DB2 Exam 610 Summary

Zeyuan Hu

January 19, 2016

Contents

1 Planning

1.1 Objectives

- Knowledge of DB2 products (z/OS vs LUW vs pureScale - at a high-level; different products and what they do)
- Knowledge of database workloads (appropriate DB2 product to use - OLTP vs warehousing)
- Knowledge of non-relational data concepts (XML data, LOB data)

1.2 Database workloads

Two main types of database application workloads:

- online transactional processing (OLTP)
- data warehousing
  - reporting
  - online analytical processing (OLAP)
  - data mining applications
  - decision support

1.3 OLTP vs. Data Warehousing

An OLTP system is typical of a web order system, where you perform transactions over the web (such as ordering a product). Online transaction processing (OLTP) systems features:

- Support day-to-day, mission-critical business activities (ie. web-based order entry, stock trading) [*current data*]
- Support hundreds to thousands of users issuing millions of transactions per day against databases that vary in size [*Frequent updates, Granular transactions*]
- Response time requirements tend to be subsecond [*Sub-second response time*]
- Queries:
  - tend to be a mix of real-time insert, update, and delete operations against current-as opposed to historical-data
  - single-row lookups with logic that likely updates a small number of records

Data warehousing system typically consist of:

- Store and manage large volumes of data that is often historical in nature and is used primarily for analysis [*Voluminous historical data*]
- Optimized for queries

- Heterogeneous data sources
- Queries: (ie. [*Summarized queries that perform aggregations and joins*])
  - bulk load operations
  - short-running queries
  - long-running complex queries
  - random queries
  - occasional updates to data
  - execution of online utilities

Example 1. *A database will be used primarily to identify sales patterns for products sold within the last three years and to summarize sales by region, on a quarterly basis. In case, a Data warehouse system is needed.*

Remark. *Different by queries that are typically used to access the data (aka workloads).*

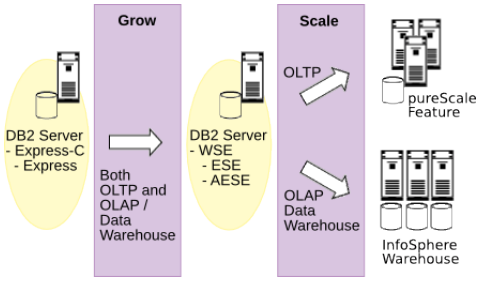

Figure 1: DB2 products, OLTP, Data Warehouse

1.4 DB2 pureScale - IBM solution for OLTP

System highlights:

- Best suited for OLTP workloads
- Enables a `DB2 for LUW` database to continously process incoming requests, even if multiple system components fail simultaneously, which makes it ideal for OLTP workloads where high availability is crucial
- Provides a database cluster solution for nonmainframe platforms
- Can **ONLY** work with the General Parallel File System (GFPS) file system

Usage:

- Can be used with `DB2 Workgroup Server Edition (WSE)`, `DB2 Enterprise Server Edition (ESE)`, `DB2 Advanced Enterprise Server Edition(AESE)`

- Can **ONLY** be installed on IBM p Series or x Series servers that are running either the AIX (p Series) or the Linux (x Series) operating system
- **CANNOT** be installed on IBM mainframes running z/OS, IBM p Series server running Linux, or IBM x Series servers running Windows

1.5 InfoSphere Warehouse - IBM solution for Data warehousing

System highlights:

- is a complete data warehousing solution that contains components that facilitate data warehouse construction and administration, as well as tools that enable embedded data mining and multidimensional online analytical processing (OLAP)

1.6 Notable DB2 features & products

IBM Data Studio

- is an Eclipse-based, integrated development environment (IDE) that can be used to perform instance and database administration, routine (SQL procedure, SQL functions, etc.) and application development, and performance-tuning tasks.
- replaces the `DB2 Control Center` as the standard GUI tool for DB2 database administration and application development.
- allows users to connect to a DB2 database using a wizard; however, users are required to provide login credentials before a connection will be established.
- components:
  - `IBM Data Studio administration client`
    - can be installed on servers running Red Hat Linux, SUSE Linux, and Windows
    - **CANNOT** be installed on AIX servers
  - `IBM Data Studio full client`
    - can be installed on servers running Red Hat Linux, SUSE Linux, and Windows
  - `IBM Data Studio web console`
    - can be installed on servers running Red Hat Linux, SUSE Linux, and Windows
    - can be installed on servers running the AIX operating system as well

IBM Workload Manager (WLM)

- is a comprehensive workload management feature that can help identify, manage, and control database workloads to maximize database server throughput and resource utilization
- cutomize execution environments for the purpose of controlling system resources so that no single workload can control and consume all of the system resources available. (This prevents any one department or service class from overwhelming the system.)

IBM InfoSphere Optim Performance Manager Extended Edition

- can be used to identify, diagnose, solve, and prevent performance problems in DB2 products and associated applications including Java and DB2 Call Level Interface (CLI) applications.

Self-Tuning Memory Manager (STMM)

- responds to significant changes in a database's workload by dynamically distributing available memory resources among several different database memory consumers

Connection Concentrator

- improves the performance of applications that require frequent, but relatively transient, simultaneous user connections by allocating host database resources only for the duration of an SQL transaction,

IBM InfoSphere Data Architect

- A complete solution for designing, modeling, discovering, relating, and standardizing data assets.
- You can use it for data modeling, transformation, and DDL generation, and to build, debug, and manage database objects such as SQL stored procedures and functions.

IBM InfoSphere Optim Query Tuner (Query Tuner)

- can analyze and make recommendations on ways to tune existing queries, as well as provide expert advice on writing new queries.

IBM InfoSphere Optim pureQuery Runtime

- Lets you deploy advanced pureQuery applications that use static SQL for a wide range of benefits.
- Briges the gap between data and Java technology by harnessing the power of SQL within an easy-to-use Java data access platform.
- Increases security of Java applications helping to prevent threats like SQL injection.

DB2 for i

- combines with `IBM BLU Acceleration` to handle Analytical workloads
- formerly known as DB2 for i5/OS, is an advanced, 64-bit Relational Database Management System that leverages the high performance, virtualization, and energy efficiency features of IBM's Power Systems
- its self-managing attributes, security, and built-in analytical processing functions make `DB2 for i` an ideal database server for applications that are analytical in nature

DB2 pureXML

- offers a simple and efficient way to create a "hybrid" DB2 database that allows XML data to be storded in its native, hierarchical format.

Data Partitioning Feature (DPF)

- provides the ability to divide very large databases into multiple parts (known as partitions) and store them across a cluster of inexpensive servers.

1.7 DB2 offering

DB2 for z/OS

- full-function database management system that has been designed specifically for z/OS, IBM's flagship mainframe operating system.
- Tightly integrated with the IBM mainframe, `DB2 for z/OS` leverages the strengths of System z 64-bit architecture to provide, among other things, the ability to support complex data warehouse.

1.8 Large Objects (LOB)

LOB data types-**not LOB locators**-are used to store binary data values in a DB2 database.

- By default, LOB data is stored in separate LOB storage objects.
- Changes to LOB data are not recorded in transaction log files.

