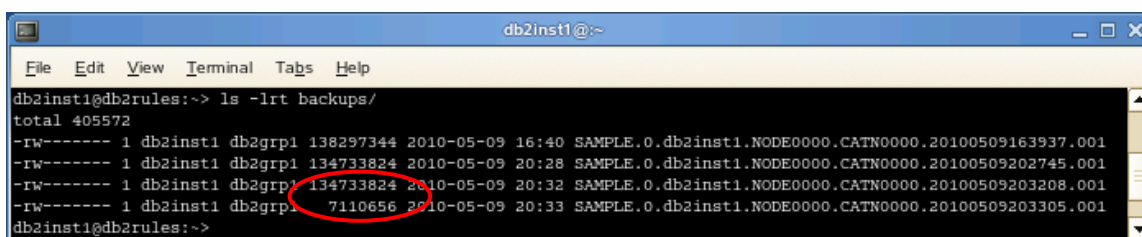
Note down the timestamp at which the incremental backup is created. This will be referred to as T4.

T4:

6. Compare the size of the full backup and the incremental backup images. At the command prompt, run the following command to check the size:

```
ls -lrt /home/db2inst1/backups
```

The circle indicates the size of the last two backup images. Notice the size of the last image (the incremental backup image) is much smaller than the image above it (the full backup image). This is because the incremental image contains only the changes since last full backup. Any table space that was not modified since the last full backup will not be included in the incremental database backup.



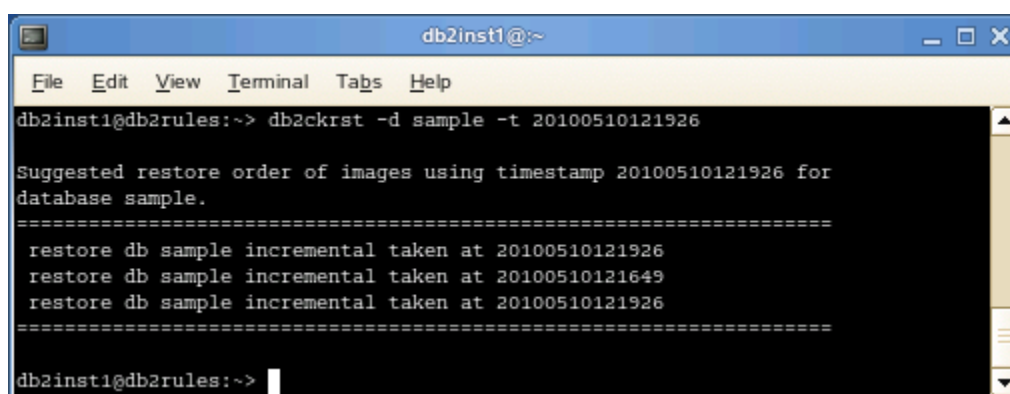
```
db2inst1@db2rules:~> ls -lrt backups/
total 405572
-rw----- 1 db2inst1 db2grp1 138297344 2010-05-09 16:40 SAMPLE.0.db2inst1.NODE0000.CATN0000.20100509163937.001
-rw----- 1 db2inst1 db2grp1 134733824 2010-05-09 20:28 SAMPLE.0.db2inst1.NODE0000.CATN0000.20100509202745.001
-rw----- 1 db2inst1 db2grp1 134733824 2010-05-09 20:32 SAMPLE.0.db2inst1.NODE0000.CATN0000.20100509203208.001
-rw----- 1 db2inst1 db2grp1 7110656 2010-05-09 20:33 SAMPLE.0.db2inst1.NODE0000.CATN0000.20100509203305.001
db2inst1@db2rules:~>
```

4.3.2 Incremental Restore

When restoring from incremental backups, the right sequence of full, incremental and incremental delta backups have to be applied. This can become very complex very quickly in a real environment. For this reason, there is an **AUTOMATIC** option available with the restore command such that DB2 figures out the right sequence for applying backups and then applies them. There is also a **MANUAL** option available, but the **AUTOMATIC** option is highly recommended.

The **db2ckrst** utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore.

```
db2ckrst -d sample -t <T4>
```



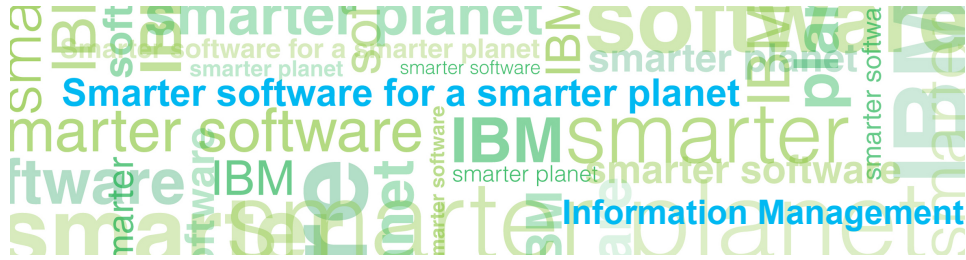
```
db2inst1@db2rules:~> db2ckrst -d sample -t 20100510121926
Suggested restore order of images using timestamp 20100510121926 for
database sample.
=====
restore db sample incremental taken at 20100510121926
restore db sample incremental taken at 20100510121649
restore db sample incremental taken at 20100510121926
=====
db2inst1@db2rules:~>
```

This output shows that last incremental image will be read first to get the control and header information only. Then the database will be restored from the full backup image. Lastly, the incremental image will be read again, this time applying the data in the image.

Issue the following command from the command line to restore SAMPLE database to the last incremental backup image:

```
db2 "restore db sample incremental automatic from  
/home/db2inst1/backups taken at <T4>"
```

DB2® pureXML



© 2010 IBM Corporation



Agenda

- Overview of XML
- pureXML in DB2
- XML Data Movement in DB2
- XQuery and SQL/XML
- XML Indexes in DB2
- Application Development

What is XML?

- **eXtensible Markup Language**
 - XML is a language designed to describe data
- A **hierarchical** data model

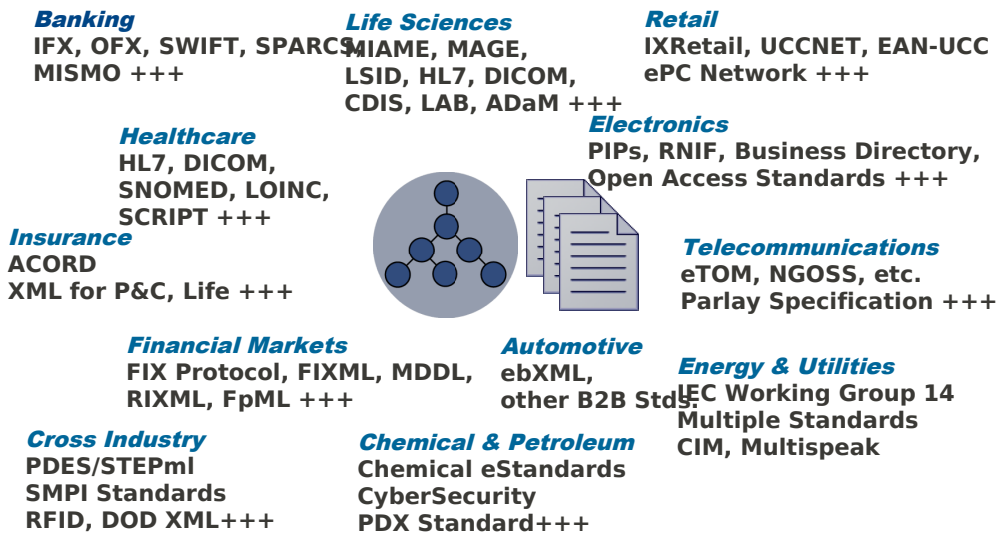
```
<book>
  <authors>
    <author id="47">John Doe</author>
    <author id="58">Peter Pan</author>
  </authors>
  <title>Database systems</title>
</book>
```



3

© 2010 IBM Corporation

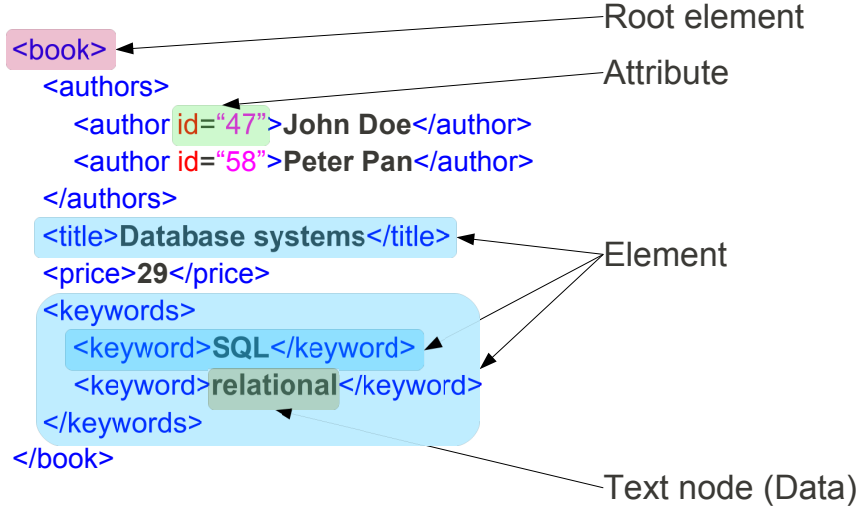
Who Uses XML?



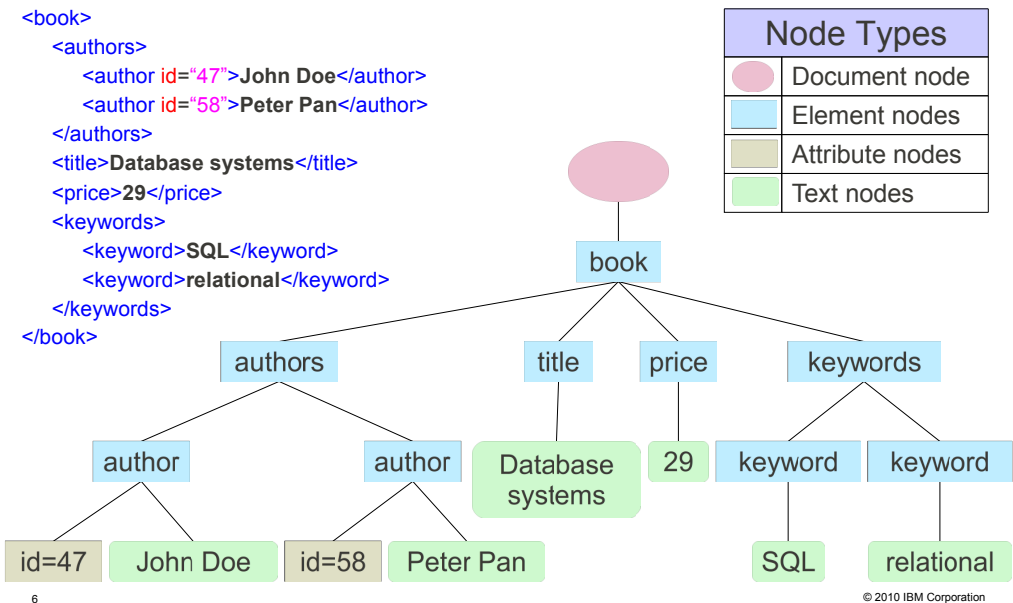
4

© 2010 IBM Corporation

XML Document Components



The XML Data Model: Node Types



Well-Formed Versus Valid XML Documents

- **A well-formed XML document is a document that follows basic rules:**
 - 1) It must have one and only one root element
 - 2) Each element begins with a start tag and ends with an end tag
 - 3) An element can contain other elements, attributes, or text nodes
 - 4) Attribute values must be enclosed in double quotes. Text nodes, on the other hand, should not.

(i.e. it can be parsed by an XML parser without error)

- **A valid XML document is BOTH:**
 - 1) A well-formed XML document
 - 2) A document compliant with the rules defined in an XML schema document or a Document Type Definition (DTD) document.

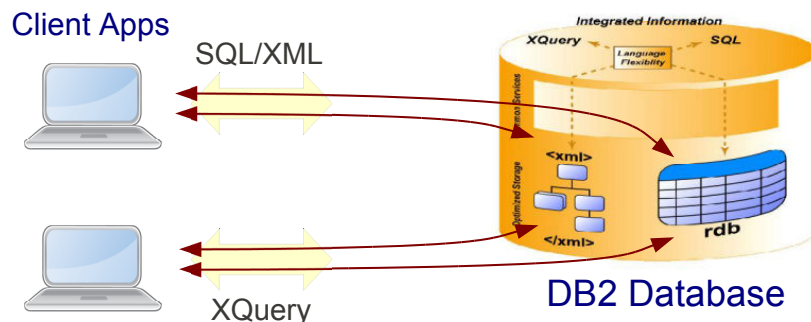
XML Parsers can optionally perform “validation”

7

© 2010 IBM Corporation

DB2 pureXML

- **Relational columns are stored in relational format (tables)**
- **Native XML Storage**
 - XML documents are stored in their original hierarchical format
 - **No XML parsing for query evaluation!**
- **XML capabilities in all DB2 components**
- **XML data can be manipulated using SQL or XQuery**

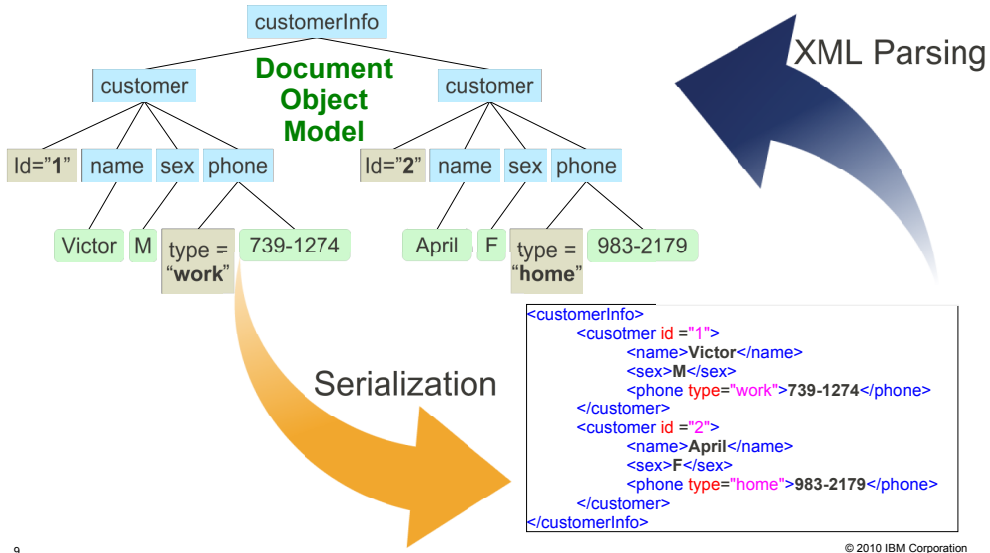


8

© 2010 IBM Corporation

Native XML Storage

- Documents are stored in parsed representation



9

© 2010 IBM Corporation

Relational Versus Hierarchical (XML) Model

Relational	Hierarchical (XML)
Relational data is flat	XML data is nested .
Relational model is set oriented . Sets are unordered.	XML retrieves sequences (the order matters)
Relational data is structured .	XML data is semi-structured .
Relational data has a strong schema , unlikely to change often.	XML data has a flexible schema , appropriate for constant changes.
Use NULL for an unknown state.	NULLS don't exist . Don't add any XML element.
Based on the ANSI/ISO industry standards.	Based on the W3C industry standards.

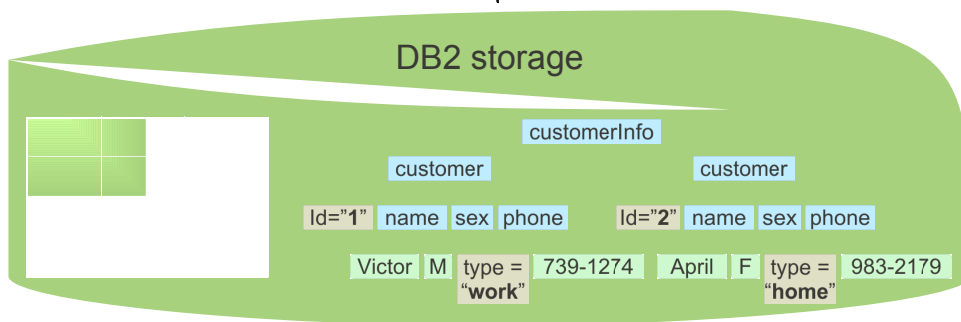
10

© 2010 IBM Corporation

PureXML Storage in DB2: XML Data Type

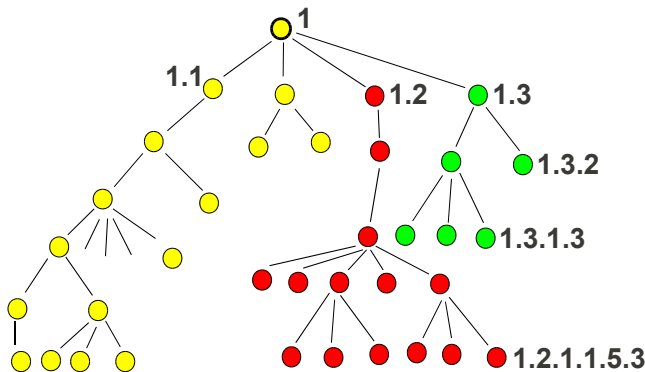
```
CREATE TABLE dept (deptID VARCHAR(30), ..., custDoc XML)
```

deptID	...	custDoc
A001	...	
...



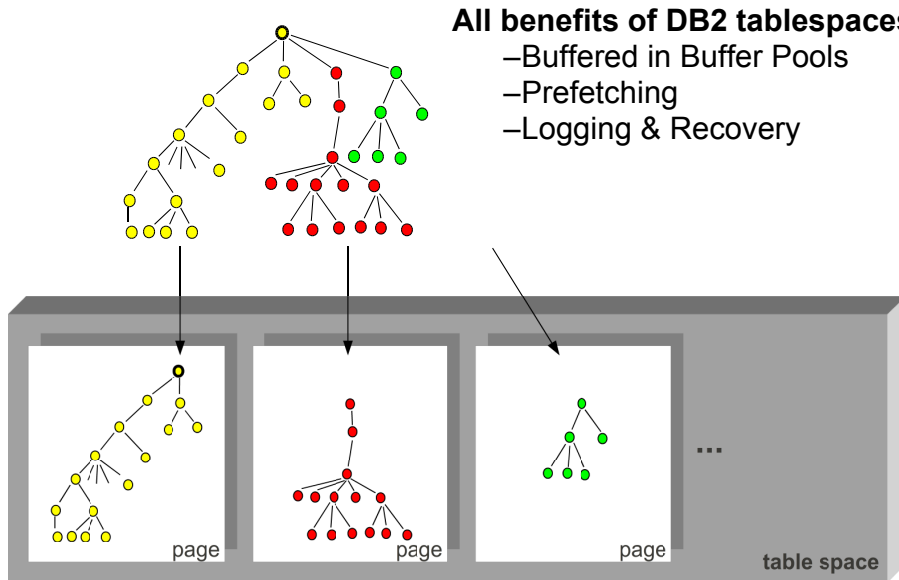
XML Node Storage Layout

- Node hierarchy of an XML document is stored on DB2 pages
- Documents that don't fit on 1 page are split into **pages/regions**
- No architectural limit for size of XML documents
- NodeIDs are used to identify individual nodes



Document split into 3 regions, stored on 3 pages..

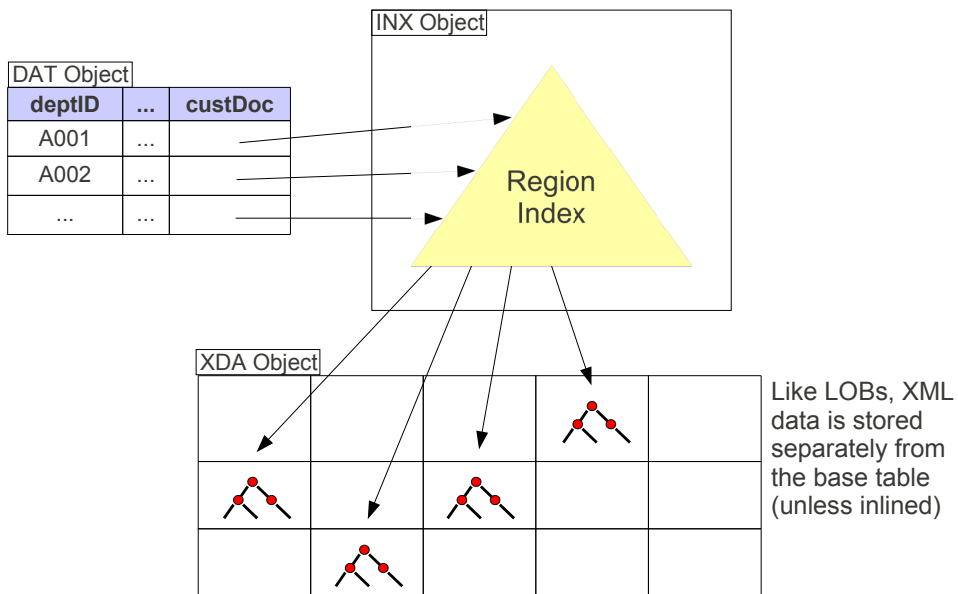
XML Data: As Trees on DB2 Pages



13

© 2010 IBM Corporation

XML Storage: Internal Objects and Their Relationship



14

© 2010 IBM Corporation

How to Get Data In?

- **Implicit XML parsing:**

- Inserting data of XML data type into a column

```
INSERT INTO dept VALUES
('PR27', ..., '<dept>...<emp>...</emp>...</dept>')
```

- **Explicit XMLPARSE**

- Transform XML value from serialized (text) form into internal representation.

- Tell system how to treat whitespaces (strip/preserve)

- Default is 'Strip WHITESPACE'

```
INSERT INTO dept VALUES ('PR27', xmlparse(document '<a>...</a>'));
INSERT INTO dept VALUES ('PR27',
xmlparse(document '<a>...</a>' preserve whitespace));
```

Deleting XML Data

- **DELETE**

- Will delete every XML document for a row

```
DELETE FROM dept WHERE deptID='A001'
```

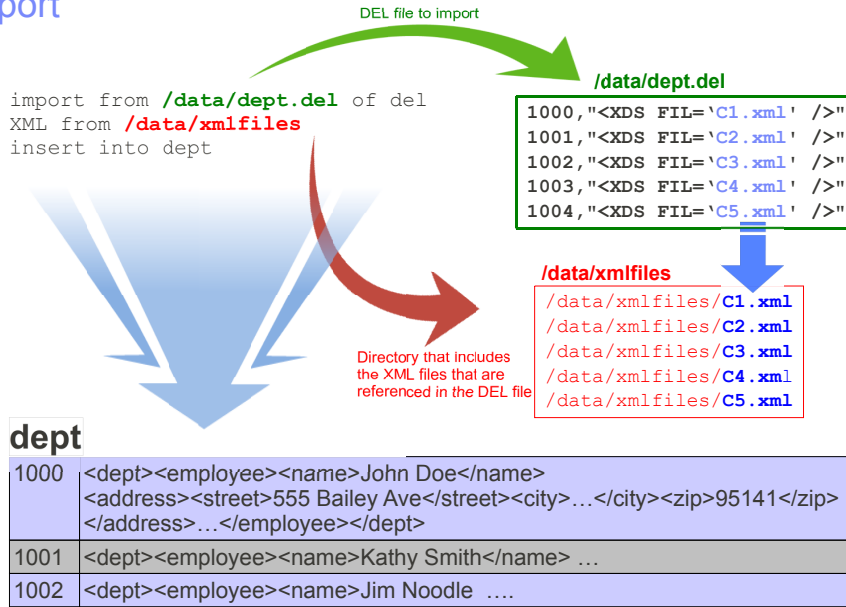
- You can also delete based on the XML content

```
DELETE FROM dept WHERE
XMLEXISTS ('$d//phone[type="Home"]'
passing INFO as "d")
```

- **Note: Setting an XML column to NULL deletes the XML document**

```
UPDATE dept SET custDoc = NULL WHERE deptID='A001'
```

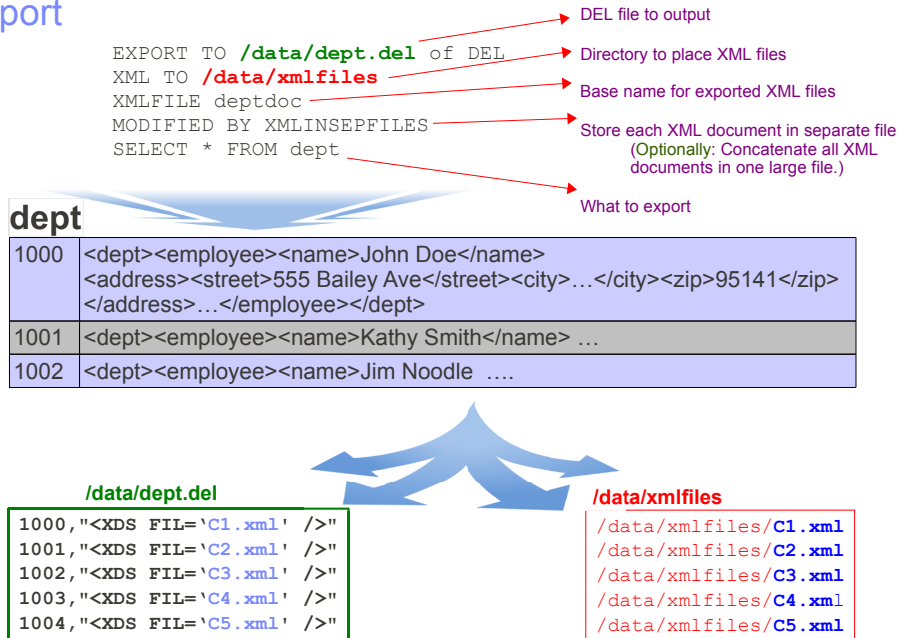
Import



17

© 2010 IBM Corporation

Export



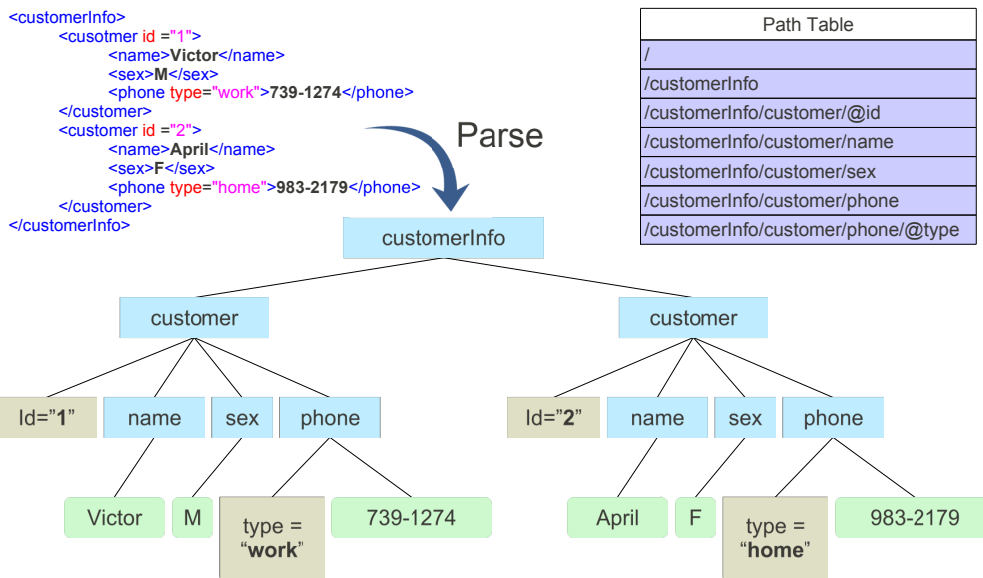
18

© 2010 IBM Corporation

SQL/XML and XQuery

- **DB2 Supports two query languages:**
 - XQuery
 - SQL/XML
- **XPath**
 - Cornerstone for both XQuery and SQL/XML standard
 - Provides ability to navigate within XML documents
- **XQuery**
 - Two important functions to access the database:
 - db2-fn:sqlquery
 - db2-fn:xmlcolumn
 - Results returned as a sequence of items
- **SQL/XML**
 - Provides functions to work with both XML and relation data at the same time.

XPath



Some Common XPath Expressions

```
<customerInfo>
  <customer id = "1">
    <name>Victor</name>
    <sex>M</sex>
    <phone type="work">739-1274</phone>
  </customer>
  <customer id = "2">
    <name>April</name>
    <sex>F</sex>
    <phone type="home">983-2179</phone>
  </customer>
</customerInfo>
```

/	Selects from the root node.
//	Selects nodes in the document from the current node that match the select.
text()	Specifies the text node under an element.
@	Specifies an attribute.
*	Matches any element node.
@*	Matches any attribute node.
[...]	Predicates

XPath Expression	Result Description	Result
/customerInfo/*/phone/text()	Selects the text node under the phone element of customerInfo	739-1274 983-2179
/customerInfo//phone/@type	Selects the type attribute under the phone element of customerInfo	work home
/customerInfo/customer[1]/phone/text()	Selects the phone element text node under the first customer of customerInfo	739-1274
/customerInfo//phone[@type='home']	Selects all phone elements under customerInfo which has an attribute named type with a value of 'home'	<phone type="home"> 983-2179 </phone>

21

© 2010 IBM Corporation

Introduction to XQuery

- Unlike relational data (which is predictable and has a regular structure), XML data is:

- Often **unpredictable**
- Highly variable
- Sparse
- Self-describing

- You may need XML queries to perform the following operations:

- Search XML data for objects that are at unknown levels of the hierarchy
- Perform structural transformations on the data
- Return results that have mixed types

22

© 2010 IBM Corporation

DB2 XQuery Functions

- To obtain XML data from a DB2 database with XQuery

- `db2-fn:xmlcolumn (xml-column-name)`

- Input argument is a string literal that identifies an XML column in a table, case sensitive

xquery

```
db2-fn:xmlcolumn ("CUSTOMER.INFO")/customerinfo
```

- Retrieves an entire XML column as a sequence of XML values

- `db2-fn:sqlquery (full-select-sql-statement)`

- Input argument is interpreted as an SQL statement and parsed by the SQL parser

- SQL statement needs to return a single XML column

xquery

```
db2-fn:sqlquery ('SELECT INFO
FROM CUSTOMER WHERE CID=6')/customerinfo
```

- Returns an XML sequence that results from the full select

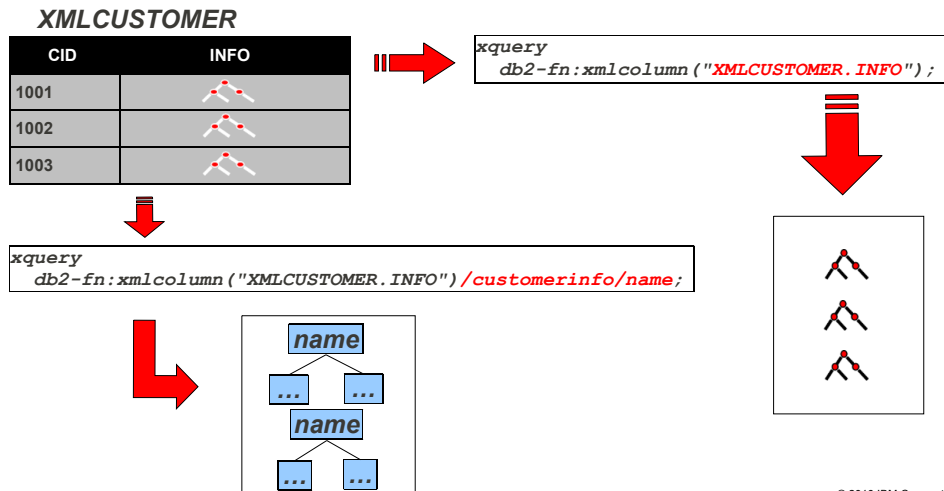
23

© 2010 IBM Corporation

XQuery: Retrieving XML Data From a Column

- **db2-fn:xmlcolumn**

- Retrieve all XML documents from an XML column, then process them with an XQuery expression.



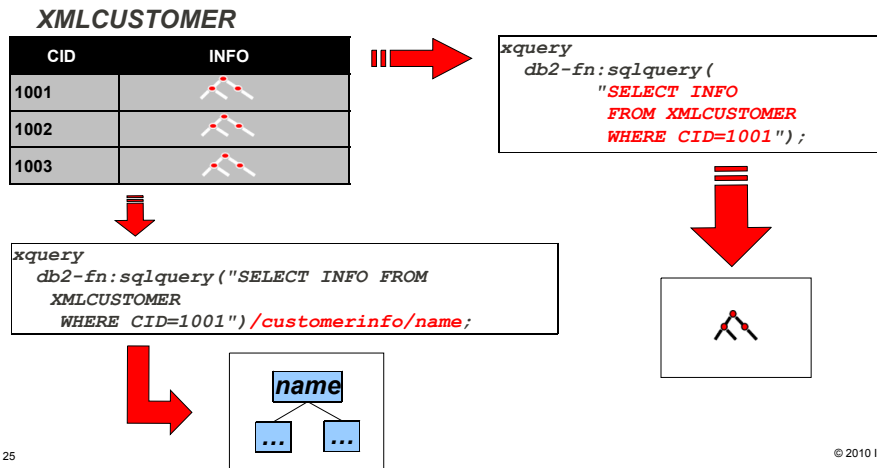
24

© 2010 IBM Corporation

XQuery: Retrieving XML Based on a SQL Query

- **db2-fn:sqlquery**

- Retrieve and XML document using SQL, then process it with an XQuery expression
- Allows filtering based on relational data



25

© 2010 IBM Corporation

SQL/XML Functions

- **XQuery can be invoked from SQL**

- XMLQUERY** ()
- XMLTABLE** ()
- XMLEXISTS** ()

- **By executing XQuery expressions from within the SQL context, you can:**

- Operate on parts of stored XML documents instead of entire XML documents
- Enable XML data to participate in SQL queries
- Operate on both relational and XML data
- Apply further SQL processing to the returned XML values

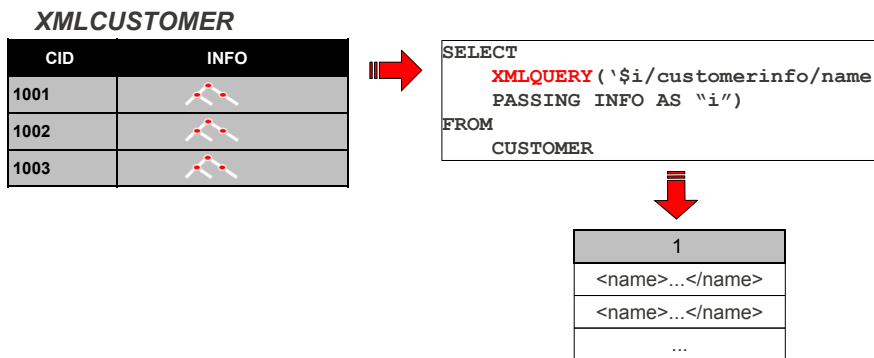
26

© 2010 IBM Corporation

SQL/XQuery: XML Data for SQL Developers

▪ XMLQUERY

- Scalar function, applied once to each qualifying document
- Evaluates an XPath (or XQuery) expression
- Input arguments can be passed into the XQuery (e.g. column names, constants, parameter markers)
- Returns a sequence of 0, 1 or multiple items from each document



27

© 2010 IBM Corporation

SQL/XQuery: XML Data for SQL Developers

```
SELECT
  XMLQUERY ('$i/customerinfo/name'
    PASSING INFO AS "i")
FROM CUSTOMER
```

- **SELECT iterates over all rows in the customer table**
- **For each row, "XMLQUERY" is invoked**
 - The "passing" clause binds the variable "\$i" to the value of the "INFO" column of the current row
 - The XQuery expression is executed
 - XMLQUERY returns the result of the XQuery expression, a value of type XML

28




© 2010 IBM Corporation

SQL/XQuery: XML Data for SQL Developers

▪ XMLTABLE

–Creates a temporary SQL table using XML data

XMLCUSTOMER

CID	INFO
1001	
1002	
1003	

```
SELECT T.*
FROM XMLTABLE (
  'db2-fn:xmlcolumn("XMLCUSTOMER.INFO")/customerinfo'
  COLUMNS "NAME" VARCHAR (20) PATH 'name',
           "STREET" VARCHAR (20) PATH 'addr/street',
           "CITY" VARCHAR (20) PATH 'addr/city'
) AS T
```

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <prov-state>Alberta</prov-state>
    <pcode-zip>M1T 2A9</pcode-zip>
  </addr>
  <phone type="work">
    963-289-4136
  </phone>
</customerinfo>
```

NAME	STREET	CITY
Amir Malik	Young	Toronto
John Smith	Fourth	Calgary
...




© 2010 IBM Corporation

SQL/XQuery: XML Data for SQL Developers

▪ XMLEXISTS


–Predicate that tests if an XQuery expression returns a sequence

XMLCUSTOMER

CID	INFO
1001	
1002	
1003	

```
SELECT CID, INFO
FROM XMLCUSTOMER WHERE
XMLEXISTS (
  '$d/customerinfo[name = "John Smith"]'
  passing INFO as "d")
```

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <prov-state>Alberta</prov-state>
    <pcode-zip>M1T 2A9</pcode-zip>
  </addr>
  <phone type="work">
    963-289-4136
  </phone>
</customerinfo>
```

CID	INFO
1003	

© 2010 IBM Corporation

XML Indexes

- **An index over XML data can be used to improve the efficiency of queries on XML documents.**
 - Index entries will provide access to nodes within the document by creating index keys based on XML pattern expressions.
- **Like relational data they may have some cost.**
 - Performance for INSERT, UPDATE and DELETE
 - Space needed to store the indexes