Inline LOBs

- improve query performance by storing LOB data in the same data pages as the rest of a table's rows, rather than in a separate LOB storage object.Thus, no additional I/O is needed to store and access this type of data.
- is eligible for compression.
- When a table contains columns with inline LOBs, fewer rows can fit on a page.
- transactions that modify inline LOB data are always logged. Consequently, the use of inline LOBs can **increase** logging overhead.
- are created by appending the `INLINE LENGTH` clause to a LOB column's definition.

LOB locator

- represents a value for a LOB resource that is stored in a database
- is a simple token value that is used to refer to a much bigger LOB value
- is a mechanism that refers to a LOB value from within a transaction
- is **NOT** a data type, nor is it a database object
- **do NOT** store copies of LOB data-they store a description of a base LOB value, and the actual data that a LOB locator refers to is only materialized when it is assigned to a specific location, such as an application host variable or another table record
- they behave as a snapshot of a piece of an LOB value, and not as a pointer to a row or a location in the database

1.9 XML data

- with `pureXML`, XML documents are stored in tables that contain one or more columns that have been defined with the XML data type.
- `CREATE TABLE employee (empid INT, resume XML)`

2 Security

2.1 Objectives

- Knowledge of restricting data access
- Knowledge of different privileges and authorities
- Given a DCL SQL statement, knowledge to identify results (grant/revoke/connect statements)
- Knowledge of Row and Column Access Control (RCAC)
- Knowledge of Roles and Trusted Contexts

Three levels of security:

- level 1: control access to the instance under which a database was created
- level 2: control access to the database itself
- level 3: control access to the data and data objects reside within the database

2.2 Authentication

- is the process by which a system verifies a user's identity.
- normally, an external facility (ie. OS, DCE) that is not pat of DB2 performs the authentication.
- *authentication type* for a server is a database manager configuration parameter to decide how and where users are authenticated.

Type	Description
SERVER	➔ Authentication occurs on the server
SERVER_ENCRYPT	➔ Authentication occurs on the server ➔ Passwords are encrypted at the client machine before being sent to the server
CLIENT	➔ Authentication occurs at the client workstation, with no further checks on the server
KERBEROS	➔ Authentication is performed by the Kerberos security software
KRB_SERVER_ENCRYPT	➔ Authentication is performed by Kerberos security software if the client's authentication type is set to KERBEROS. Otherwise, SERVER_ENCRYPT is used
DATA_ENCRYPT	➔ SERVER_ENCRYPT ➔ user data are encrypted
DATA_ENCRYPT_CMP	➔ DATA_ENCRYPT ➔ Use SERVER_ENCRYPT if DATA_ENCRYPT not supported
GSSPLUGIN	➔ Authentication is controlled by an external GSS-API plugin
GSS_SERVER_ENCRYPT	➔ GSSPLUGININ ➔ Use SERVER_ENCRYPT if GSSPLUGIN not supported

2.3 Authorities

- convey the right to perform high-level administrative and maintenance/utility operations on an instance or a database
- instance-level authorities:
  - System Administrator (**SYSADM**) authority:
    - Highest level of administrative authority at the instance level
    - Only a user with SYSADM authority can perform the following:
      - Upgrade a database
      - Restore a database
      - Change the database manager configuration file (including specifying the groups having SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority)
      - Grant and revoke table space privileges and also use any table space
      - Perform any SQL or XQuery operation that does not attempt to access data that is protected by RCAC or LBAC.
    - When a user with SYSADM authority creates a database, that user is automatically granted ACCESSCTRL, DATAACCESS, DBADM, SECADM authority on the database.
  - Installation System Administrator (Installation SYSADM) authority:
    - Conveys the same set of abilities that SYSADM authority provides.
    - Can perform recovery operations when the system catalog for a database is inaccessible or unavailable.
  - System Control (**SYSCTRL**) authority:
    - Highest level of system and instance control authority
    - Intended to provide select users with nearly complete control of a DB2 system without letting them access sensitive data
    - Similar to SYSADM but cannot access any data within the databases unless they are granted the privileges required to do so.
    - Commands a SYSCTRL user can perform against any database in the instance are:
      - `db2start/db2stop`
      - `db2 create/drop database`
      - `db2 create/drop tablespace`
      - `db2 backup/restore/rollforward database`
      - `db2 runstats` (against any table)
      - `db2 update db cfg for database` dbname
  - System Operator (SYSOPER) authority:
    - the ability to execute all DB2 commands available *except* ARCHIVE LOG, START DATABASE, STOP DATABASE, and RECOVER BSDS.
    - run the DSN1SDMP utility, and terminate any running utility job.
  - Installation System Operator (Installation SYSOPER) authority:

    * SYSOPER authority
    * Perform select operations when the system catalog for a database is unavailable.
- System Maintenance (**SYSMAINT**) authority:
    * SYSMAINT users can only perform tasks related to mainenance (subset of SYSCTRL authority):
        · `db2start/db2stop`
        · `db2 backup/restore/rollforward database`
        · `db2 runstats` (against any table)
        · `db2 update db cfg for database` dbname
    * users with SYSMAINT **CANNOT** create or drop databases or tablespaces.
    * They cannot access any data within the databases unless they are granted the explicit privileges required to do so.
- System Monitor (**SYSMON**) authority:
    * Provides the ability to take database system monitor snapshots of an instance and its databases.
    * SYSMON authority enables the user to run the following commands:
        · `GET DATABASE MANAGER MONITOR SWITCHES`
        · `GET MONITOR SWITCHES`
        · `GET SNAPSHOT`
        · `LIST ACTIVE DATABASES`
        · `LIST APPLICATIONS`
        · `LIST DATABASE PARTITION GROUPS`
        · `LIST DCS APPLICATIONS`
        · `LIST PACKAGES`
        · `LIST TABLES`
        · `LIST TABLESPACE CONTAINERS`
        · `LIST TABLESPACE`
        · `LIST UTILITIES`
        · `RESET MONITOR`
        · `UPDATE MONITOR SWITCHES`
    * following APIs:
        · `db2GetSnapshot` - Get snapshot
        · `db2GetSnapshotSize` - Estimate size required for `db2GetSnapshot` output buffer
        · `db2MonitorSwitches` - Get/update monitor switches
        · `db2ResetMonitor` - Reset monitor
        · `db2mtrk` - Memory tracker
    * Users with the SYSADM, SYSCTRL or SYSMAINT authority level also possess SYSMON

- Database-level authorities:
  - Database Administrator (**DBADM**) authority:
    - DBADM users **CANNOT** perform such maintenance or administrative tasks as:
      - `db2 drop database`
      - `db2 drop/create tablespace`
      - `db2 backup/restore database`
      - `db2 update db cfg for database` dbname
    - DBADM users can perform the following tasks:
      - `db2 drop/create table` (index, views)
      - `db2 grant/revoke` (any privilege)
      - `db2 runstats` (any table)
    - access data stored in tables, views, including system catalog tables and views-provided that data is not protected by RCAC or LBAC
  - Database Control (DBCTRL) authority:
    - create database objects
    - issue database-specific DB2 commands
    - run DB2 utilities (*including those that change data*)
    - terminate any running utility *except* DIAGNOSE, REPORT, and STOSPACE
  - Database Maintenance (DBMAINT) authority:
    - create database objects
    - issue database-specific DB2 commands
    - run DB2 utilities that do not change data
    - terminate any running utility *except* DIAGNOSE, REPORT, and STOSPACE
  - Package Administrator (PACKADM) authority:
    - BIND, COPY, and EXECUTE privileges on all packages in one or more specific collections
    - BIND subcommand to create new packages in certain collections
  - System Database Administrator (System DBADM) authority:
    - create, alter and drop database objects
    - issue database-specific DB2 commands
    - run the following DB2 utilities:
      `CHECK INDEX, CHECK LOB, COPY, COPYTOCOPY, DIAGNOSE, MODIFY RECOVERY, MODIFY STATISTICS, QUIESCE, REBUILD INDEX, RECOVER, REPORT, RUNSTATS`
    - access and modified data stored in system catalog tables and views
    - **cannot** access user data
    - **cannot** grant and revoke authorities and privileges
  - Security Administrator (**SECADM**) authority:
    - can only be granted by a SYSADM user
    - CANNOT access user data and create databases

    * can perform the following:
        - Create and drop security label components
        - Create and drop security policies
        - Create and drop security labels
        - Grant and revoke security labels
        - Grant and revoke LBAC rule exemptions
        - Grant and revoke setsessionuser privileges
        - Grant and revoke database privileges and authorities
        - Execute the SQL statement `TRANSFER OWNERSHIP` on objects that you do not own
        - Execute the following audit routines:
            1. `SYSPROC.AUDIT_ARCHIVE` used to archive audit logs
            2. `SYSPROC.AUDIT_LIST_LOGS` used to locate audit files present in a specific directory
            3. `SYSPROC.AUDIT_DELIM_EXTRACT` used to extract audit data to delimited files format
    * No other user can perform these functions, not even the SYSADM, unless SECADM was explicitly granted to that SYSADM user
- Access Control (ACCESSCTRL) authority:
    * can only be granted by SECADM
    * cannot grant to PUBLIC group
    * can access and modify data stored in system catalog tables and views
    * cannot access or modify user data
    * issue following GRANT (and REVOKE) statements:
        - GRANT (Database Authorities). Does not give the holder the ability to grant ACCESSCTRL, DATAACCESS, DBADM, or SECADM authority. Only SECADM can grant these authorities.
        - GRANT (Global Variable Privileges)
        - GRANT (Index Privileges)
        - GRANT (Module Privileges)
        - GRANT (Package Privileges)
        - GRANT (Routine Privileges)
        - GRANT (Schema Privileges)
        - GRANT (Sequence Privileges)
        - GRANT (Server Privileges)
        - GRANT (Table, View or Nickname Privileges)
        - GRANT (Table Space Privileges)
        - GRANT (Workload Privileges)
        - GRANT (XSR Object Privileges)
- Data Access (DATAACCESS) authority:

    - can be granted only by SECADM
    - cannot be granted to PUBLIC
    - provides the following privilege and authorities:
        - LOAD authority
        - SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
        - EXECUTE privilege on packages
        - EXECUTE privilege on modules
        - EXECUTE privilege on routines *except* AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT
        - READ privilege on all global variables and WRITE privilege on all global variables except variables that are read-only
        - USAGE privilege on all XSR objects
        - USAGE privilege on all sequences
- SQL Administrator (SQLADM) authority:
    - Monitor and tune SQL statements
    - granted by ACCESSCTRL and SECADM authority
    - can perform the following:
        - EXPLAIN SQL statements and PROFILE commands
        - run the RUNSTATS and MODIFY STATISTICS utilities
        - execute system-defined stored procedures, functions, and packages
        - DB2 for LUW can also run the following:

          ```
          CREATE EVENT MONITOR, DROP EVENT MONITOR, FLUSH EVENT MONITOR
          FLUSH OPTIMIZATION PROFILE CACHE, FLUSH PACKAGE CACHE,
          PREPARE, REORG, SET EVENT MONITOR STATE
          ```
- Workload Management Administrator (WLMADM) authority:
    - manage workload management objects (service classes, work action sets, work class sets, workloads)
    - granted by ACCESSCTRL or SECADM authority

Remark. *DB2 for LUW only:*

```
SYSMAINT, SYSMON, WLMADM
```

DB2 for z/OS:

```
Installation SYSADM, SYSOPER, INSTALLATION SYSOPER, DBCTRL,
DBMAINT, PACKADM, System DBADM
```

2.4 Privileges

- database-level privileges, which span all objects within the database
  - DB2 for LUW
    - BINDADD: create packages in the database using the BIND command
    - CONNECT: connect to the database
    - CREATETAB: create tables within the database
    - CREATE_EXTERNAL_ROUTINE: create a procedure for use by applications and other users of the database
    - CREATE_NOT_FENCED_ROUTINE: create unfenced user-defined functions (UDFs)
    - EXPLAIN: generate Explain query plans
    - IMPLICIT_SCHEMA: implicitly create schemas within the database without using the CREATE SCHEMA command
    - LOAD: load data into a table
    - QUIESCE_CONNECT: access a database while it is in a quiesced state
  - DB2 for z/OS
    - CREATETAB: create tables within the database
    - CREATETS: create table spaces for database
    - DISPLAYDB: display the status of a database
    - DROP: drop or alter a database
    - IMAGCOPY: prepare for, make, and merge copies of table spaces in a database; remove records of any table space copies made
    - LOAD: load data into a database
    - RECOVERDB: recover objects in a database and report recovery status
    - REORG: reorganize objects in a database (run REORG utility)
    - REPAIR: generate diagnostic information about and repair data stored in a database's objects
    - STARTDB: start database
    - STATS: gather statistics; check index and referential constraints for associated objects; delete unwanted statistics history records from the system catalog tables
    - STOPDB: stop database
- object privilege, apply to specific database objects: DB2 z/OS only; DB2 LUW only; Both

Privilege name	Relevant objects	Description
CONTROL	table, view, index, package,nickname	Provides full authority on the object. Users with this privilege can also grant or revoke privileges on the object to other users
DELETE	table, view, nickname	allows a user to remove data from object
INSERT	table, view, nickname	allows a user to add data into object
SELECT	table, view, nickname	allows a user to retrieve data from the object

UPDATE	table, view, nickname	allows a user to modify data within the object
ALTER	table, sequence, nickname	allows a user to alter the object definition, comment associated with
INDEX	table, nickname	allows a user to create an index on object
REFERENCES	table, nickname	provides the ability to create or drop foreign key constraints on the object
BIND	package, plan	allows a user to rebind(recreate) existing packages
COPY	package	allows a user to copy a certain package
EXECUTE	function, stored procedure, routine, package, plan	allows a user to invoke object
USAGE	sequence, jar, XSR, workload	allows a user to use PREVIOUS VALUE and NEXT VALUE associated with sequence/use a jar file/use a XSR object /use a workload
USAGE OF	TYPE, DISTINCT TYPE	allows a user to use object
TRIGGER	table	allows a user to create triggers for object
ALTERIN	schema	allows a user to change the comment associated with any object in a schema or modify definitions of objects in a schema
CREATEIN	schema	allows a user to create objects within a schema
CREATE IN	collection	allows a user to name a collection, execute the BIND PACKAGE subcommand
DROPIN	schema	allows user to drop objects within a schema
SETSESSIONUSER		set the session authorization ID to one of a set of specified authorization IDs available
USE OF	BUFFERPOOL, ALL BUFFERPOOLS, TABLESPACE, STORAGEGROUP	allows user to use the object
ARCHIVE		allows a user to archive active log
BINDADD		allows a user to create new plans and packages
BINDAGENT	plan, package	allows a user to bind, rebind, or free object
BSDS		recover the bootstrap data set
CREATEALIAS	table, view	allows a user to create alias for object
CREATEDBA		allows a user to create a new database and have DBADM authority over it
CREATEDBC		allows a user to create a new database and have DBCTRL authority over it
CREATESG		allows a user to create a storage group
CREATE_SECURE_OBJECT		allows a user to create secure triggers or UDFs
CREATETMTAB		allows a user to define a created temporary table
DEBUGSESSION		allows a user to control debug session activity for stored procedures, functions