Regular Indexes	Indexes for XML
Based on columns	Based on XML pattern expressions
1 or more columns	Only 1 XML column
1 row → 1 index key	All nodes that satisfy the XML pattern: 1 document → 0, 1 or more index keys
B-Tree	B-Tree

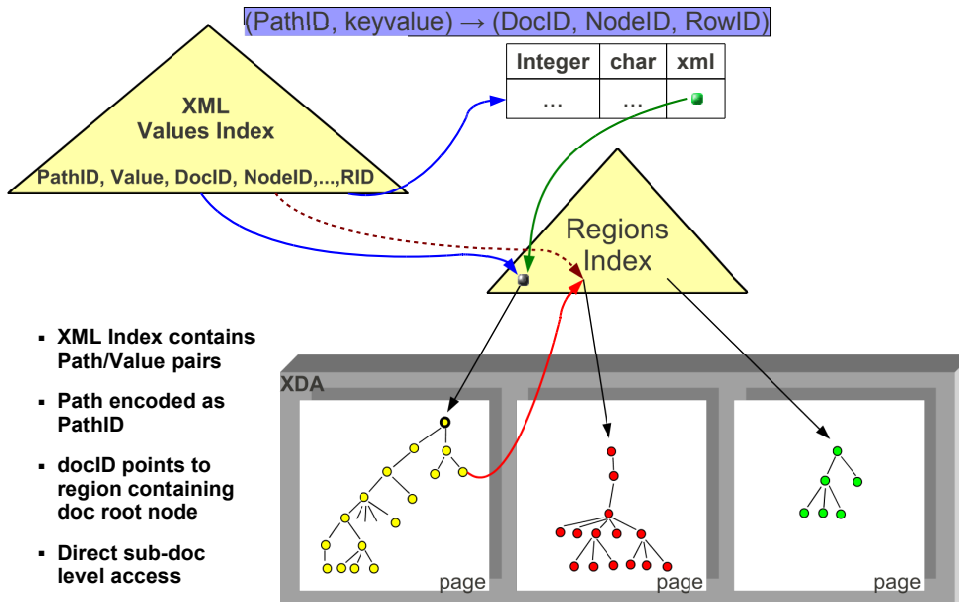
```
CREATE INDEX IDX1 ON
  TB1 (XMLDOC)
  GENERATE KEY USING XMLPATTERN
  '/company/emp/salary'
  AS SQL DOUBLE;

CREATE INDEX IDX2 ON
  TB1 (XMLDOC)
  GENERATE KEY USING XMLPATTERN
  '//@id' AS SQL VARCHAR(20);
```

31

© 2010 IBM Corporation

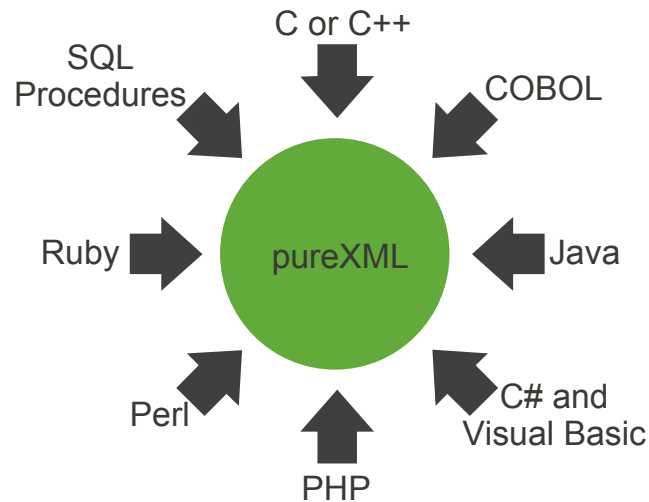
XML Indexes: Under the Covers



32

© 2010 IBM Corporation

Development Support for XML Data



33

© 2010 IBM Corporation

XML – Conclusion

- **Native XML hierarchical storage**
 - No shredding, no CLOBs, no BLOBs required
 - Optimized for XPATH and XQuery (LUW Only) processing
- **High performance**
 - Superior indexing technology
 - No parsing of XML data at query runtime
- **Fully integrated XML and relational processing**
 - Seamlessly query various types of data at once
 - No internal translation of XQuery into SQL

34

© 2010 IBM Corporation



IBM DB2[®] 9.7

**DB2 pureXML -
Storing XML Data
Made Easy
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

1. XML BASICS AND INTRODUCTION.....	3
2. SETUP AND CREATION OF XML TABLES.....	3
2.1 ENVIRONMENT SETUP REQUIREMENTS	3
2.2 LOGIN TO THE VIRTUAL MACHINE.....	3
2.3 START DB2 SERVER AND ADMINISTRATION SERVER.....	4
3. DATABASE CREATION AND CONTROL CENTER	4
4. XQUERY	6
4.1 USING XML QUERIES	6
5. SQL/XML	8
6. XMLTABLE FUNCTION	9
SUGGESTED READING	10

1. XML Basics and Introduction

It is sometimes desirable for users accustomed to SQL to use or extend SQL statements to query XML data. Many existing relational applications are augmented with XML data. Therefore, it is not uncommon to extend existing SQL for XML capabilities. Since XML has been a data type since DB2 9, SQL/XML functions makes it much easier for queries of XML with relational data. SQL/XML is also useful since it allows us to retrieve and transform relational data into XML format, producing a single column of XML data type that serves as input to XQuery. As we will see throughout the lab, XQuery provides a way to extract and manipulate data from XML documents or other XML structured data sources.

Since the introduction of pureXML technology in DB2 9, many users have already learned how to manage XML data with DB2. This lab refreshes some simple SQL/XML and XQuery functionality and concentrates on some advanced SQL/XML functionality as well as the new transform function from the XQuery Update Facility.

This lab works on the DB2 sample database, allowing these tasks to be completed from any machine.

2. Setup and Creation of XML Tables

2.1 Environment Setup Requirements

To complete this lab you will need the following:

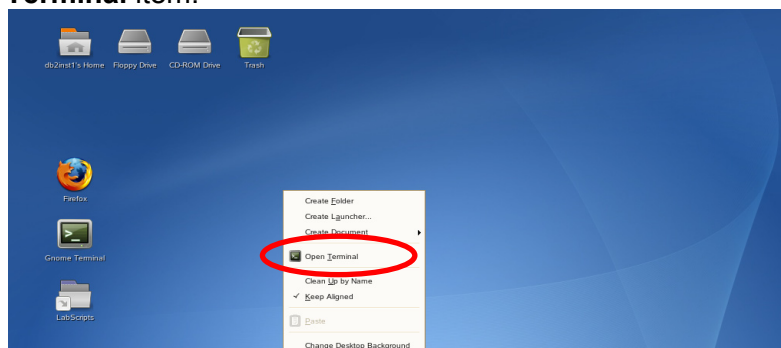
1. DB2 Academic Workshop VMware® image
2. VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in the **VMware Basics and Introduction** module.

2.2 Login to the Virtual Machine

1. Login to the VMware virtual machine using the following information:
User: **db2inst1**
Password: **password**
2. In the command window enter **startx** to bring up the graphical environment.

3. Open a terminal window by right-clicking on the **Desktop** area and choosing the **Open Terminal** item.



2.3 Start DB2 Server and Administration Server

1. Start up **DB2 Server and Administration Server** by typing the following commands in the terminal window in order:

```
db2start
su - dasusr1
db2admin start
exit
```

3. Database Creation and Control Center

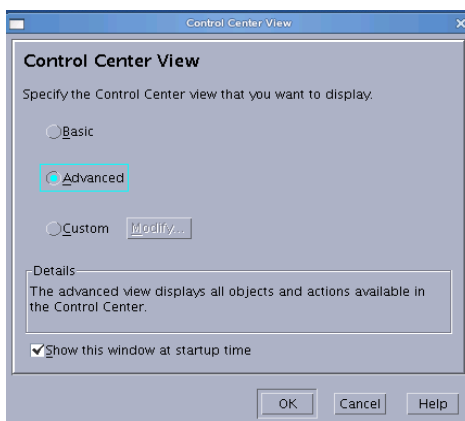
1. Open a terminal window by right-clicking on the **Desktop** area and select the **Open Terminal** item.
2. Execute the command below to create a sample database named "purexml" that will be populated with XML data.

```
db2sampl -name purexml -xml
```

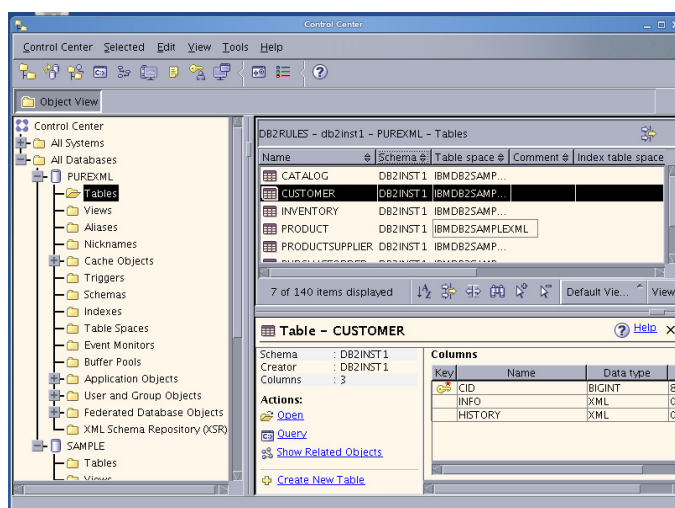
3. We will use the Control Center to work with the PUREXML database. Start the **DB2 Control Center** by typing the following in the command window:

```
db2cc
```

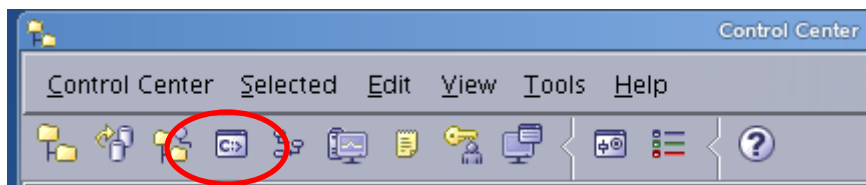
4. In the Control Center View, select the "**Advanced**" display mode to have access to all the options. Then click "**OK**" to continue.



5. A screen similar to the following should display:



6. Open the Command Editor by clicking in the icon illustrated below to interact with the database.



7. Connect to the PUREXML database created earlier by entering the following command within the newly opened Command Editor and pressing the button to execute this command:

```
connect to purexml;
```

- Clear the results output by this command by right-clicking on the bottom panel and selecting the “**Clear Results**” option.


4. XQuery

XQuery is used for querying XML data in the same manner as SQL is used for querying traditional relational data within databases. As we will see in the steps below, this is achieved by allowing XQuery to use XPath expression syntax to address specific parts of an XML document.

4.1 Using XML Queries

We are going to start by querying an XML document that contains a list of customers with information, such as name, address, phone number, etc.

Note: All of the commands below should be placed on a single line as one query.

- Enter the following query within the Command Editor window and click  to execute it and retrieve the results:

```
xquery db2-fn:xmlcolumn("CUSTOMER.INFO");
```

You probably noticed that the function `xmlcolumn` returns the complete XML document. If we want to retrieve specific information within the XML documents we can use an XPath expression. Additionally, XPath allows us to specify predicates within square brackets, in order to filter the results of your query.

- In XPath, there is a special predicate called the positional predicate that returns the node from a specified position for a given XML document. For example, the XQuery below has an XPath expression containing the positional predicate `[1]` and always returns the first phone number from every XML document (i.e. the first phone number of every customer). You may enter the query below in the Command Editor window and execute the query to see the results.

```
xquery db2-fn:xmlcolumn("CUSTOMER.INFO")
/*:customerinfo/*:phone[1]
```

- We can query for the details of customers who live in Toronto by entering the following XQuery into the Command Editor window and executing the query to see the results:

```
xquery db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo[addr
/city='Toronto'];
```

4. We can write an XPath expression that fetches the assistant name (without tags) of the customer whose Cid is greater than 1003 and belongs to Canada with the following:

```
xquery db2-fn:xmlcolumn("CUSTOMER.INFO")/*:customerinfo
[@Cid > 1003]/*:addr[@country="Canada"]/../*:assistant
/*:name/text()
```

5. Now retrieve the names of customers that have a “work” phone number of “905-555-7258” as follows:

```
xquery db2-fn:xmlcolumn('CUSTOMER.INFO')/
customerinfo/phone[@type='work' and text()='905-555-
7258']/../name
```

6. Then we can retrieve the cities where the country is “Canada” using the following query:

```
xquery db2-
fn:xmlcolumn('CUSTOMER.INFO')//addr[@country="Canada"]/city
```

7. So far we have seen how to fetch individual element/attribute values from an XML document. XQuery further allows construction of XML documents during querying. Now, we will write an XQuery that returns a single element <ShippedItems> containing the names of all items from orders that have been shipped:

```
xquery <ShippedItems>
{db2-fn:xmlcolumn("PURCHASEORDER.PORDER")
/*:PurchaseOrder[@Status="Shipped"]/*:item/*:name}
</ShippedItems>
```

8. Apart from constructing XML fragments on the fly, XQuery also allows nested loop operations. The XQuery expression shown below returns the name and quantity of all items from the purchase orders whose status is shipped (You may use a second “for” clause to iterate over the quantity of items):

```
xquery for $po in
db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/*:PurchaseOrder
for $quantity in $po/*:item/*:quantity
where $po/@Status="Shipped"
return ($po/*:item/*:name, $quantity)
```

5. SQL/XML

Apart from supporting XQuery, DB2 also provides a number of built in SQL/XML functions that can transform XML data into relational and vice versa. Some of the SQL/XML functions can also be used for parsing, serializing and casting XML data type into relational types.

We can now look at a couple of SQL/XML functions such as XMLQUERY, XMLEXISTS that are used to fetch XML nodes that satisfy a given predicate.

1. The following SELECT statement returns the customer IDs (CID) of only those customers who have an assistant:

```
select CID from CUSTOMER where XMLEXISTS
('$d/customerinfo/assistant' passing INFO as "d")
```

Here, only the CID is returned for the documents containing an assistant element.

2. The following SELECT statement returns all the customers whose address country is "Canada" and whose city is "Toronto":

```
select XMLQUERY( '$d/*:customerinfo/*:name' passing INFO
as "d") from CUSTOMER where XMLEXISTS
('$x/*:customerinfo/*:addr[@country="Canada" and
*:city="Toronto"]' passing INFO as "x");
```

3. We will now construct an XML document with a <PurchaseOrder> element tag and 4 children element tags (poid, status, custid and orderdate). The values for the document can be obtained from the PURCHASEORDER table where the POID is 5001.

```
select XMLELEMENT (NAME "PurchaseOrder",
XMLELEMENT (NAME "poid", POID),
XMLELEMENT (NAME "status", STATUS),
XMLELEMENT (NAME "custid", CUSTID),
XMLELEMENT (NAME "orderdate", ORDERDATE))
from PURCHASEORDER where POID = 5001
```

4. The SQL/XML function XMLAGG aggregates certain values together into one group. The following SELECT statement returns an XML fragment with parent element <Orders> containing all the POIDs from table PURCHASEORDER as children:

```
select XMLELEMENT (NAME "Orders",
XMLAGG (XMLELEMENT (NAME "poid", POID))) from PURCHASEORDER
```


- The XMLAGG function is commonly used with the GROUP BY clause of the SELECT statement as follows:

```
select XMLELEMENT (NAME "Orders",
XMLATTRIBUTES (STATUS as "status"),
XMLAGG (XMLELEMENT (NAME "poid", POID)))
from PURCHASEORDER group by STATUS
```

The above SELECT statement groups the result by the status of purchase orders which helps us notice that there are duplicate rows.

We are also able to construct new namespaces within XML documents using the XMLNAMESPACES function.

- For example, the following query returns a new element node <allProducts> with a namespace "http://posample.org", and children element(s) <item> containing the name from the PRODUCT table

```
select XMLELEMENT (NAME "allProducts",
XMLNAMESPACES (DEFAULT 'http://posample.org'),
XMLAGG (XMLELEMENT (NAME "item", NAME))) from PRODUCT
```

6. XMLTABLE function

The XMLTABLE function is one of the most commonly used SQL/XML function since it helps generate a relational table from XML data. This function is used to help create views for XML data. This is useful when certain portions of the XML documents need to be exposed as relational data. For example, this helps the report designer write queries for relational views without worrying about the XML data model.

- The following SELECT statement returns a relational table containing two columns (NAME as varchar(30) and ADDRESS as varchar(65)) with all of the elements of address concatenated as one single item:

```
select X.* from
XMLTABLE ('db2-fn:xmlcolumn ("CUSTOMER.INFO")/customerinfo'
COLUMNS name varchar(30) PATH 'name', address
varchar(65) PATH 'fn:string-join(addr/*," ")') as X;
```

The syntax of the XMLTABLE function is straightforward. It takes an XQuery or XPath expression as input and populates the named relational columns with values of the XPath expression and the PATH clause.

Note: Make sure that the resulting values from the path expressions always yield atomic values to successfully cast the values into relational data types.

For XPath expressions resulting in multiple values, these values can be stored as part of an XML column in the relational table.

- We will now retrieve table data containing columns storing the customer names and an XML column containing an XML file with customer phone numbers:

```
select X.* from
XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
COLUMNS name varchar(30) PATH 'name',
phone xml PATH 'for $x in phone return $x') as X;
```

Please note that if there is more than one phone element in the same XML document, then they will all appear in the same XML column value.