DISPLAY		allows a user to display system information
EXPLAIN		allows a user to generate Explain query plans
MONITOR1		allows a user to receive trace data
MONITOR2		allows a user to receive trace data regardless of its sensitivity
RECOVER		allows a user to recover threads
STOPALL		allows a user to stop DB2
STOSPACE		allows a user to obtain information about storage space usage
TRACE		allows a user to control tracing
PASSTHRU		allows a user to issue DDL and DML directly to a data source via a federated database server
READ		allows a user to read the value of a certain global variable
WRITE		allows a user to assign a value to a certain global variable

Remark. • *Objects that can be manipulated within a schema:*

- *DB2 for LUW: tables, views, index, packages, data types, functions, triggers, procedures, alias*
- *DB2 for z/OS: distinct data types, UDFs, triggers, procedures*

2.5 Granting/Revoking Authorities and Privileges

- Implicitly:
  - DB2 may grant privileges automatically when certain commands are issued without the need for an explicit GRANT statement being issued.
- Indirectly:
  - When a user executes a package that performs operations that require certain privileges (ie. a package that deletes a row of data from a table will require DELETE privilege on the table), he or she is indirectly given those privileges for the express purpose of executing the package.
- Explicit:
  - To explicitly grant authorities and privileges, a user must possess SECADM, ACCESSCTRL, or CONTROL privilege on the object that privileges are to be granted for
- GRANT statement:

  - if the `ALL PRIVILEGES` clause is specified with the `GRANT` statement used, all authorities and privileges for the designated object- *except* the CONTROL privilege- will be granted to each recipient indicated.
  - CONTROL privilege must be granted separately.
  - if the `GRANT OPTION` clause is specified with the `GRANT` statement used, the individual receiving the designated authorities and privileges will receive the ability to grant those authorities and privileges to others.
  - if the `WITH ADMIN OPTION` clause is specified with the `GRANT` statement used, the individual being granted will receive the ability to grant that role to others.

- REVOKE statement:
- Roles:
  - is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement.

2.6 Row and Column Access Control (RCAC)

- RCAC controls access to a table at the row level, column level or both.
- can use RCAC to ensure that your users have access to only the data that is required for their work.
- Regular SQL privileges cannot restrict access to portions of a table. This was usually done through views or application logic. However, users with direct access to the database can bypass these layers.
- With RCAC, even higher level authorities such as users with DATAACCESS authority are not exempt from RCAC rules. Only users with SECADM authority can manage RCAC within a database. Thus, you can use RCAC to prevent users with DATAACCESS from freely accessing all data in a database.
- RCAC rules: RCAC is compromised of SQL rules that place access control at the table level around the data itself. RCAC permits all users to access the same table. But, RCAC restricts access to the table based upon individual user permissions or roles as specified by a policy associated with the table
  - Row permissions
    * Row permission is a database object that expresses a row access control rule for a specific table. A row access control rule is an **SQL search condition** that describes what set of rows a user has access to.
  - Column masks
    * Column mask is a database object that expresses a column access control rule for a specific column in a table. A column access control rule is an **SQL CASE expression** that describes what column values a user is permitted to see and under what conditions

2.7 Trusted contexts

- is a database object that defines a trust relationship for a connection between database and an external entity such as an application server.
- the following information is used to define a trusted context:
  - **System authorization ID** - Represents the user that establishes a database connection
  - **IP address (or domain name)** - Represents the host from which a database connection is established
  - **Data stream encryption** - Represents the encryption setting (if any) for the data communication between the database server and the database client
- When a user establishes a database connection, the DB2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.
- trusted context objects can only be defined by SECADM
- implicit trusted connection
  - results from a normal connection request and allows users to inherit a role that is unavailable to them outside the scope of the trusted connection
- explicit trusted connection
  - is established by making a connection request within an application.
  - can switch the connection's user to a different authorization ID

2.8 Label-Based Access Control (LBAC)

3 Working with Databases and Database Objects

3.1 Objectives

- Ability to create and connect to DB2 servers and databases (requirements to give ability to connect)
- Ability to identify DB2 objects
- Knowledge of basic characteristics and properties of DB2 objects
- Given a DDL SQL statement, knowledge to identify results (ability to create objects)
- Knowledge of Temporal (Time Travel) Tables-System-period, Application-period, and Bi-temporal- ability to create (Greater precision time stamps)

3.2 Basic DB2 organization

- servers
- instances
- databases

3.3 DB2 Objects

3.3.1 Data objects

- Schemas
  - A way to logically group objects in a database (organize data objects into sets)
  - `CREATE TABLE HR.EMPLOYEES ...`
  - When table spaces, tables, index, *distnct data types*, functions, stored procedures, and triggers are created, they are automatically assigned to (or defined into) a schema, based upon the qualifier that was provided as part of the user-supplied name.
- Tables
  - Base tables (regular tables)
  - Multidimensional clustering (MDC) tables
    - are physically clustered on more than one key or dimension simultaneously.
  - Insert time clustering (ITC) tables
    - are used to cluster data according to the time in which rows are inserted
  - Ranged-clustered tables (RCTs)
  - Partitioned tables
  - Temporal tables
    - application-period temporal tables: used to track effective dates for data that is subject to changing business conditions

    - system-period temporal tables
    - bitemporal tables
  - Auxiliary tables
  - Clone tables
  - History tables
  - Result tables
  - Materialized query tables (MQTs)
    - improve the execution performance of qualified SELECT statements
    - derive their definitions from the results of a query (SELECT statement)
    - their data consists of precomputed values taken from one or more tables the MQT is based upon.
    - MQTs can greatly improve performance and response time for complex queries, particularly queries that aggregate data over one or more dimensions or that join data across multiple base tables.
  - Temporary tables
  - Declared global temporary tables
    - are used to hold nonpresistent data temporarily, on behalf of a single application
  - Created global temporary tables
  - Typed tables

  Remark. *Base tables, temporary tables, and indexes can be enabled for data compression.*

- Views
  - do not contain data (only a view's definition is stored in a database)
  - can be derived from other views
  - view can be defined as being *insertable*, *updatable*, *deletable*, or *read-only*
  - used to control access to sensitive data (each see different presentations of data that resides in the same table)

- Indexes
  - an object that contains pointers to rows in a table that are logically ordered according to the values of one or more columns (known as keys)
  - Usage:
    - fast, efficient method for locating specific rows of data in large tables
    - logical ordering of the rows in a table
    - enforce the uniqueness of records in a table
    - force a table to use *clustering* storage, which causes the rows of a table to be physically arranged according to the ordering of their key column values.

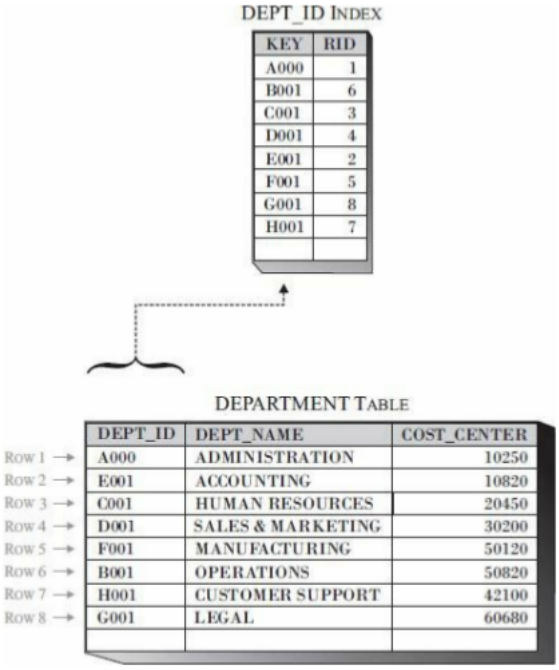

Figure 4.5: A simple index that has a one-column key

Figure 2: Index that has a one-column key

  - tables that are used for data mining, business intelligence, business warehousing, and by applications that execute many (and often complex) queries but that rarely modify data are prime candidates for index.
  - tables in OLTP or environments where data throughput is high should use index sparingly or avoid them altogether.

- Aliases
  - Similar to table or view, but cannot be used in the check condition of a check contraint or to reference a user-defined temporary table.
  - Usage: allow to construct SQL statements in such a way that they are independent of the base tables or views they reference.

- Sequences
  - PREVIOUS VALUE, NEXT VALUE
  - Example scenario: An application running on a remote client needs to ensure that every new employee that joins the company is assigned a unique, sequential employee number.
  - "Change whether a sequence cycles", "establish new minimum and maximum sequence values", and "change the number of sequence numbers that are cached" can be done with ALTER SEQUENCE.

  - "Change a sequence's data type" CANNOT done with ALTER SEQUENCE.

- Triggers
  - an object that 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.
  - Types:
    * BEFORE triggers - The trigger's actions occur just before the triggering event takes place.
    * AFTER triggers - The trigger's actions occur immediately after the triggering event takes place
    * INSTEAD OF triggers - are executed in place of the trigger event (to ensure that applications can perform insert, update, delete and query operations against an updatable view only)
  - Example scenario: An application running on a remote client needs to track every modification made to a table that contains sensitive data.

- User-defined data types (UDTs)
  - distinct data type - derived from one of the built-in data types that are provided with DB2
  - structured data type - contains multiple attributes, each of which has a name and data type of its own

- User-defined functions (UDFs)
  - Example scenario: An application running on a remote client needs to be able to convert degress Celsius to degrees Fahrenheit and vice versa.
  - category"
    * SQL
    * Sourced (Template): A function that is based on some other function that already exists.
    * External Scalar: A function that is written using a high-level programming language.
    * External Table: A function that returns a result data set in the form of a table.
    * OLE DB External Table: A function that can access data from on Object Linking and Embedding Database (OLE DB) provider and return a result data set in the form of a table.

- Stored procedures
  - Example scenario: An application running on a remote client needs to collect input values from a user, perform a calculation using the values provided, and store the input data, along with the calculation results, in a base table.
  - Category
    * SQL: body is written entirely in SQL or SQL PL.

        * External SQL: body is written entirely in SQL, but that DB2 supports by generating an associate C program for.
        * External: body is written in a high-level programming language.

- Packages

3.3.2 System objects

- Buffer pools
  - One buffer pool is created automatically as part of the database creation process
  - Once a page has been copied to a buffer pool, it remains there until the space it occupies is needed
- Table spaces
  - Every table space must have a buffer pool assigned to it
  - DB2 for LUW:
    * System Managed Space (SMS)
    * Database Managed Space (DMS)
    * Automatic Storage (AS)
  - DB2 for z/OS:
    * Partitioned table space
    * Segmented table space
    * Universal table space
    * Large object (LOB) table space
    * XML table space
- The system catalog: DB2 updates the information stored in the system catalog whenever any of the following events occur:
  - Database objects (ie. tables, index, views) are created, altered, or dropped
  - Authorizations and privileges are granted or revoked
  - Statistical information is collected
  - Packages are bound to the database
- Transaction log files
  - write-ahead logging:
    * insert operation - a record for the new row is written to the log buffer
    * delete operation - a record containing the row's original values is written to the log buffer
    * update operation - a record containing the row's original data, together with the corresponding new data, is stored in the log buffer
- The DB2 directory

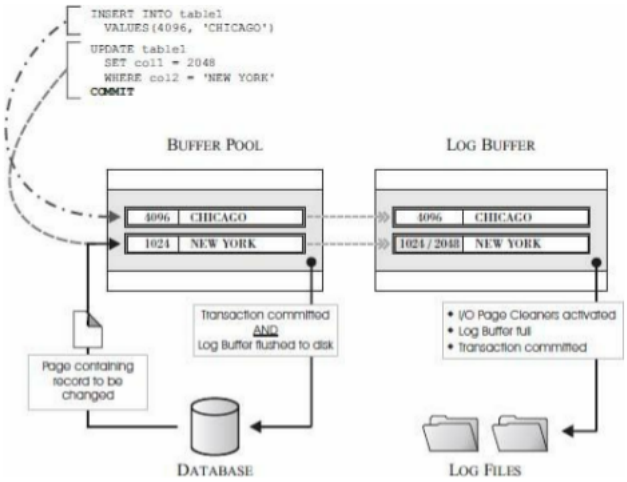

Figure 4.7: The transaction logging process

Figure 3: Transaction logging process

- The bootstrap data set
- The data definition control supports (DDCS) database
- Resource limit facility tables
- The work file database

3.4 Creating a DB2 Database

- DB2 for LUW:
  - Syntax:

```
CREATE [DATABASE | DB] [DatabaseName]
<AUTOMATIC STORAGE [YES | NO]>
<ON [StoragePath, ...] <DBPATH [DBPath]>>
<ALIAS [Alias]>
<USING CODESET [CodeSet] TERRITORY [Territory]>
<COLLATE USING [CollateType]>
<PAGESIZE [4096 | Pagesize <k>]>
<DFT_EXTENT_SZ [DefaultExtentSize]>
<RESTRICTIVE>
<CATALOG TABLESPACE [TS_Definition]>
<USER TABLESPACE [TS_Definition]>
<TEMPORARY TABLESPACE [TS_Definiton]>
<WITH "[Description]">
```

  - Example:
    - Requirement:

        - Uses automatic storage
        - Uses the paths "/mnt/fsystem1" and "/mnt/fsystem2" to store its data and metadata
        - Recoginzes the United States and Canada code set
        - Uses a collating sequence that is based on the United States and Canada code set
        - Has a page size of 8 KB
    * SQL:

```
CREATE DATABASE sample
ON /mnt/fsystem1, /mnt/fsystem2
USING CODESET 1252 TERRITORY US
COLLATE USING SYSTEM
PAGESIZE 8192
```