The XMLTABLE function can also be used to populate another relational table by using the SELECT statement along with the INSERT statement

- For example, we can use the SQL statements below to first create a table named CUSTOMERDATA with the given schema:

```
create table CUSTOMERDATA (CID integer, NAME varchar(30),
CITY varchar(20), COUNTRY varchar(20));
```

- We can then use the INSERT statement to populate the table with the result set of the XMLTABLE function as follows:

```
insert into CUSTOMERDATA
select X.* from CUSTOMER,
XMLTABLE ('$d/customerinfo' passing INFO as "d" COLUMNS
cid integer PATH '@Cid', name varchar(30) PATH 'name',
city varchar(20) PATH 'addr/city', country varchar(20) PATH
'addr/@country') as X;
```

- Finally, we can check the result running by running the following query:

```
select * from CUSTOMERDATA
```

Suggested Reading

External Links:

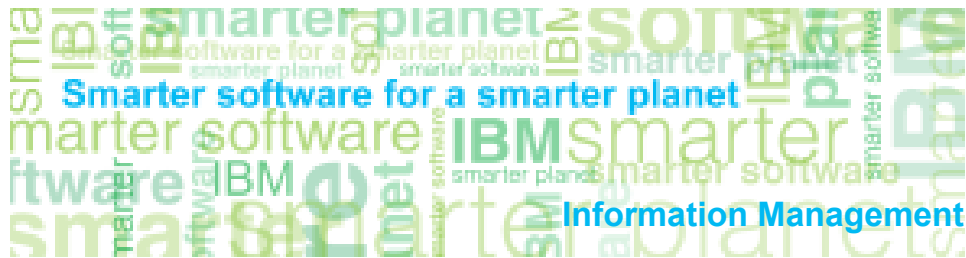
- XQuery 1.0:
www.w3.org/TR/XQuery
- XQuery Tutorial:
www.w3schools.com/XQuery/default.asp

- XQuery FLWOR Expressions:
http://www.w3schools.com/XQuery/XQuery_f
- What is XQuery:
<http://www.XQuery.com/>

Articles:

- An Introduction to XQuery, by Howard Katz.
<http://www-128.ibm.com/developerworks/xml/library/x-XQuery.html>
- Query DB2 XML Data with XQuery, by Don Chamberlin and C. M. Saracco.
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604saracco/>
- XQuery: An XML query language, by Don Chamberlin.
<http://www.research.ibm.com/journal/sj/414/chamberlin.html>
- Native XML Support in DB2 Universal Database, by Matthias Nicola and Bert Van der Linden.
<http://www.vldb2005.org/program/paper/thu/p1164-nicola.pdf>
- Query DB2 XML Data with SQL, by C. M. Saracco.
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604saracco/>
- XML Matters: Indexing XML documents, by David Mertz.
<http://www-128.ibm.com/developerworks/xml/library/x-matters10.html>
- What's new in DB2 Viper, by Cynthia M. Saracco.
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0602saracco/>

DB2® Programming Fundamentals



© 2010 IBM Corporation

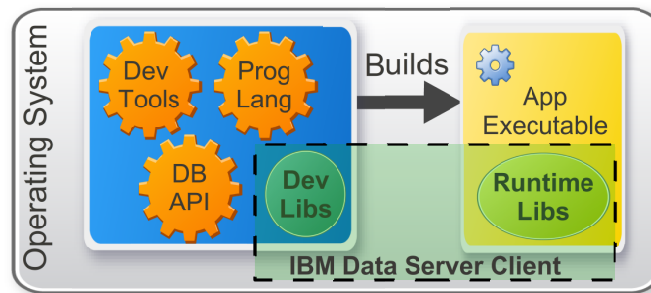


Agenda

- **Application Development Environment**
- **Embedded SQL**
- **Static SQL**
- **Dynamic SQL**
- **Routines**
 - Types
 - Benefits
 - Usage
 - Tools for Developing Routines
- **Triggers**

Application Development Environment

- **Combination of hardware and software** used to develop an application
- **The DB2 Application Development Environment (ADE) is composed of several software elements:**
 - operating system
 - IBM® Data Server Client
 - Database Application Programming Interface (API)
 - programming language
 - development tools

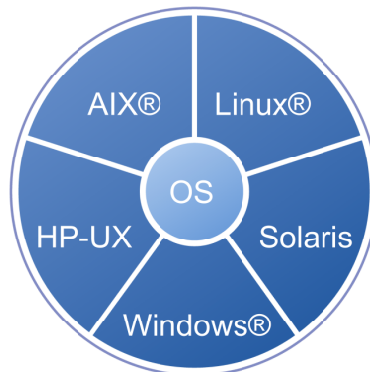


3

© 2010 IBM Corporation

Operating System

- **You can develop DB2 database applications on the following operating systems:**



4

© 2010 IBM Corporation

IBM Data Server Client

- **The IBM Data Server Client provides**
 - Support to database application development
 - DB2 administration tools
 - Runtime connectivity to applications

- **To configure the DB2 application development, you must have:**
 - installed a Data Server Client
 - completed basic configuration steps for the Data Server Client

Types of IBM Data Server Client

- **IBM Data Server Client is available in 2 packagings:**
 - **IBM Data Server Client**
 - Complete package: includes ALL development drivers and administration tools
 - Supports database administration and **application development** using an API such as ODBC, CLI, .NET, or JDBC
 - Required for applications using DB2CI API
 - **IBM Data Server Runtime Client**
 - If DB2 command line processor (CLP) support and basic client support for **running and deploying applications** is needed – i.e. no development libraries
 - Includes all runtime libraries

Types of IBM Data Server Client and Drivers

- **IBM Data Server Drivers**
 - Smaller footprint than Data Server Client
 - It can be embedded in applications for redistribution
- **Types available:**
 - **IBM Data Server Driver for JDBC and SQLJ**
 - for Java™ applications only
 - **IBM Data Server Driver for ODBC and CLI**
 - for applications using ODBC or CLI only
 - **IBM Data Server Driver Package**
 - for applications using ODBC, CLI, .NET, OLE DB, PHP, Ruby, JDBC, or SQLJ
 - if DB2 Command Line Processor Plus (CLPPlus) support is needed

7

© 2010 IBM Corporation

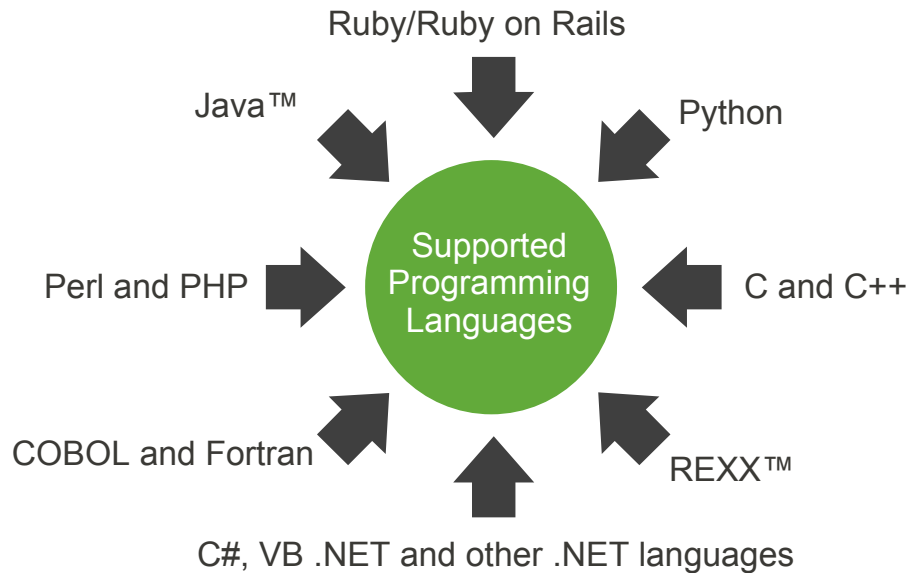
Database Application Programming Interface

- **To configure the ADE for the APIs that you will use, you must have:**
 - installed a Data Server Client
 - installed the API driver(s)
- **APIs available for use include:**
 - ADO.NET
 - DB2 CLI and ODBC
 - DB2CI (counterpart to Oracle's OCI)
 - Embedded SQL
 - JDBC and SQLJ
 - OLE DB
 - Perl
 - PHP
 - Ruby/Ruby on Rails
 - Python

8

© 2010 IBM Corporation

Programming Languages



9

© 2010 IBM Corporation

Introduction to Embedded SQL

- **Embedded SQL**
 - Applications are coded by embedding **SQL statements** within the application source code
- **Characteristics of Embedded SQL**
 - Embedded SQL database applications **connect** to databases and **execute** embedded SQL statements.
 - Embedded SQL statements are **embedded** within a host programming language code.
 - Embedded SQL statements can be executed **statically** or **dynamically**.
 - You can develop embedded SQL applications for DB2 in the following **host programming languages**:
 - C, C++, COBOL, FORTRAN, and REXX™

10

© 2010 IBM Corporation

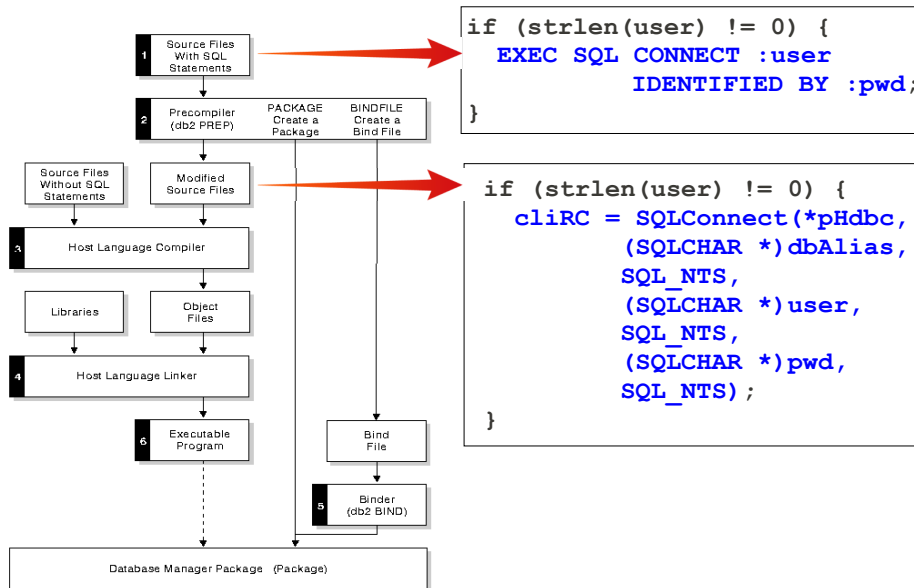
Building Embedded SQL Applications

- **Since source code has Embedded SQL code, the host language compiler cannot process it.**
 - A pre-compilation phase is necessary to replace the Embedded SQL with valid host language syntax.
- **Prior to application compilation and linking**
 - Prepare the **source files** containing embedded SQL statements using the **DB2 precompiler**. Outputs:
 - **Modified source file**
 - **Bind file** – contains access plans for static SQL statements in the application code
 - **Bind** the statements in the application to the target database.
 - Once precompiled and bound the embedded SQL application is ready to be **compiled** and **linked** using the host language-specific development tools.

Building Embedded SQL Applications - Steps

1. **Create** source files with embedded SQL
2. **Connect** to a database, then **precompile** each source file to convert embedded SQL source statements
3. **Compile** the modified source files (and other files without SQL statements) using the host language compiler (Eg: C compiler)
4. **Link** the object files with the DB2 and host language libraries to produce an executable program.
 - Compiling and linking (steps 3 and 4) create the required object modules
5. **Bind** the bind file, if this was not already done at precompile time, or if a different database is going to be accessed
6. **Run** the application

Building Embedded SQL Applications - Steps



13

© 2010 IBM Corporation

PREP and BIND

▪ PREP (PRECOMPILE)

- Reads your source code, parses and converts the embedded SQL statements to DB2 run-time services API calls
- writes the output to a new modified source file
- The precompiler produces access plans for the SQL statements which are stored together as a package within the database

▪ BIND

- done by default during precompilation (the PREP command)
- if deferred then the BINDFILE option needs to be specified at PREP time in order for a bind file to be generated

14

© 2010 IBM Corporation

Static and Dynamic SQL

- **There are two different types of SQL statements:**
 - statically executed SQL
 - dynamically executed SQL
- **Statically executed SQL statements**
 - Syntax is **fully known** at precompile time
 - names for the columns and tables referenced in a statement must be fully known at precompile time
 - Static → SQL statement doesn't change
 - SQL statements are compiled (access plan is created) before the application is built.
 - Statically executed SQL is **best used** on databases whose **statistics do not change** a great deal.
 - Since the access plan is created at compilation time.

```
EXEC SQL UPDATE staff
SET salary = salary + 10000
WHERE id >= 310 AND dept = 84;
```

15

© 2010 IBM Corporation

Static and Dynamic SQL

- **Dynamically executed SQL statements**
 - are **built** and executed by an application at **run-time**
- **A Scenario where Dynamic SQL would be used:**
 - an **interactive application** that prompts the end user for key parts of an SQL statement
 - Eg: Search for employees based on their name, or the last name, or both.

```
strcpy(hostVarStmtDyn,
"UPDATE staff SET salary = salary * 1.1 WHERE dept = ?");

EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
EXEC SQL EXECUTE StmtDyn USING :dept;
```

SQL statement is created at execution time

SQL statements is dynamically prepared and executed

16

© 2010 IBM Corporation

Routines

- **Routines are database objects:**
 - can encapsulate **programming** and **database logic** that can be invoked like a programming sub-routine from a variety of SQL interfaces
- **There are many useful applications and benefits of using routines within a database or database application. Egs:**
 - Extending built-in SQL function support
 - Encapsulate application logic that can be invoked from an SQL interface
 - Improve application performance by reducing network traffic
 - Allow for faster, more efficient SQL execution
 - Allow the interoperability of logic implemented in different programming languages
 - Access to features that exist only on the server
 - Enforcement of business rules

17

© 2010 IBM Corporation

Types of Routines

- **Definition of a routine can be**
 - **System-defined:** built-in; provided with the product
 - **User-defined:** created by users
- **The supported functional types of routines are:**
 - Functions
 - Procedures (also called stored procedures)
 - Methods
- **The supported routine implementations are:**
 - Built-in routines
 - Eg: SUM(), COUNT() are built-in functions
 - Sourced routines
 - SQL routines
 - Composed of SQL and SQL PL (Procedural Language)
 - External routines
 - Developed outside the DB2 database using a programming language (Eg: Java, C, C++, etc)

18

© 2010 IBM Corporation

System-defined and User-defined Routines

- **System-defined routines**
 - Provided with the product
 - Immediately **ready-to-use**
 - Require the necessary privileges to invoke these routines
- **User-defined routines**
 - Created by the **user**
 - Extend the SQL language beyond the support which is currently available
 - **Implemented** in a variety of ways including:
 - sourcing built-in routines
 - using SQL statements only
 - using SQL with another programming language

User Defined Functions

- **User-Defined Functions (UDFs)** are special objects that are used to extend and enhance the support provided by the built-in functions available with DB2.
- Unlike DB2's built-in functions, user-defined functions can take advantage of **system calls and DB2's administrative APIs**.
 - **SQL UDFs** – coded using SQL PL
 - **External UDFs** – coded using a programming language
- Functions always return a value:
 - **SQL Scalar, Table, or Row**
- User-defined functions are created (or registered) by executing the **CREATE FUNCTION** SQL statement.

User Defined Functions – Example

- UDF returning a table as result
- The **CREATE FUNCTION** statement defines a table function that returns the employees in a specified department number.

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO VARCHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN

      SELECT EMPNO, LASTNAME, FIRSTNAME FROM EMPLOYEE
      WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

21

© 2010 IBM Corporation

User Defined Functions :: Executing

- Functions can be invoked from inside a SQL statement
- Example: a **SELECT** statement that makes use of the **DEPTEMPLOYEES** function

```
SELECT EMPNO, LASTNAME, FIRSTNAME FROM
      TABLE(DEPTEMPLOYEES('A00')) AS D
```
- The User-Defined Table Function is invoked by referencing the function in the FROM clause of an SQL statement where it can process a set of input values.
- The reference to the table function must be preceded by the **TABLE** clause and be contained in brackets.

22

© 2010 IBM Corporation

Stored Procedures

- It is an ordinary program composed entirely of SQL statements and SQL PL code that can be called by an application.
- Stored procedures (SP) can be called **locally or remotely**.
 - **Locally:** from another stored procedure or trigger
 - **Remotely:** from an application
- An **external stored procedure** is a stored procedure that is written using a high-level programming language
 - External stored procedures can be **more powerful** than SQL stored procedures because they can take advantage of system calls and administrative APIs along with SQL statements.
 - The drawback is that since they are external to the DB2 engine, they are usually not as efficient as SQL Stored Procedures.

23

© 2010 IBM Corporation

Stored Procedures – SQL PL Support

- The **SQL Procedural Language (SQL PL)** is a language extension of SQL
 - consists of statements and language elements
 - used to **implement procedural logic** in SQL statements
 - Conditional (IF), loops (FOR), exception handling, etc
- **SQL procedures with SQL PL**
 - allows you to effectively **program** in SQL
 - **complete set of SQL PL** statements can be used in SQL procedures

24

© 2010 IBM Corporation

Stored Procedures – PL/SQL Support

- **PL/SQL (Procedural Language/Structured Query Language) statements:**
 - can be **compiled** and **executed** using **DB2 interfaces**
 - reduces the complexity of enabling **existing PL/SQL** solutions to **work** with the DB2 data server
- **The supported interfaces include:**
 - DB2 command line processor (CLP)
 - DB2 CLPPlus
 - IBM® Data Studio
 - IBM Optim™ Development Studio
- **PL/SQL statement execution is not enabled from these interfaces by default. PL/SQL statement execution support must be enabled on the DB2 data server.**

Stored Procedures – Creating and Invoking

- **Stored Procedures**
 - created by executing the **CREATE PROCEDURE** statement
 - invoked by executing the **CALL** statement with a reference to a procedure
 - can take **input**, **output**, and **input-output** parameters, execute a wide variety of SQL statements, and **return multiple result sets** to the caller
- **Procedures can be invoked from anywhere that the CALL statement is supported including:**
 - client applications
 - External routines (procedure, UDF, or method)
 - SQL routines (procedure, UDF, or method)
 - Triggers (before triggers, after triggers, or instead of triggers)
 - Dynamic compound statements
 - Command line processor (CLP)

Stored Procedures – Example

- An example **CREATE PROCEDURE** statement for the **DEPT_MEDIAN** procedure signature is as follows:

```
CREATE PROCEDURE DEPT_MEDIAN  
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

- Using the **CALL** statement from the **CLP**
 - specify the procedure name and appropriate parameter arguments

```
db2 call dept_median (51, ?)
```

External Routines

- **External routines**
 - Routine logic is implemented in a programming language application that resides **outside** of the database
- **You can create**
 - external procedures
 - external functions
 - external methods.
- **Benefits**
 - harness the full functionality and performance of the chosen implementation programming language
 - access and manipulate entities outside of the database
- **When to use an External routine**
 - require a smaller degree of interaction with the DB2 database, but that must contain a **lot of logic** or **very complex logic**

Developing Routines

▪ Procedure for **Developing Routines**

1. When there is no system-defined routine available that provides the functionality that is required
2. Determine what type of routine to create
3. What implementation to use
 - SQL Routine
 - External Routine
4. Define the interface for the routine
5. Develop the routine logic
6. Execute SQL to create the routine
7. Test the routine
8. Deploy it for general use

Tools for Developing Routines

▪ **Graphical User-Interface (GUI) tool, provided with DB2:**

– IBM® Data Studio

- easy-to-use development environment
- simplify the process of creating routines
- develop stored procedures on one operating system and build them on other server operating systems

▪ **Command Line Interface, provided with DB2:**

– DB2 Command Line Processor (DB2 CLP)

Triggers

- A **trigger** defines a set of actions that are performed in response to an **insert**, **update**, or **delete** operation on a specified table.
- **Like constraints**, triggers are often used to enforce data integrity and business rules.
- **Unlike constraints**, triggers can also be used to update other tables, automatically generate or transform values for inserted or updated rows, and invoke functions to perform tasks such as issuing errors or alerts.
- Using triggers places the logic that **enforces business rules** inside the database.

31

© 2010 IBM Corporation

Triggers – Example

- Suppose you had the following **EMPLOYEES** base table..

Column Name ...	Data Type ...
EMPNO	INTEGER
FNAME	CHAR(20)
LNAME	CHAR(30)
TITLE	CHAR(10)
DEPARTMENT	CHAR(20)
SALARY	DECIMAL(6,2)

..and you wanted to create a trigger for **EMPLOYEES** that will store information about salary changes in a table called **SALARY_HIST**.

32

© 2010 IBM Corporation

Triggers – Example (Continued)

The action is to be applied *after* the changes caused by the actual update of the subject table

Names the trigger