- DB2 for z/OS

3.5 Establishing a Database Connection

- connect example:

```
CONNECT TO sample USER db2user USING ibmdb2
```

- Type 1 and Type 2 Connections

Table 4.2: Differences between Type 1 and Type 2 connections	
Type 1 Connections	**Type 2 Connections**
The current transaction (unit of work) must be committed or rolled back before a connection to another application server can be established.	The current transaction does not have to be committed or rolled back before a connection to another application server can be established.
Establishing a connection to another application server causes the current connection to be terminated. The new connection becomes the current connection.	Establishing a connection to another application server places the current connection into the dormant state. The new connection then becomes the current connection.
The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until another CONNECT statement is executed.	The CONNECT statement establishes the current connection the first time it is executed against a server. If the CONNECT statement is executed against a connection that is in the dormant state, that connection becomes the current connection—provided the SQLRULES precompiler option was set to DB2 when the application was precompiled. Otherwise, an error is returned.
The SET CONNECTION statement is supported, but the only valid target is the current connection.	The SET CONNECTION statement changes the state of a connection from dormant to current.
Connecting with the USER…USING clauses will cause the current connection to be disconnected and a new connection to be established with the given authorization ID and password.	Connecting with the USER…USING clauses is allowed only when there is no dormant or current connection to the same named server.

Figure 4: Difference between Type1 and Type2 connection

  - Type 1 connections are used by default with DB2 LUW; Type 2 connections are used by default with DB2 z/OS.
  - Type 2 connections allow a single transaction to connect to and work with multiple databases simultaneously.
  - Type 1 connections allow a transaction to be connected to only one database at a time.

- DB2 Connect
  - IBM DB2 Connect Application Server Edition
  - IBM DB2 Connect Personal Edition
    * makes DB2 data stored on System z, System i, and IBM Power Systems servers directly available to desktop applications

    - enables applications to work transparently with data stored on multiple systems *without using a gateway*
  - IBM DB2 Connect Enterprise Edition
    - connects LAN-based systems and desktop applications to System z, System i, and IBM Power Systems databases
    - host access can be consolidated through a gateway, making it easier to deploy web and multitier client/server applications
  - IBM DB2 Connect Unlimited Advanced Edition for System z
    - makes it easy to access, manage, and optimize the performance of enterprise information, wherever it resides.
    - Because it is licensed for unlimited deployment on authorized servers, this edition is cost-effective solution for organizations that use DB2 Connect extensively, especially where multiple applications are involved.
  - IBM DB2 Connect Unlimited Edition for System i
    - integrates IBM System i data with client/server, web, mobile, and service-oriented architecture (SOA) applications.
    - it delivers unified application development, integrated data, and pervasive data functionality to System i users.

3.6 Temporal Data Management and Time Travel Tables

3.6.1 Basic Temporal Data Concepts

- time period: [date/time/timestamp, date/time/timestamp)
- system period: refers to the period in which a particular row is considered **current**. (*system time* is often associated with traching when changes are made to the state of a record)
- application period: refers to the period in which a particular row is considered valid. (*business time* (aka *valid time*, *application time*) is usually associated with tracking the effective dates of certain business conditions.

3.6.2 System-period temporal tables

- table that maintains historical versions of its rows.
- must be associated with a history table
- any time a row in a system-period temporal table is modified, DB2 automatically inserts a copy of the original row into the corresponding history table.
- Requirement:
  - system/row time begin (`sys_start`) (TIMESTAMP(12) data type)
  - system/row time end (`sys_end`) (TIMESTAMP(12) data type)
  - transaction start-ID (`ts_id`) (TIMESTAMP(12) data type): allow DB2 to capture the start times of transactions that perform update or delete operations on a particular row

- PERIOD SYSTEM_TIME clause

- Example:
  - Create a system-period temporal table named TAX_INFO

```
CREATE TABLE tax_info
 (taxpayer_id  CHAR(4) NOT NULL,
  tax_amount   INT NOT NULL,
  sys_start    TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS ROW BEGIN,
  sys_end      TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS ROW END,
  ts_id        TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS,
                                 TRANSACTION START ID
                                 IMPLICITLY HIDDEN
  PERIOD SYSTEM_TIME (sys_start, sys_end))
```

  - Create a history table named HIST_TAX_INFO with the same definition as the TAX_INFO

```
    CREATE TABLE hist_tax_info LIKE tax_info
```

  - Establish a link between the system-period temporal table named TAX_INFO and the history table named HIST_TAX_INFO

```
            ALTER TABLE tax_info
             ADD VERSIONING
             USE HISTORY TABLE hist_tax_info
```

3.6.3 Application-period temporal tables

- a table that maintains "currently in effect" values of application data.

- Requirement:
  - business time begin (bus_start)
  - business time end (bus_end)
  - PERIOD BUSINESS_TIME clause

- Example:
  - Create an application-period temporal table named INVENTORY

```
                          CREATE TABLE inventory
                           (item_id         CHAR(4) NOT NULL,
                            price           DOUBLE NOT NULL,
                            bus_start       DATE NOT NULL,
                            bus_end         DATE NOT NULL,
                           PERIOD BUSINESS_TIME (bus_start, bus_end),
                           PRIMARY KEY(item_id, BUSINESS_TIME WITHOUT OVERLAPS))
```

3.6.4 Bitemporal tables

- combines the historical tracking of a system-period temporal table with the time-specific data storage capabilities of an application-period temporal table.
- a single table needs to maintain historical versions of its rows and keep track of data values that are currently considered valid.
- When querying a bitemporal table, you have the option of providing a system time-period specification, a business time-period specification, or both, or neither.
- Example:
  - Create a bitemporal table named INVENTORY

```
CREATE TABLE inventory
(item_id         CHAR(4) NOT NULL,
 price           DOUBLE NOT NULL,
 sys_start       TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS ROW BEGIN,
 sys_end         TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS ROW BEGIN,
 ts_id           TIMESTAMP(12) NOT NULL
                                 GENERATED ALWAYS AS
                                 TRANSACTION START ID,
 bus_start       DATE NOT NULL,
 bus_end         DATE NOT NULL,
 PERIOD SYSTEM_TIME (sys_start, sys_end)
 PERIOD BUSINESS_TIME (bus_start, bus_end))
```

  - Create a corresponding history table

```
CREATE TABLE hist_inventory LIKE inventory
```

  - Establish a link between `HIST_INVENTORY` and `INVENTORY` table

```
ALTER TABLE inventory
ADD VERSIONING
USE HISTORY TABLE hist_inventory
```

4 Working with DB2 Data Using SQL

4.1 Objectives

- Ability to use SQL to SELECT data from tables
- Ability to use SQL to SORT or GROUP data
- Ability to use SQL to UPDATE, DELETE, or INSERT data
- Knowledge of transactions (ie. commit/rollback and transaction boundaries)
- Ability to create and call an SQL supported procedure or a user defined function (understanding of passing parameters and results)
- Given an XQuery statement, knowledge to identify results
- Knowledge of Temporal(Time Travel) Tables - System-period, Application-period, and Bitemporal- ability to query