```
CREATE TRIGGER empno_inc
AFTER UPDATE ON employees
REFERENCING NEW AS n OLD AS o
FOR EACH ROW
  INSERT INTO salary_hist
  VALUES (o.empno,
          o.salary,
          CURRENT TIMESTAMP)
```

The action is to be applied once *for each row* affected by the trigger

Specifies the action to be performed when a trigger is activated

33

© 2010 IBM Corporation

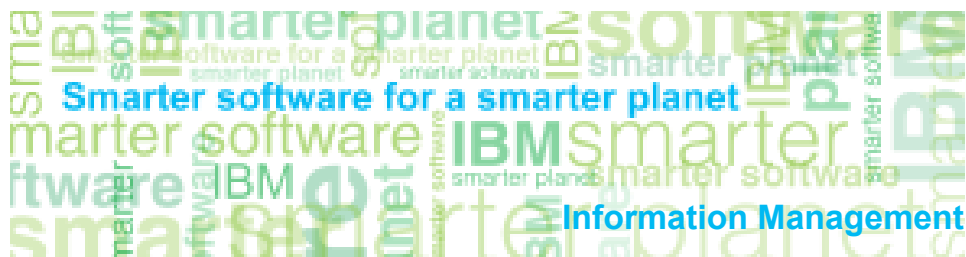
Information Management Ecosystem Partnerships
IBM Canada Lab



Questions?

Summer/Fall 2010

E-mail: imschool@us.ibm.com
Subject: “DB2 Academic Workshop”



© 2010 IBM Corporation



IBM DB2[®] 9.7

**Accessing DB2
Databases from
Applications
Hands-On Lab**

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

CONTENTS	1
1. INTRODUCTION	3
2. OBJECTIVES OF THIS LAB	3
3. SETUP AND START DB2	4
3.1 Environment Setup Requirements	4
3.2 Login to the Virtual Machine	4
3.3 SAMPLE Database	5
3.4 Create and populate the table.....	5
4. CONFIGURING THE APPLICATION FOR ACCESS TO DB2 USING JDBC 6	
4.1 Open the Application in IBM Data Studio.....	6
4.2 Install JDBC Driver.....	9
5. CONNECTING TO DB2	12
5.1 Closing the Connection.....	13
6. QUERYING DATA	13
6.1 Incorporating SELECT with the Application	15
6.2 Search the Database using the Application	16
7. INSERTING DATA	19
7.1 Incorporating INSERT with the Application.....	20
7.2 Insert into the Database using the Application.....	21

1. Introduction

DB2 provides support to a wide range of programming languages and APIs (Application Programming Interface) that allow applications to access and manipulate data in a DB2 database.

A development environment is typically composed of several elements such as the operating system, programming language, database API and DBMS drivers, development tools and of course the database server. The combination of these elements will define which database API and drivers you will need in order to access a DB2 database.

Database drivers and APIs are specialized pieces of software that implement the necessary functions to execute operations against a DBMS, and transfer data between the database server and the client application. This way, developers can focus on coding business logic in their system and leave the details of data communication to the drivers.

2. Objectives of This Lab

In this lab, we assume that your company wants you to create a Java application capable of querying a database for employee information. You then decide to use JDBC (Java Database Connectivity) as the API to access the DB2 server.

After completion of this lab, the student should be able to:

- Install the JDBC driver in IBM Data Studio
- Write Java code that can:
 - Create a connection to DB2
 - Properly close a connection to DB2
 - Query data using SELECT statements
 - Add new data to the database

3. Setup and Start DB2

3.1 Environment Setup Requirements

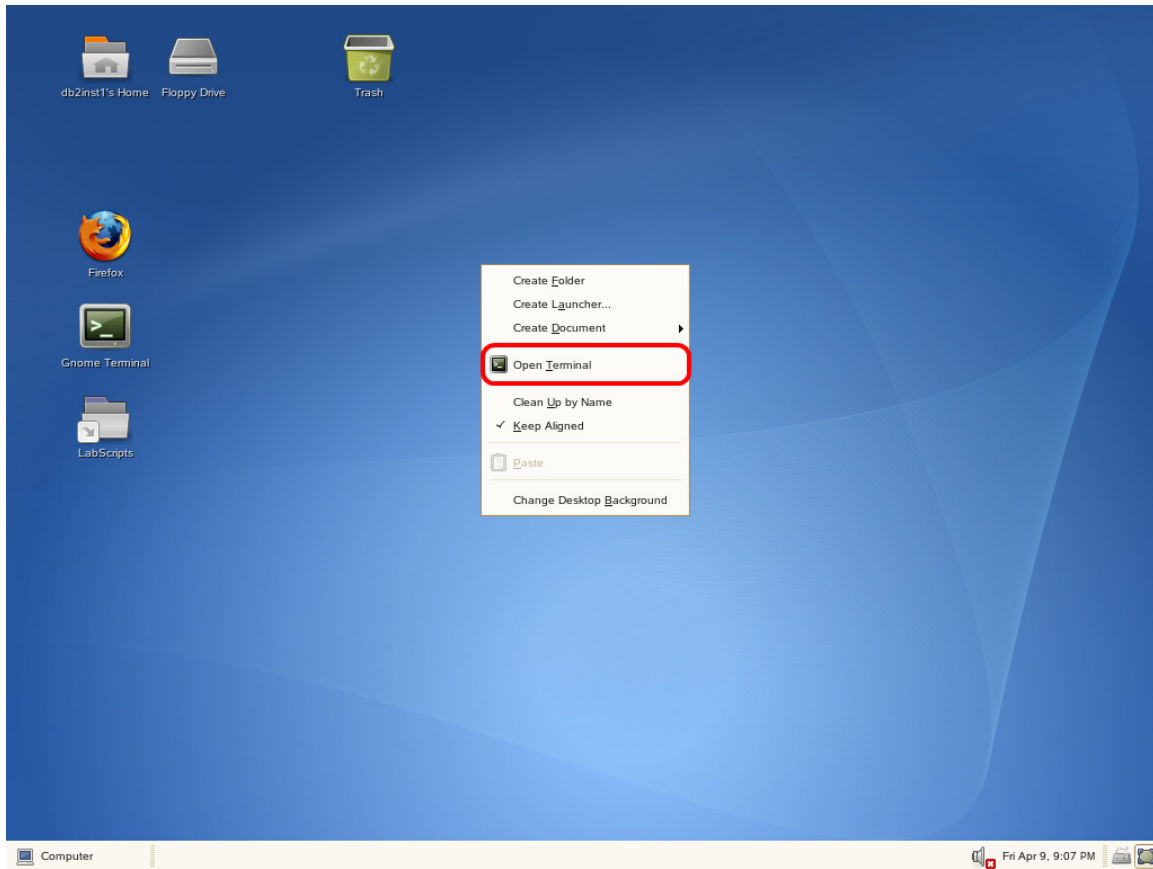
To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in the **VMware Basics and Introduction** module.

3.2 Login to the Virtual Machine

1. Login to the VMware virtual machine using the following information:
User: **db2inst1**
Password: **password**
2. Type in the command “**startx**” to bring up the graphical environment.
3. Open a terminal window as by right-clicking on the **Desktop** area and choose the “**Open Terminal**” item.



4. Start up DB2 Server by typing “**db2start**” in the terminal window.

```
db2start
```

3.3 SAMPLE Database

For executing this lab, you will need the DB2’s sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the **SAMPLE** database:

```
db2 force applications all
db2 drop db sample
db2samp1
```

3.4 Create and populate the table

We will create a simple table that will be updated during this lab session. The table named “**ESQLEMPLOYEE**” will be created and will be populated with 1 row of data.

1. Change to the directory where the script files are.

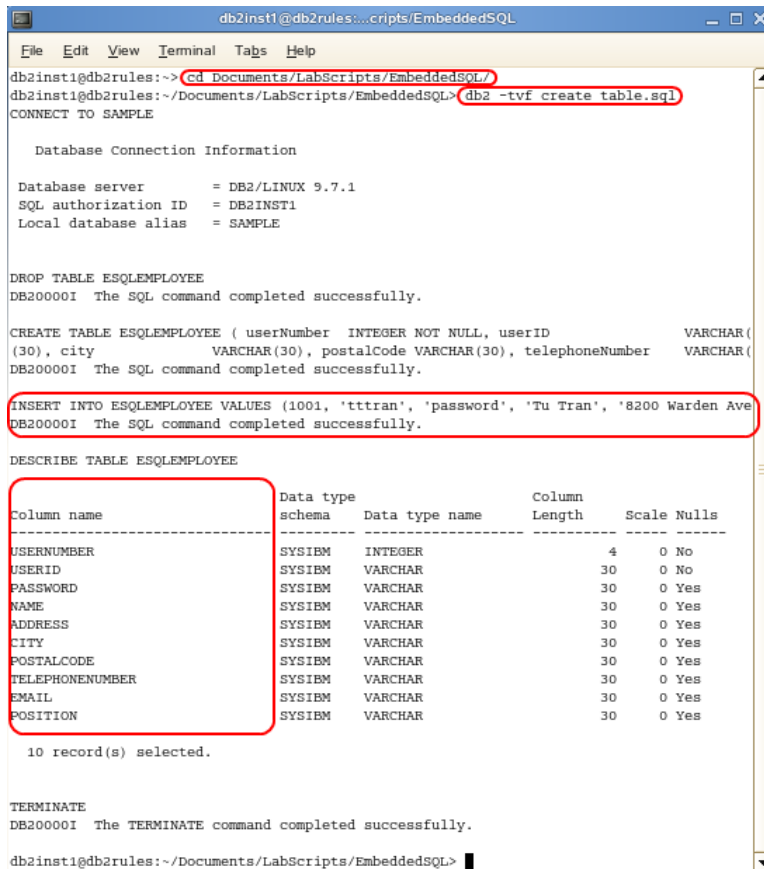
```
cd /home/db2inst1/Documents/LabScripts/EmbeddedSQL
```

2. We will take a look at the simple query first by using the command

```
cat create_table.sql
```

3. To run the query, in the terminal window, type in

```
db2 -tvf create_table.sql
```



```
db2inst1@db2rules:~/Documents/LabScripts/EmbeddedSQL
db2inst1@db2rules:~/Documents/LabScripts/EmbeddedSQL> db2 -tvf create_table.sql
CONNECT TO SAMPLE

Database Connection Information

Database server      = DB2/LINUX 9.7.1
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

DROP TABLE ESQLEMPLOYEE
DB20000I The SQL command completed successfully.

CREATE TABLE ESQLEMPLOYEE ( userNumber  INTEGER NOT NULL, userID          VARCHAR(
(30), city          VARCHAR(30), postalCode VARCHAR(30), telephoneNumber  VARCHAR(
DB20000I The SQL command completed successfully.

INSERT INTO ESQLEMPLOYEE VALUES (1001, 'ttran', 'password', 'Tu Tran', '8200 Warden Ave
DB20000I The SQL command completed successfully.

DESCRIBE TABLE ESQLEMPLOYEE

Column name          Data type          Column
schema              Data type name     Length   Scale Nulls
-----
USERNUMBER           SYSIBM             INTEGER   4       0 No
USERID               SYSIBM             VARCHAR   30      0 No
PASSWORD             SYSIBM             VARCHAR   30      0 Yes
NAME                 SYSIBM             VARCHAR   30      0 Yes
ADDRESS              SYSIBM             VARCHAR   30      0 Yes
CITY                 SYSIBM             VARCHAR   30      0 Yes
POSTALCODE           SYSIBM             VARCHAR   30      0 Yes
TELEPHONENUMBER     SYSIBM             VARCHAR   30      0 Yes
EMAIL                SYSIBM             VARCHAR   30      0 Yes
POSITION             SYSIBM             VARCHAR   30      0 Yes

10 record(s) selected.

TERMINATE
DB20000I The TERMINATE command completed successfully.

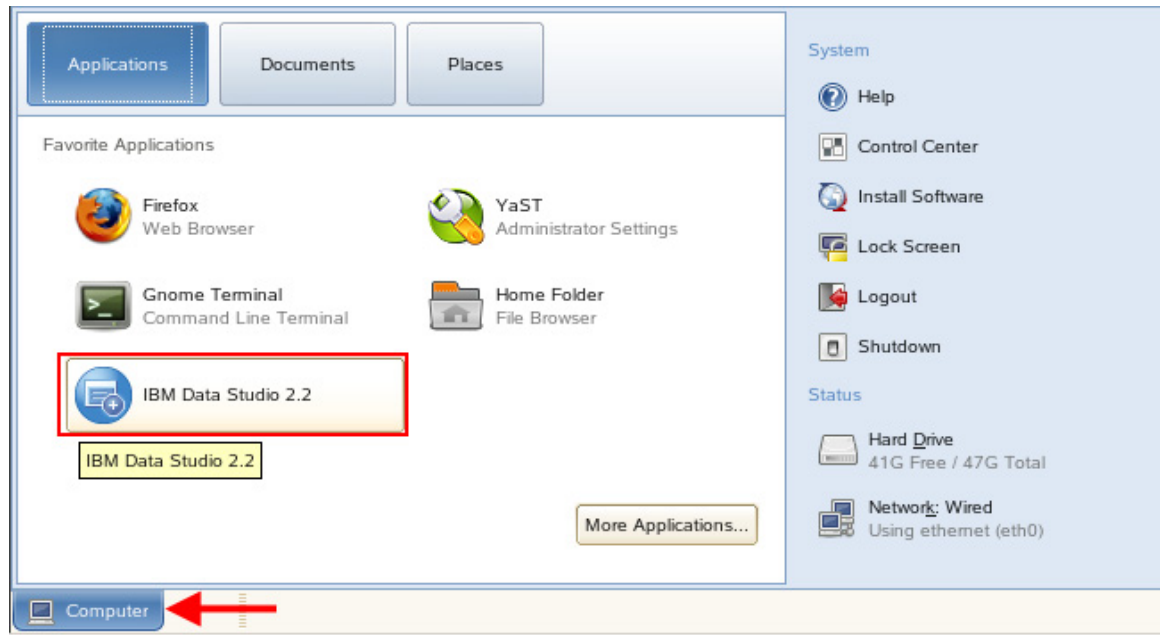
db2inst1@db2rules:~/Documents/LabScripts/EmbeddedSQL>
```

4. Configuring the Application for Access to DB2 Using JDBC

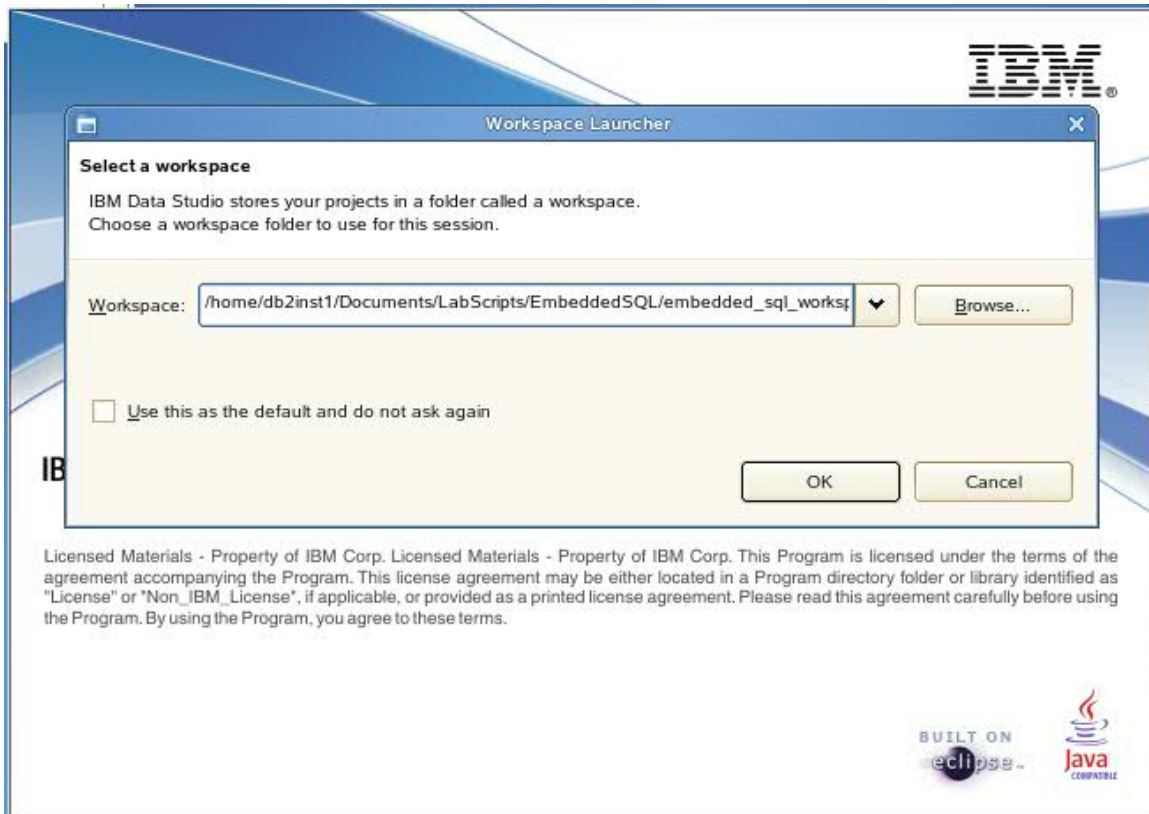
4.1 Open the Application in IBM Data Studio

Now that the database is ready and the ESQLEMPLOYEE table is created, we need to open the application that we will be working with in IBM Data Studio. Once opened, we can begin configuring the DB2 JDBC driver for the application.

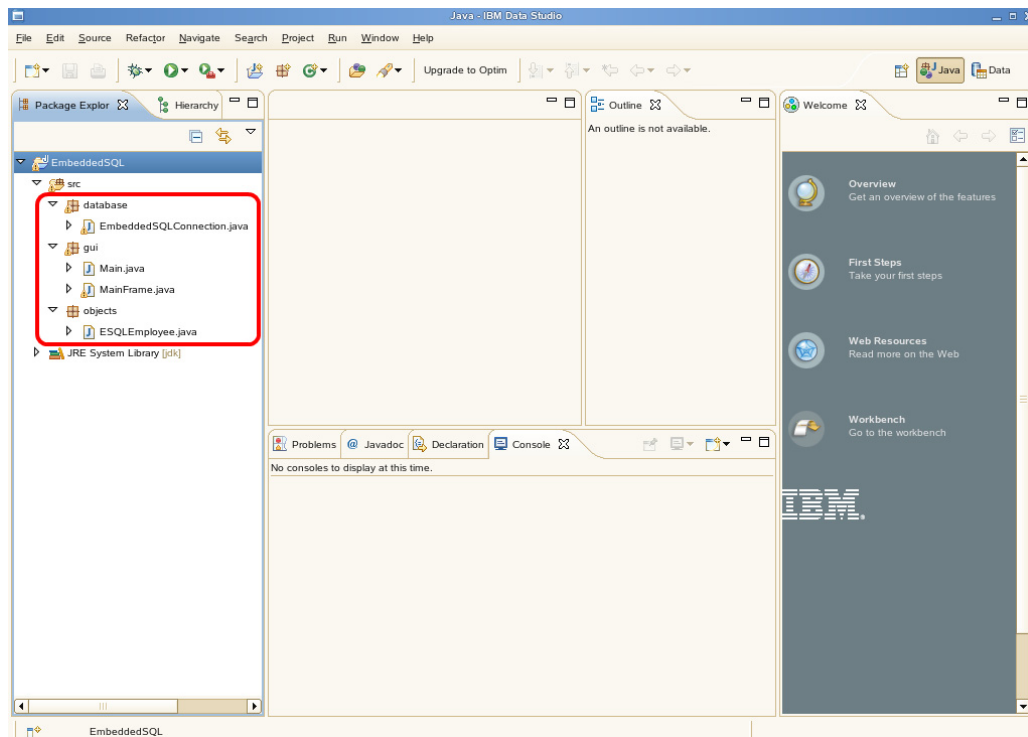
1. Open IBM Data Studio by clicking **Computer** and choosing **IBM Data Studio 2.2**.



2. A prompt to select a workstation will appear. Enter `~/home/db2inst1/Documents/LabScripts/EmbeddedSQL/embedded_sql_workspace` as the workstation and select **OK**.



3. The following screen will be displayed.

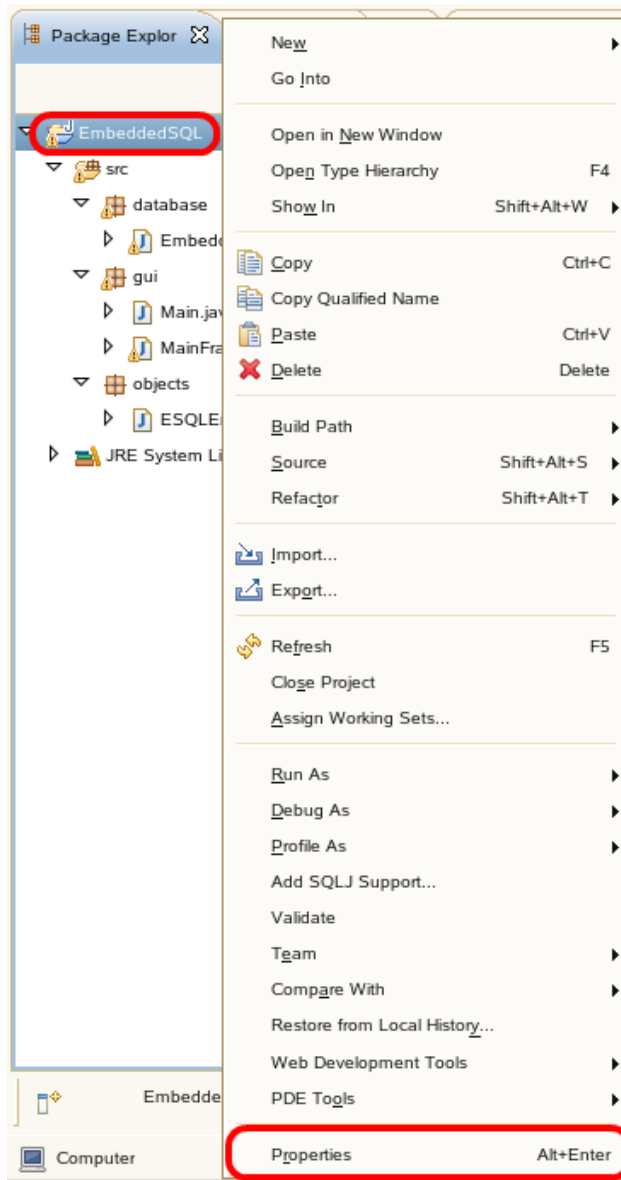


We have successfully opened our application in IBM Data Studio; however it is not ready to connect to DB2 just yet. In order to connect to DB2 we must first install the JDBC driver in our project.

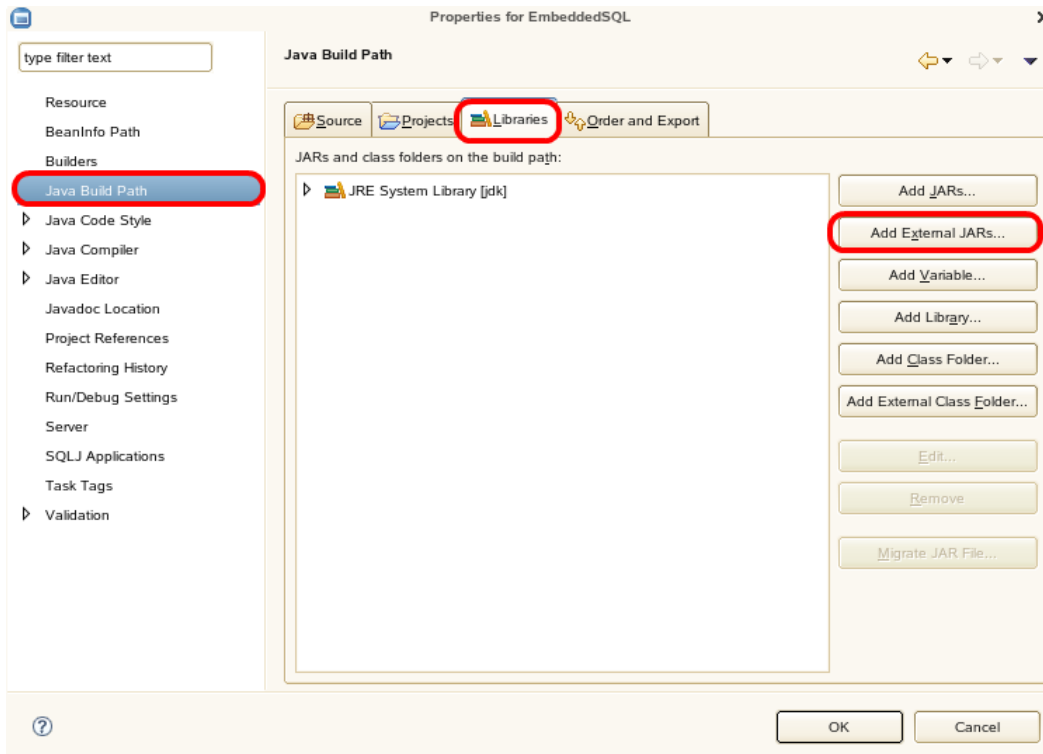
4.2 Install JDBC Driver

The JDBC Driver allows Java applications to connect to SQL compliant databases, send SQL statements, and process return messages and data.

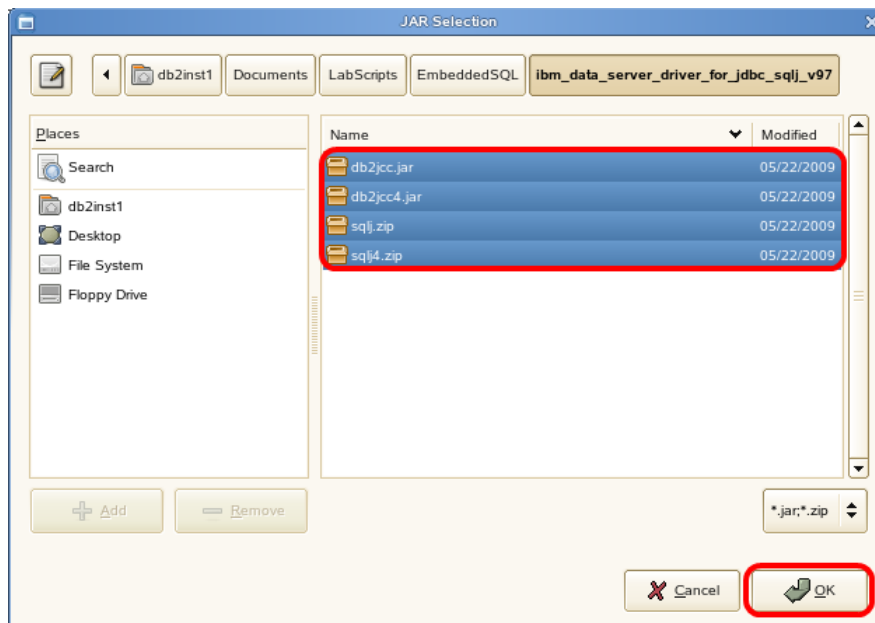
1. With IBM Data Studio opened, right click **EmbeddedSQL** and select **Properties**.



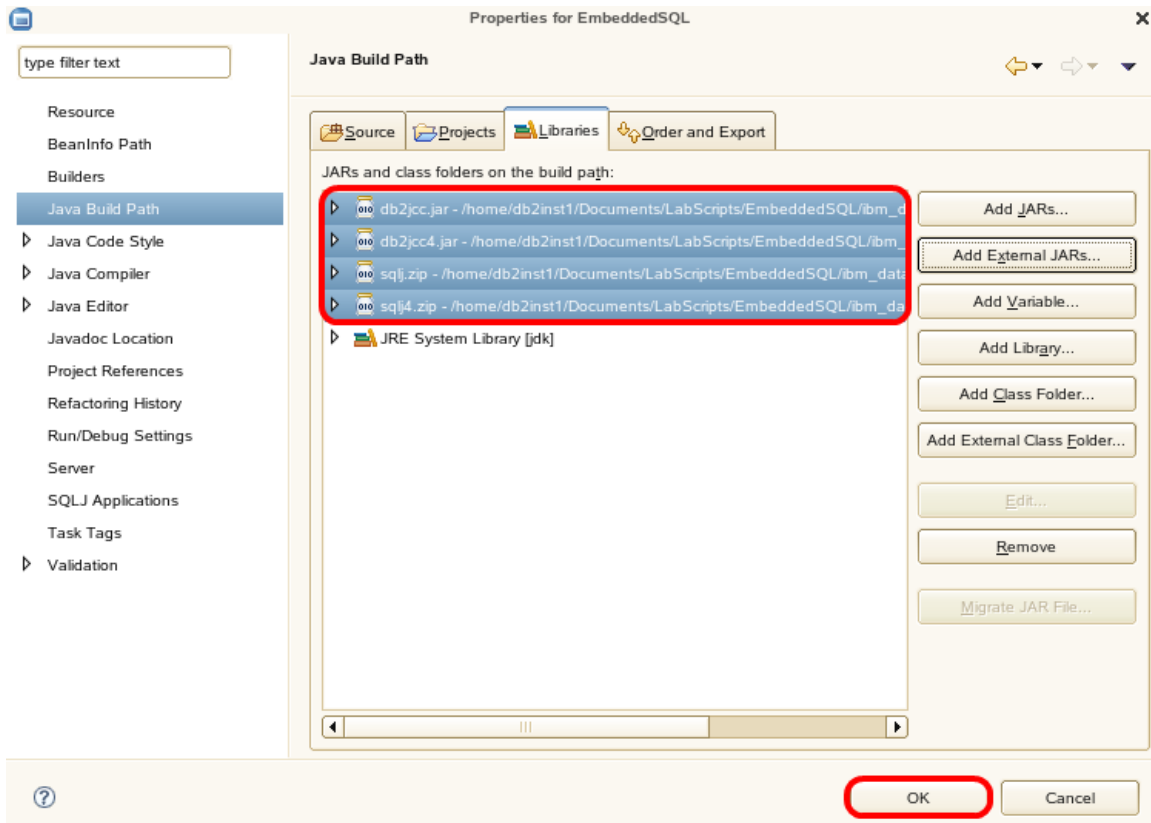
2. Select **Java Build Path** from the list. Then select the **Libraries** tab and click the **Add External JARs** button.



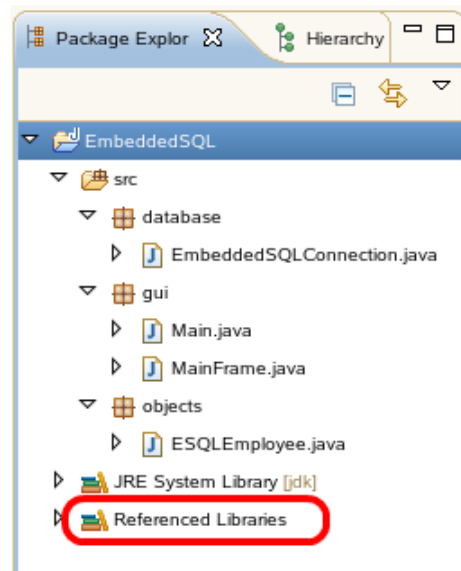
3. Browse to `~/home/db2inst1/Documents/LabScripts/EmbeddedSQL/ibm_data_server_driver_for_jdbc_sqlj_v97` and select all the files in this folder.



- The selected files will now appear under the **Libraries Tab**. Select **OK** to continue.



- The **JDBC Driver** has been successfully installed. You can view the libraries that were added by selecting **Referenced Libraries** in the **Package Explorer**.

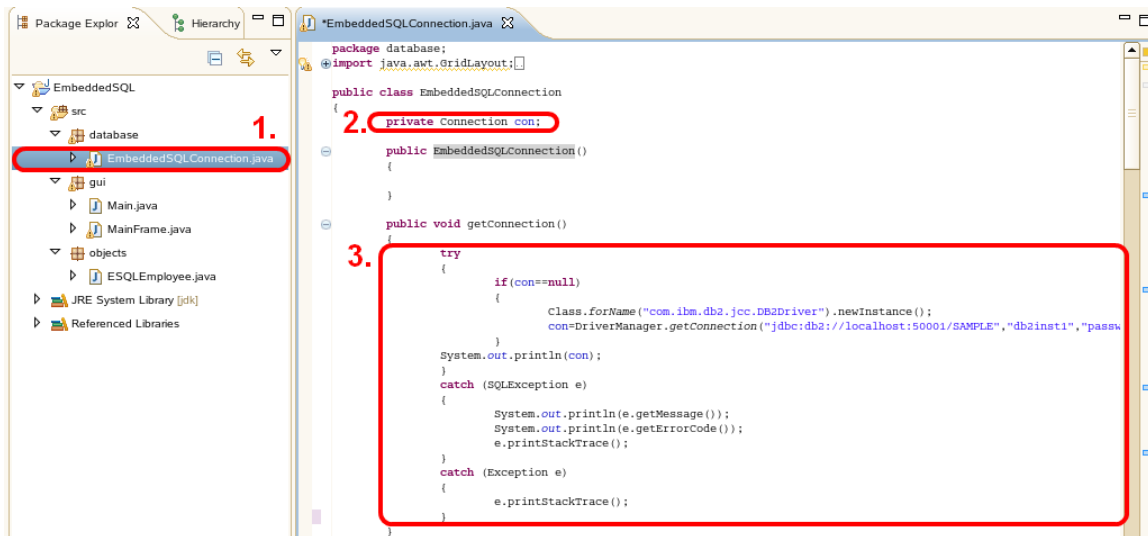


Now that the JDBC driver is properly set up in our development environment, we can start adding code in our application to connect to DB2.

5. Connecting to DB2

Whenever we wish to interact with a database, we must first establish a connection to the database server.

1. In the database package, open the EmbeddedSQLConnection class. Select the EmbeddedSQL project and press F5 to open the class



The EmbeddedSQLConnection class will be the most important class in this exercise. This class contains all the functions in the application related to data access.

2. In order to create a connection, a variable of type Connection must first be declared to hold the Connection object.

```
private Connection con;
```

Now we can create the connection to the DB2 server using the DriverManager object, and finally we store that connection in the “con” variable.

3. Complete the getConnection() function by uncommenting the code provided within getConnection().

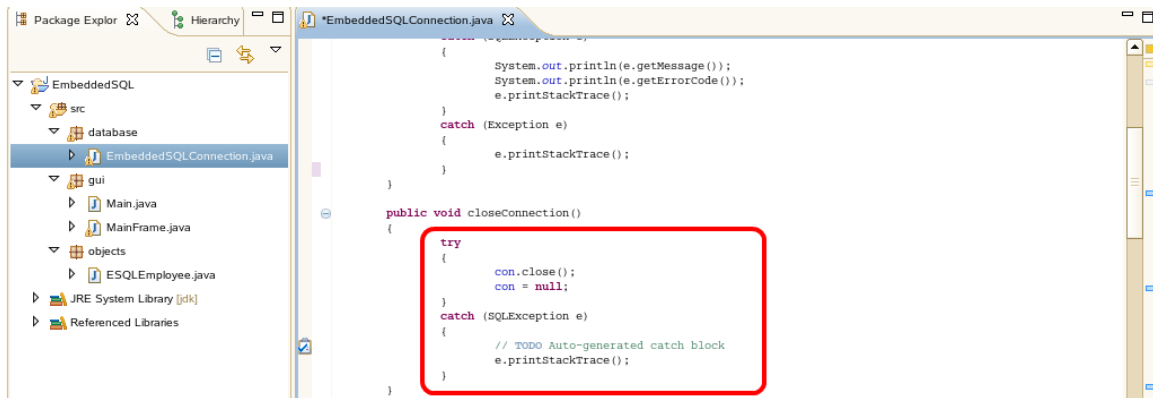
```
con=DriverManager.getConnection("jdbc:db2://localhost:50001/SAMPLE","db2inst1","password");
```

The method getConnection() is attempting to establish a connection to the given database URL. We are connecting to DB2’s SAMPLE database with the user “db2inst1” using the password “password” through the JDBC API on port 50001.

5.1 Closing the Connection

Now that we know how to establish a connection, we also need to know how to properly close our connection to DB2 once it is not needed any more. This is an important step as it will free up system's resources for your application and the database server.

1. In the EmbeddedSQLConnection class, complete the closeConnection() function by uncommenting the code provided within closeConnection().



```
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

public void closeConnection()
{
    try
    {
        con.close();
        con = null;
    }
    catch (SQLException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

```
con.close()
```

The method close() is terminating the connection to the database specified during the getConnection() method above. Once the connection is terminated we can also set `con = null`.

6. Querying Data

Now that we have created functions to create and close a connection to DB2, we are ready to write a query to search through and display data.

1. In the EmbeddedSQLConnection class, complete the getEmployeeInformation() function by uncommenting the code provided within getEmployeeInformation ().

```
public boolean getEmployeeInformation(ESQLEmployee u, String name)
{
    PreparedStatement s = null;
    try
    {
        String query = "SELECT userNumber, userID, password, name, address, city, postalCode, telepho
        s = con.prepareStatement(query);
        s.setString(1, name);
        ResultSet rs=s.executeQuery();

        while(rs.next())
        {
            u.changeUserNumber(rs.getInt(1));
            u.changeUserID(rs.getString(2));
            u.changePassword(rs.getString(3));
            u.changeName(rs.getString(4));
            u.changeAddress(rs.getString(5));
            u.changeCity(rs.getString(6));
            u.changePostalCode(rs.getString(7));
            u.changeTelephoneNumber(rs.getString(8));
            u.changeEmail(rs.getString(9));
            u.changePosition(rs.getString(10));
        }
    }
    catch (SQLException e1)
    {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    return true;
}
```

The `getEmployeeInformation()` function selects employee information based on a provided name. It is used by the application to search through the database for a specific employee.

2. In order to create and execute a query, an object of type `PreparedStatement` must first be specified.

```
PreparedStatement s = null;
```

The `PreparedStatement` object “s”, will be used to hold the SQL `SELECT` statement.