4.2 SELECT data from tables

4.2.1 FETCH FIRST - Getting a limited amount of data

- To retrieve a specific number of rows from a table, use the SELECT statement with the FETCH FIRST clause
- Example: retrieve the first two rows from the *sales* table:

```
1 SELECT * FROM sales FETCH FIRST 2 ROWS ONLY
```

4.2.2 Isolation clause - Accessing restricted data

- To specify the isolation level a query is to be run under, and in some cases, to suggest the type of lock DB2 should acquire and hold on the data being queried.
- Example: retrieve uncommitted data from *sales* table:

```
1 SELECT * FROM sales WITH UR
```

- Example: retrieve data from a table with minimal locking, use the FOR FETCH ONLY or FOR READ ONLY

```
1 SELECT * FROM sales FOR FETCH ONLY
```

4.2.3 WHERE clause - Restricting the result set

WHERE clause

- To restrict the data in the result set
- Example: get all the employees from the *employee* table who are hired after year 2005 and whose workdept is in 'AOO' and 'E21'

```
SELECT * FROM employee WHERE YEAR(hiredate) > '2005' AND workdept IN ('AOO', 'E21')
```

Predicate

- **LIKE** predicate: search for string patterns in column values
- Example: find all employees whose first name starts with 'E' in the *employee* table:

```
SELECT * FROM employee WHERE firstnme LIKE 'E%'
```

- **BETWEEN** predicate: specifying data ranges
- Example: select employees whose hire date is between 1998 and 2000:

```
SELECT firstnme FROM employee WHERE YEAR(hiredate) BETWEEN '1998' AND '2000'
```

- **NULL** predicate: search for colmuns that have null values
- Example: select all employees without a middle initial:

```
SELECT firstnme FROM employee WHERE midinit IS NULL
```

- **EXISTS** predicate: determine whether a particular value exists in a given result set
- **IN** predicate

4.2.4 DISTINCT clause - Eliminating duplicates

- To eliminate duplicates from the final result set, use the SELECT DISTINCT clause
- Example: select all the names of salespersons from the *sales* table without duplicates:

```
SELECT DISTINCT sales_person FROM sales
```

4.2.5 GROUP BY clause - grouping data in result sets

GROUP BY

- Example: determine the average sales of a sales person in sales table:

```
SELECT sales_person, AVG(sales) avg_sales FROM sales GROUP BY sales_person
```

```
SALES_PERSON   AVG_SALES
------------  -----------
GOUNOT                 3
LEE                    5
LUCCHESSI              1
```

- Specify all columns that are not aggregated in GROUP BY clause

HAVING clause

- **HAVING** clause: If you use the aggregate functins such as MIN() or MAX() to specify a condition, you must use the HAVING clause.
- Example: determine the minimum and maximum salary paid for a job and the maximum salary is greater than 27000

```
SELECT job, MIN(salary), MAX(salary)
FROM employee
GROUP BY job
HAVING MAX(salary) >= 27000
```

GROUP BY ROLLUP clause

- Example:

```
db2 => select job, sex, dec(avg(salary),9,2) from employee group by rollup (job, sex)

JOB      SEX 3
-------- --- -----------
-        -      58155.35
ANALYST  -      70213.33
CLERK    -      43876.25
DESIGNER -      57437.00
FIELDREP -      39274.00
MANAGER  -      88142.14
OPERATOR -      37898.33
PRES     -     152750.00
SALESREP -      56500.00
ANALYST  F      70213.33
```

```
CLERK    F      42315.00
CLERK    M      44396.66
DESIGNER F      58317.50
DESIGNER M      56850.00
FIELDREP F      35370.00
FIELDREP M      40250.00
MANAGER  F      94723.33
MANAGER  M      83206.25
OPERATOR F      38575.00
OPERATOR M      36545.00
PRES     F     152750.00
SALESREP F      46500.00
SALESREP M      66500.00

23 record(s) selected.
```

GROUP BY CUBE clause

- Example:

```
db2 => select job, sex, dec(avg(salary),9,2) from employee group by rollup (job, sex)

JOB      SEX 3
-------- --- -----------
-        F      63243.68
-        M      53951.95
-        -      58155.35
ANALYST  -      70213.33
CLERK    -      43876.25
DESIGNER -      57437.00
FIELDREP -      39274.00
MANAGER  -      88142.14
OPERATOR -      37898.33
PRES     -     152750.00
SALESREP -      56500.00
ANALYST  F      70213.33
CLERK    F      42315.00
CLERK    M      44396.66
DESIGNER F      58317.50
DESIGNER M      56850.00
FIELDREP F      35370.00
FIELDREP M      40250.00
MANAGER  F      94723.33
MANAGER  M      83206.25
OPERATOR F      38575.00
OPERATOR M      36545.00
PRES     F     152750.00
SALESREP F      46500.00
SALESREP M      66500.00

25 record(s) selected.
```

4.2.6 ORDER BY - Sorting data

- sort the data in the result set. If multiple sort keys are specified, they are applied in the order of specification.
- can indicate ascending order or descending order. The default is ascending order.
- Example: order the result set by first name and then by last name in ascending order:

```
SELECT firstnme, midinit, lastname FROM employee ORDER BY firstnme, lastname
```

- can specify the position of a column in the result set instead of the column name in the ORDER BY clause.
- Example: order the result set using the firstnme column in ascending order and the lastname column in descending order:

```
SELECT firstnme, midinit, lastname FROM employee ORDER BY 1 asc, 3 desc
```

4.2.7 Retrieving data from multiple tables

Cartesian product

- A cartesian product merges all the values from two tables in one result set.
- A cartesian product happens when you specify multiple tables in the FROM clause without a WHERE clause.
- Example:

```
SELECT * FROM employee, department
```

- Explanation: the above query returns a total row count of 630 because the EMPLOYEE table has 42 rows and the DEPARTMENT table has 15 rows.

Inner joins

- Inner joins can be thought of as the cross product of two tables, in which every row in one table is paired with rows in another table that have matching values in one or more columns.
- Example:

```
SELECT deptno, deptname, firstnme FROM department, employee WHERE deptno=workdept
```

- Explanation: returns the department number, department name, and first name for each employee where the department number in the employee table is found in the department table. The employees that have a department number in the employee table without a match in the department table are not listed in the result set.
- Example:

```
SELECT deptno, deptname, firstnme FROM department INNER JOIN employee ON deptno=workdept
```

- This returns the same result set as previous.

Outer joins

- An outer joins returns rows that match the join condition and the row from both tables that do not match the join condition.

Left outer join

- returns the matching rows from both tables and the rows in the left table that do not match the join condition
- Example:

```
SELECT deptno, deptname, firstnme FROM department LEFT OUTER JOIN employee ON deptno=workdept
```

Right outer join

- returns the matching rows from both tables and the rows in the right table that do not match the join condition
- Example:

```
SELECT deptno, deptname, firstnme FROM department RIGHT OUTER JOIN employee ON deptno=workdept
```

Full outer join

- returns the matching rows from both tables and the rows in both table that do not match the join condition
- Example:

```
SELECT deptno, deptname, firstnme FROM department FULL OUTER JOIN employee ON deptno=workdept
```

4.2.8 Using set operators in queries

UNION

- use UNION to combine two sets of values and eliminate duplicates
- Example:

```
SELECT sales_person, MAX(sales) FROM sales GROUP BY sales_person
UNION
SELECT sales_person, MIN(sales) FROM sales GROUP BY sales_person
```

- to retrieve all the rows in the result set including duplicates, use `UNION ALL`.