3. The SQL statements itself is coded as a `String`.

```
String query = "SELECT userNumber, userID, password, name, address, city, postalCode, telephoneNumber, email, position FROM ESQLEMPLOYEE WHERE name = ?";
```

4. We can now create the `PreparedStatement` using the `Connection` object `con` and the `prepareStatement()` method. The resulting object is stored in the variable “s”.

```
s = con.prepareStatement(query);
```

5. Looking at the query `String`, notice the “?”. This is called a parameter marker. It marks the place where a value will be inserted during runtime, in this case, the search criteria provided by the user when executing the application.

The following command is used to associate a value with the parameter marker.

```
s.setString(1, name);
```

For example: If we wish to search for an employee with the name Tu Tran, the String query becomes, "SELECT userNumber, userID, password, name, address, city, postalCode, telephoneNumber, email, position FROM ESQLEMPLOYEE WHERE name = Tu Tran";.

6. As we execute the query, the results returned from the query must be stored. We store the data in an object of the type ResultSet.

```
ResultSet rs=s.executeQuery();
```

7. Finally we can retrieve the data stored in the ResultSet.

```
while(rs.next())
{
    u.changeUserNumber(rs.getInt(1));
    u.changeUserID(rs.getString(2));
    u.changePassword(rs.getString(3));
    u.changeName(rs.getString(4));
    u.changeAddress(rs.getString(5));
    u.changeCity(rs.getString(6));
    u.changePostalCode(rs.getString(7));
    u.changeTelephoneNumber(rs.getString(8));
    u.changeEmail(rs.getString(9));
    u.changePosition(rs.getString(10));
}
```

6.1 Incorporating SELECT with the Application

We have created functions to create and close a connection to DB2 as well as to return data using a SELECT statement. How can we use these functions in our application?

1. In the GUI package, open the MainFrame class.



The MainFrame class is where we will be using the functions created to interact with DB2. This class contains all the functions in the application that are responsible for allowing the user to interact with the application.

2. Go to the Search() function and uncomment the code provided.


```

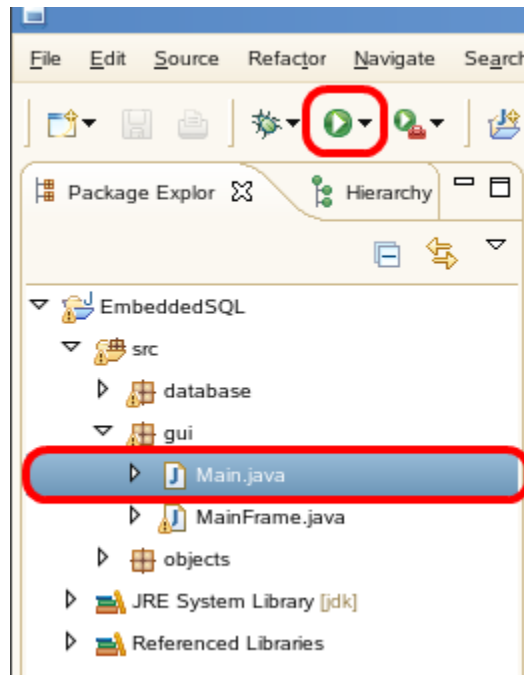
//5.1 Incorporating SELECT with the Application
x.getConnection();
x.getEmployeeInformation(u, name.getText());
x.closeConnection();

```

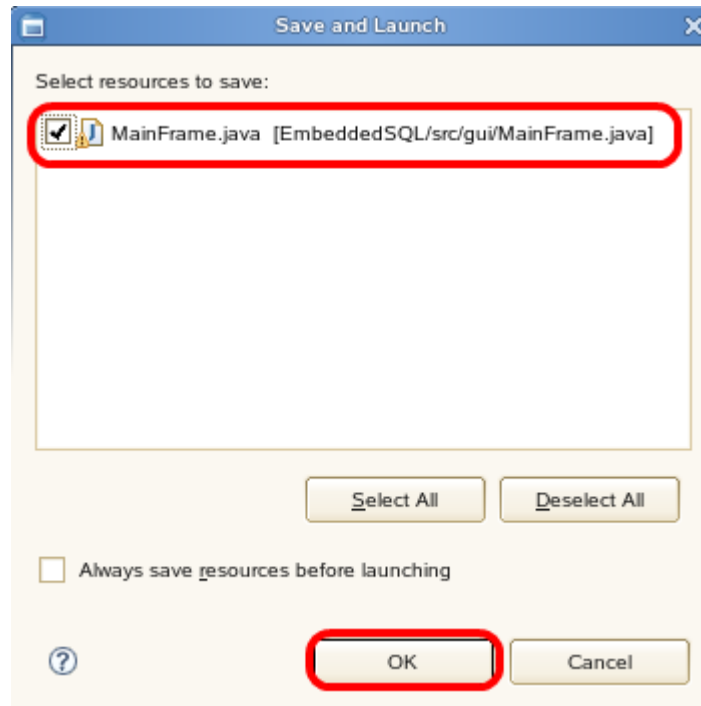
6.2 Search the Database using the Application

The application is now capable of performing a SELECT statement on the database and displaying the information returned.

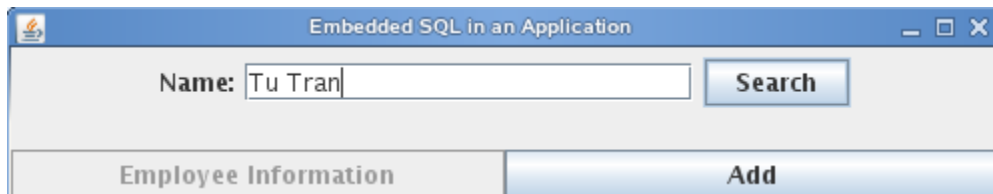
1. In the gui package, open the Main class and press .



2. If you have not saved your changes, you will be prompted to save the file(s). Select all the resources that need to be saved and press **OK**.



3. The following program will appear. Enter the name "Tu Tran" and press Search.



Employee Information

User Number: 1001	
User ID: tttran	tttran
Password:	••••••••
Name: Tu Tran	Tu Tran
Address: 8200 Warden Ave	8200 Warden Ave
City: Markham	Markham
Postal Code: L6G 1C7	L6G 1C7
Telephone Number: 2432	2432
Email: tttran@ca.ibm.ca	tttran@ca.ibm.ca
Position: student	student

Delete Update

OK

We can see that the following employee data was returned from the database.

4. Open a Terminal and connect to the Sample database to view the ESQLEMPLOYEE table.

```
db2 connect to sample
db2 "SELECT * FROM ESQLEMPLOYEE"
```

```

db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2inst1@db2rules:~> db2 "SELECT * FROM ESQLEMPLOYEE"

USERNUMBER  USERID                PASSWORD                NAME
-----
1001 tttran                password                Tu Tran

1 record(s) selected.

db2inst1@db2rules:~> █

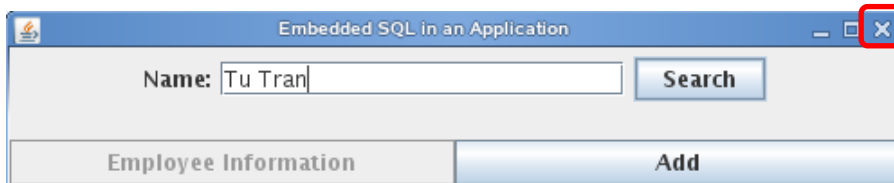
```

We can see that the data returned through the Embedded SQL application is the same as the data we see by directly connecting to the Sample database.

5. Go back to the Java program and press “OK” to close the window.



6. Press “X” to close the application.



7. Inserting Data

Using the INSERT statement is similar to using the SELECT statement except we do not need to store data in a ResultSet.

1. In the EmbeddedSQLConnection class, complete the addEmployee() function by uncommenting the code provided within addEmployee().



The addEmployee () function inserts new employee information into the table ESQLEMPLOYEE.

2. As before, an object of type PreparedStatement is specified to store the SQL statement.

```
PreparedStatement s = null;
```

3. The SQL statement is coded as the following String.

```
String query = "INSERT INTO ESQLEMPLOYEE (userNumber, userID, password, name, address, city, postalCode, telephoneNumber, email, position) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
```

4. We can now create the PreparedStatement using the Connection object con and the prepareStatement() method.

```
s = con.prepareStatement(query);
```

5. Notice the INSERT statement has several parameter markers. As before, they are necessary to associate values provided by the user to the SQL statement being executed. A command like the one below is used to associate a value to one of the parameters.

```
s.setString(2, userID);
```

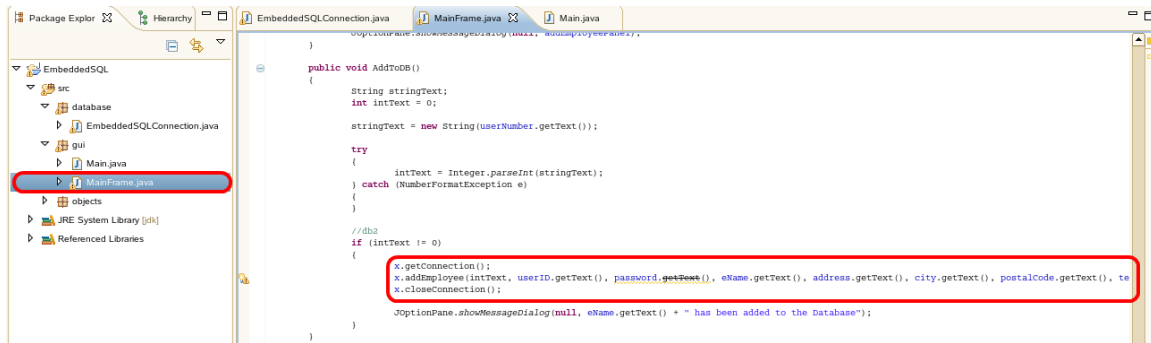
6. The statement can now be executed.

```
s.execute();
```

7.1 Incorporating INSERT with the Application

We have created functions to create and close a connection to DB2 as well insert data using an INSERT statement. How can we use these functions in our application?

1. In the GUI package, open the MainFrame class.



The MainFrame class is where we will be using the functions created to interact with DB2. This class contains all the functions in the application that are responsible for allowing the user to interact with the application.

2. Go to the AddToDB() function and uncomment the code provided.


```

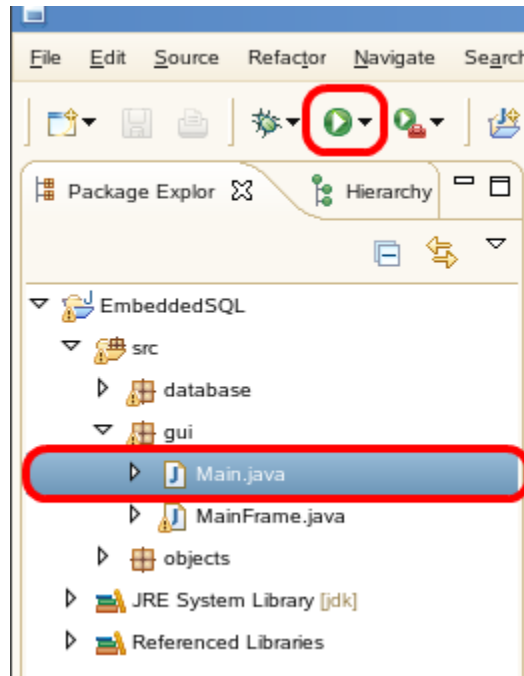
//6.1 Incorporating INSERT with the Application
x.getConnection();
x.addEmployee(intText, userID.getText(), password.getText(),
eName.getText(), address.getText(), city.getText(),
postalCode.getText(), telephoneNumber.getText(), email.getText(),
position.getText());
x.closeConnection();

```

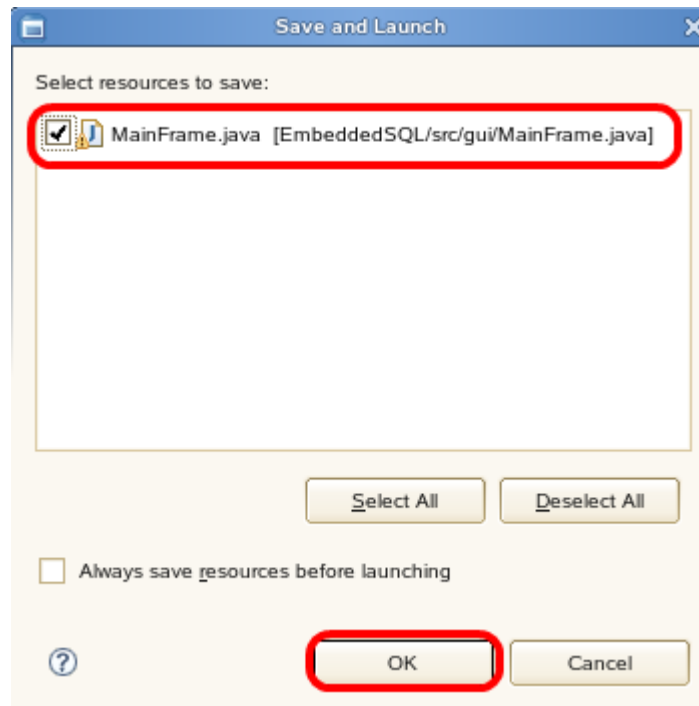
7.2 Insert into the Database using the Application

The application is now capable of performing an INSERT statement to add new employees to the database.

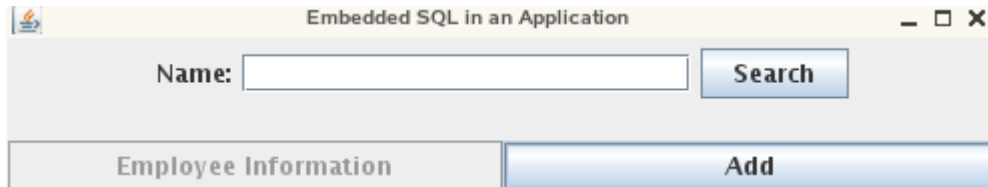
1. In the gui package, open the Main class and press .



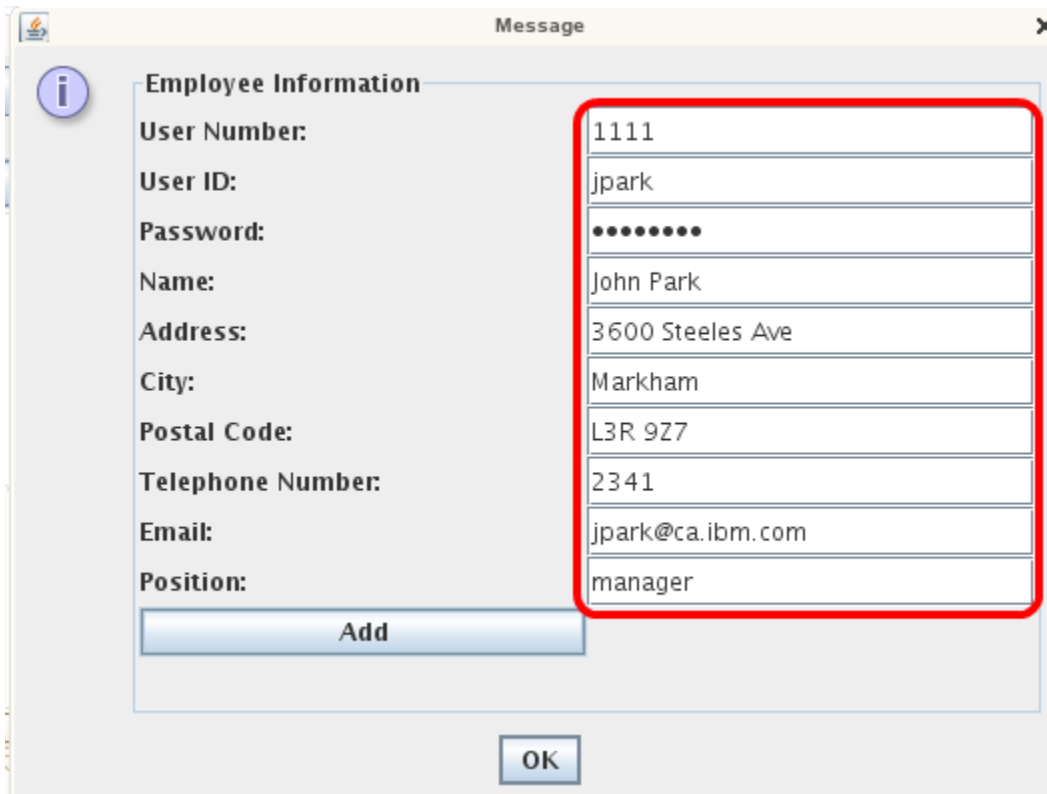
2. If you have not saved the changes, you will be prompted to save the file. Select all resources that need to be saved and press **OK**.



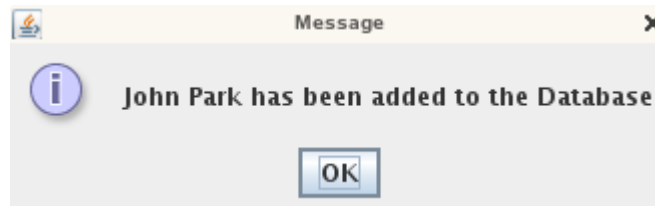
3. The following program will appear. Press the Add button.



4. The following popup will appear. Enter the information as seen below and press Add.



We can see that the employee data was successfully added to the database. Press "OK" to close the popup message.



5. Open a Terminal and connect to the SAMPLE database to view the ESQLEMPLOYEE table.

```
db2 connect to sample
db2 "SELECT * FROM ESQLEMPLOYEE"
```

```
File Edit View Terminal Tabs Help
db2inst1@db2rules:~> db2 connect to sample

Database Connection Information

Database server          = DB2/LINUX 9.7.1
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

db2inst1@db2rules:~> db2 "SELECT * FROM ESQLEMPLOYEE"

USERNUMBER  USERID          PASSWORD          NAME
-----
1001 tttran          password          Tu Tran
1111 jpark           password          John Park

2 record(s) selected.

db2inst1@db2rules:~> █
```

We can see that the employee “John Park” has been added to the Sample database.

Congratulations! You have just created a simple application capable of interacting with DB2. In this exercise we learned how to retrieve and insert data from a database using the JDBC API. Included with the application, there are also functions for the DELETE and UPDATE statements. Feel free to read through the code for a better understanding of how we can use JDBC within a Java application.



VMware[®] Basics and Introduction

Information Management Ecosystem Partnerships

IBM Canada Lab

Contents

1. VMWARE BASICS AND INTRODUCTION	2
2. HOW TO OBTAIN VMWARE SOFTWARE?	3
3. UNPACKING THE IMAGE	4
4. USING THE VMWARE VIRTUAL MACHINE	4
4.1 OPEN THE VIRTUAL MACHINE IN VMWARE.....	4
4.2 START THE VIRTUAL MACHINE	5
4.3 LOGIN TO THE VIRTUAL MACHINE AND ACCEPT THE LICENSE AGREEMENT	6
4.4 START THE GRAPHICAL USER INTERFACE	6
4.5 OPEN THE TERMINAL WINDOW	7
4.6 CLOSE THE TERMINAL WINDOW.....	8

1. VMware Basics and Introduction

The VMware® Player and VMware Workstation are the synonym for test beds and developer environments across the IT industry. While having many other functions for this specific purpose it allows the easy distribution of an “up and running” Linux® system featuring latest DB2® 9.7 and WebSphere® Application Server technology right to anybody’s computer – be it a notebook, desktop, or server.

The VMware image can be deployed for simple demos and educational purposes or it can be the base of your own development and experiments on top of the given environment.

What is a VMware image?

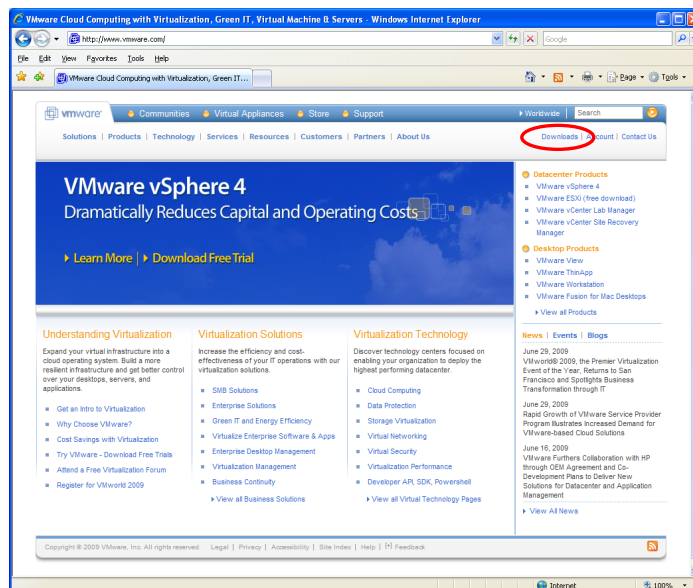
VMware is providing a virtual computer environment on top of existing operating systems on top of Intel® or AMD™ processor based systems. The virtual computer has all the usual components like a CPU, memory and disks as well as network, USB devices or even sound. The CPU and memory are simply the existing resources provided by the underlying operating system (you can see them as processes starting with “vmware....”). The disks are different. For the host operating systems they show up as a collection of files that can be copied between any system – even between Windows® and Linux flavors. Those virtual disk files make up the most part of the image while the actual description file of the virtual machine is very small.

The following will illustrate how to obtain VMware Player. Then, it will show you how to start the VMware image for the Hands-On Labs used in this technical session.

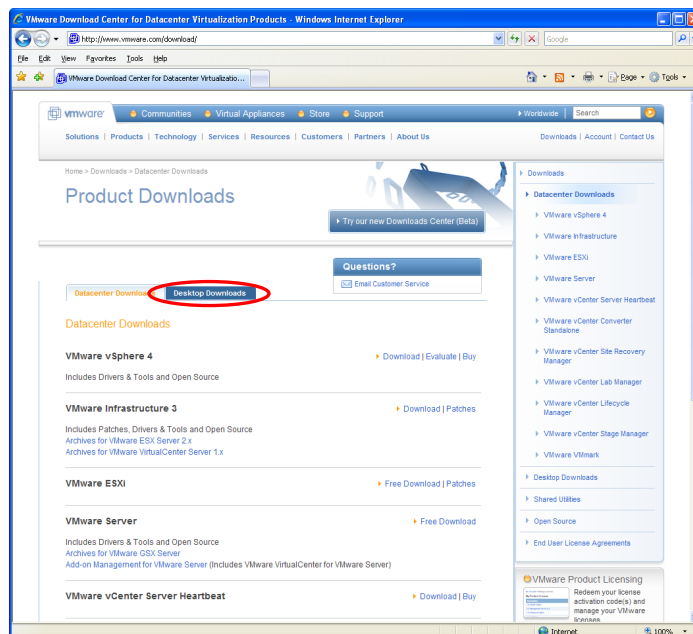
2. How to obtain VMware Software?

Open a web browser and visit www.vmware.com

Click on the Downloads link. Look for the Downloads link on the upper right hand corner of the page.



Click on the Desktop Downloads Tab.



Click on the product of your choice. We recommend VMware Player or VMware Workstation. Follow the on screen instructions for registration and download.

3. Unpacking the image

The image is delivered in a self-extractable set of rar files. For easy handling the files are compressed to 700MB volumes. Download all the volumes to the same directory.

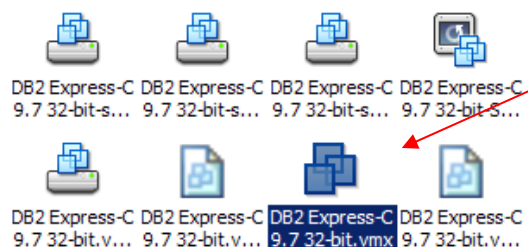
Double click the executable file and select the destination folder.

4. Using the VMware Virtual Machine

4.1 Open the Virtual Machine in VMware

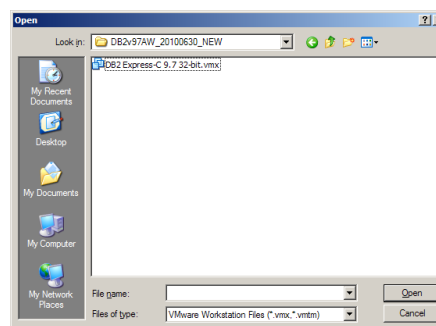
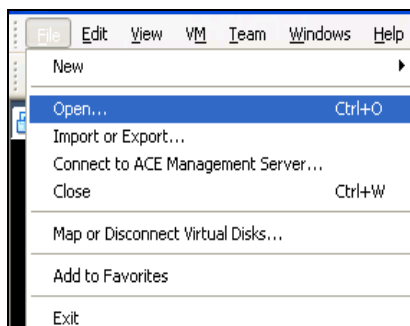
Starting the VMware virtual machine can happen through either way:

- Double click on the file “**DB2 Express-C 9.7 32-bit.vmx**” in your Windows Explorer or Linux file browser.

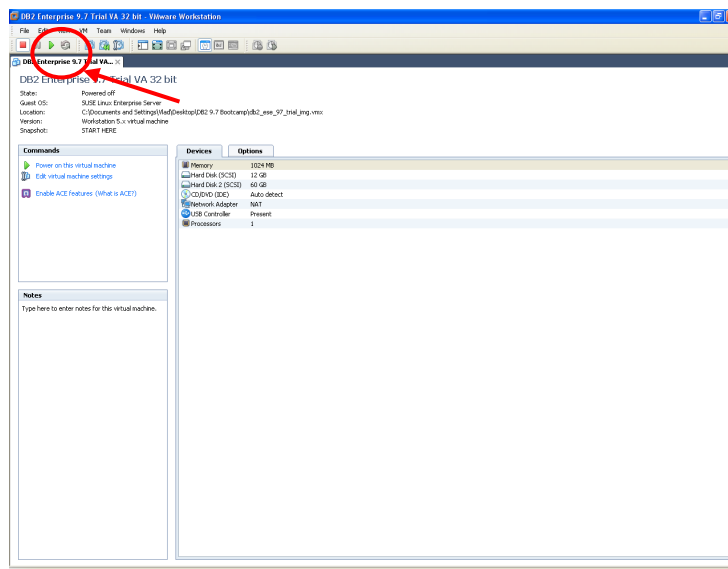


Or:


- Select it through the **File > Open...** icon in the VMware console.



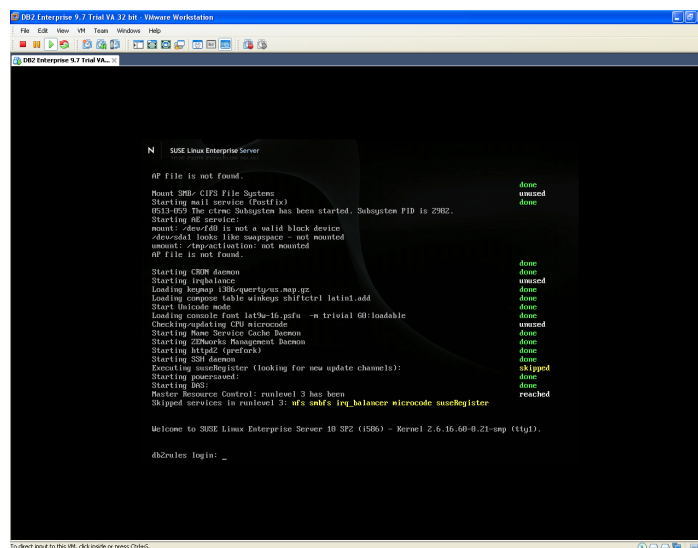
Either way should result in the screen below:




4.2 Start the Virtual Machine

Next the image can be booted up by pressing the “Power On”  button in the upper left side (marked in a red circle above).

The system will power up like any other Linux system and will come to the state as shown in the next picture below:



After the virtual machine has finished booting up, you can now work inside the virtual machine environment. To bring focus into the virtual machine environment, click inside the virtual machine screen with your mouse or click on the “Full Screen” button  in the toolbar on top of the VMware window.

After clicking on the screen, you may not see your mouse pointer anymore, this is normal as you are now operating in a command line mode inside the virtual machine. You can bring focus to the host operating system at any point by pressing “**Alt + Ctrl**” at the same time.

4.3 Login to the Virtual Machine and Accept the License Agreement

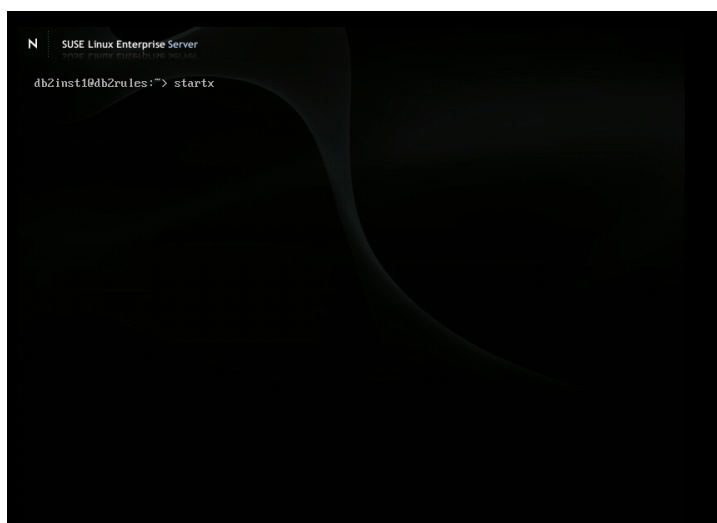
At this time, simply logon to the virtual machine at the command prompt with user “**db2inst1**” and password “**password**”.

You will see some pop-up messages asking you to read and accept the license agreement.

In order to use this image, you must accept all of the listed agreements and notices that were displayed. Select “**I accept**” to go to the next screen. If you do not agree with the license, select “**Abort**” and the virtual machine will be shutdown automatically.


You will be successfully login and presented with a login prompt:

```
db2inst1@db2rules:~>
```



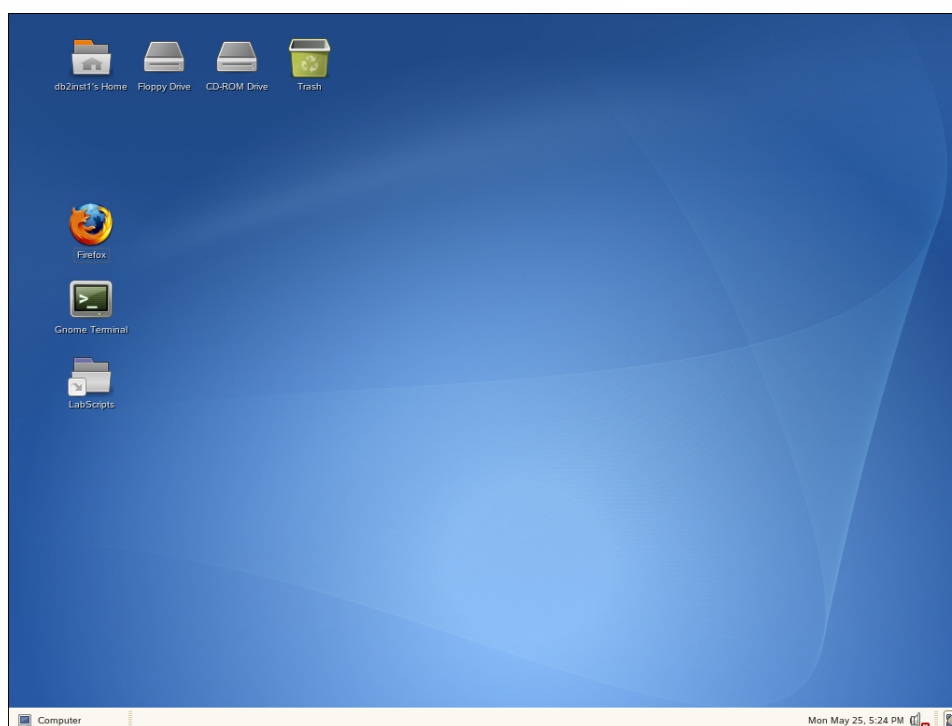
4.4 Start the Graphical User Interface

The virtual machine is capable of running in graphical mode as well. The default setup is configured for a 1024x768 pixel screen that provides good results for any given system today. Finer resolutions are possible but are not recommended since the text gets very small.


The system comes up with the following screen – still in the window of the VMware console and probably is showing scroll bars on the sides. Select the **“Full Screen”** button  on the console to switch to “full screen” mode any time – the result is significantly better. If you want to leaving the full screen mode, just press the combination **“Alt + Ctrl + Enter”** at the same time.



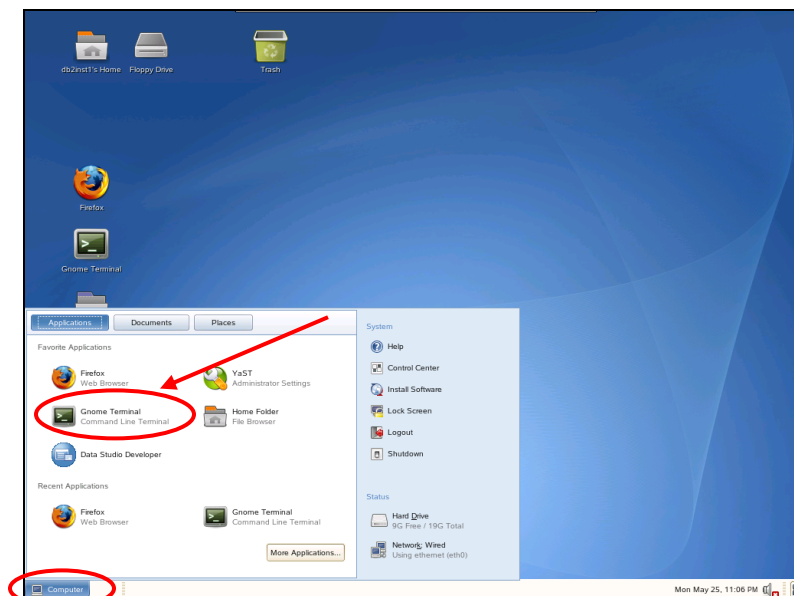
The full screen mode looks as follows:



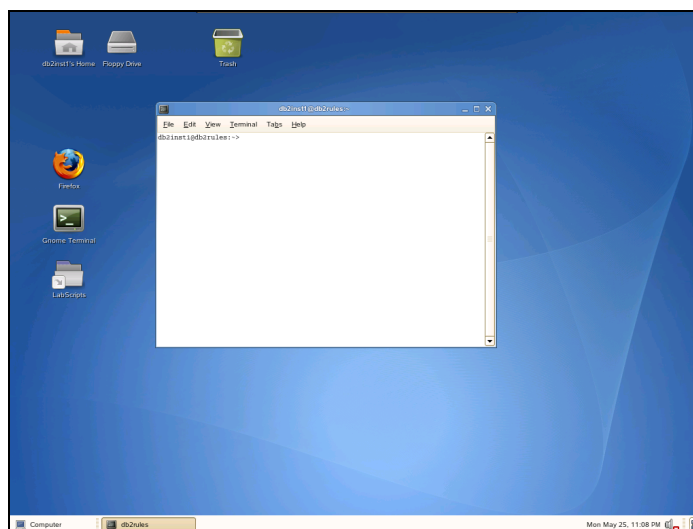
4.5 Open the Terminal Window

In order to execute commands, we will use the **Command Line Terminal**. To launch the terminal window, press the  menu at the bottom left corner of the screen,

and select  **Gnome Terminal** **Command Line Terminal**.



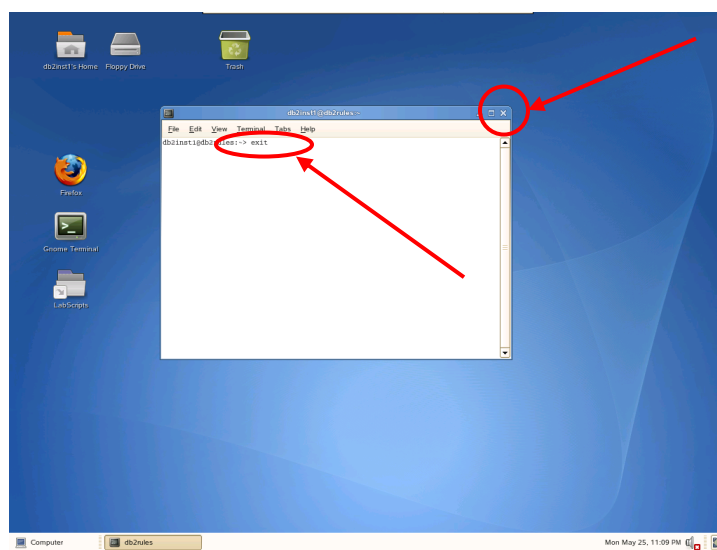
A terminal window similar to the above will pop up after you click on the icon.

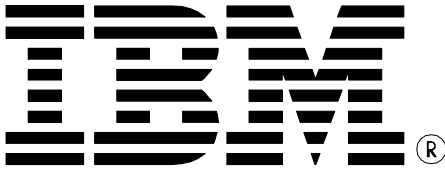


The terminal gives you a command line prompt and allows you to execute any commands using this prompt.

4.6 Close the Terminal Window

To close the terminal window, simply click on the “X” button on the top right hand corner of the terminal window, or type “**exit**” at the command prompt to exit out of the logged-in terminal. (**Note**, it might take multiple “**exit**” commands to logout of all logged-in sessions and close the terminal window if you have remote login or you are logged into a different user from this terminal window).





© Copyright IBM Corporation 2010
All Rights Reserved.

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Printed in Canada
07/07/2010

IBM, IBM (logo), and DB2 are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

VMware is a trademark of VMware Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates. The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

The information in this publication is provided AS IS without warranty. Such information was obtained from publicly available sources, is current as of July 2010, and is subject to change. Any performance data included in the paper was obtained in the specific operating environment and is provided as an illustration. Performance in other operating environments may vary. More specific information about the capabilities of products described should be obtained from the suppliers of those products.