INTERSECT

- use INTERSECT to combine answers from two different sets. It returns the common values between the two sets.
- Example: list all the employees whose resumes are in emp_resume table:

```
SELECT empno FROM emp_resume INTERSECT SELECT empno FROM employee
```

- `INTERSECT ALL` retrieve the common values including duplicates.

EXCEPT

- use EXCEPT to retrieve the rows that are not present in another result set.
- Example: determine how many employees do not have a project assigned:

```
SELECT empno FROM employee EXCEPT SELECT empno FROM empproject
```

- `EXCEPT ALL` retains duplicate

4.3 Manipulating data with SQL

You can add, update, or remove data into tables using SQL statements such as `INSERT`, `UPDATE`, `DELETE`, and `MERGE`. These statements are part of the DML.

4.3.1 UPDATE - Updating column values in a table or view

- can use the WHERE clause to update a selection of columns or rows that match the condition.
- Example: updates the commission to 10 for all the employees with ETHEL as first name:

```
UPDATE employee SET commission = 10 WHERE firstnme = 'ETHEL'
```

- if you do not specify a condition with the WHERE clause, all the columns in the table are updated:

```
UPDATE employee SET (salary, phoneno) = (900000.50, '8888')
```

- you can update a table using calculated values from a subquery:

```
UPDATE employee EMP set (EMP.salary, EMP.comm) =
 (SELECT avg(salary), avg(comm) FROM employee WHERE firstnme = 'ETHEL')
```

4.3.2 Deleting data from a table or view

Use the DELETE statement to eliminate data from a table or view.

- To remove specific rows that match a condition, use the WHERE clause.
- Example: delete employees whose name starts with 'J':

```
DELETE FROM employee WHERE firstnme LIKE 'J%'
```

- can use a subquery in the condition to specify which rows to delete

```
DELETE FROM employee WHERE lastname IN (SELECT sales_person FROM sales WHERE YEAR(sales_date) =
```

- can delete all rows with one DELETE statement
- Example: remove all the rows from the employee table

```
DELETE FROM employee
```

- For a faster delete operation (ie. not generating transaction logs), use TRUNCATE TABLE statement to delete all the data
- Example:

```
TRUNCATE TABLE employee IMMEDIATE
```

4.3.3 INSERT - Adding data to tables, nicknames, or views

- To insert a row into a table that has columns with the NOT NULL constraint, you must specify values for all these columns
- Example: to insert data to the act table, all the columns with the NOT NULL constraints are specified in the INSERT statement

```
INSERT INTO act VALUES (190, 'DBA', 'CREATE DATA')
```

- can specify column names in the INSERT
- Example:

```
INSERT INTO department (deptno, deptname, admrdept) VALUES
        ('B11', 'PURCHASING', 'B01'),
        ('E41', 'DATABASE ADMINISTRATOR', 'E01')
```

4.4 Working with transactions

For DB2 databases, a transaction is a unit of work (UOW) consisting of a series of sequential SQL statements such as CREATE, INSERT, UPDATE or DELETE that ends when a COMMIT or ROLLBACK happens.

4.4.1 Committing or rolling back UOWs

You can explicitly set transactions by issuing a COMMIT or ROLLBACK statement after issuing a series of DML statements. When you issue a COMMIT statement, the database manager makes all the changes, within that UOW, to the database permanent. However, if there is an error in a statement within the UOW or you issue a ROLLBACK statement, the database manager reverts all the changes made to database and the database gets to the same state it was before the UOW started.

4.4.2 Using savepoints UOWs

- Use the SAVEPOINT statement to define levels within a UOW. By setting a savepoint within a UOW, you can roll back to that savepoint.

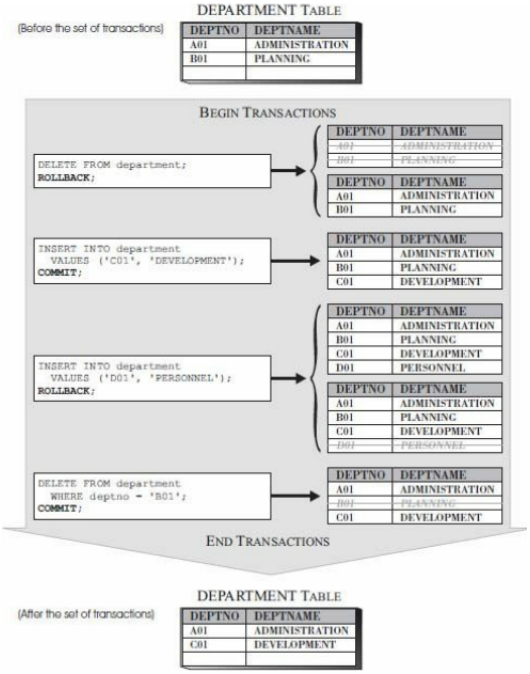

Figure 5.16: Evaluating the effects of a series of transactions

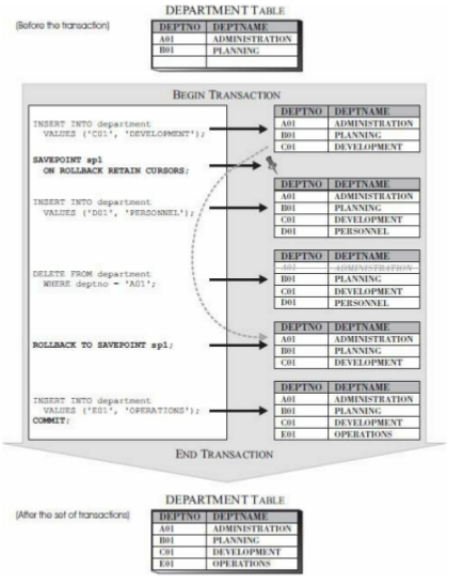

Figure 5.17: Using savepoints with the ROLLBACK *statement to provide granular control over a long-running transaction*

4.5 Using a Cursor to Obtain Results from a Result Data Set

- When a SELECT statement is executed from within an application program, DB2 uses a mechanism known as a *cursor* to retrieve data values from the result data set produced.
- Category:
  - Read-only
  - Updatable: rows in the result data set that cursor are associated with can be modified or deleted
- use a cursor:
  1. Declare the cursor along with its type and associate it with a query (SELECT or VALUES statement)
  2. Open the cursor. This will cause the corresponding query to be executed an a result data set to be produced.
  3. Fetch each row in the result data set, one by one, until an "End of data" condition occurs- each time a row is fetched, the cursor is automatically moved to the next row.
  4. If appropriate, alter or delete the current row by executing an UPDATE ... WHERE CURRENT OF or a DELETE ... WHERE CURRENT OF statement (only cursor is updatable).
  5. Close the cursor. This will delete the result data set that was produced when the cursor was first opened.
- corresponding SQL statements:
  - `DECLATE` cursor-name `CURSOR FOR` SELECT-Statement
  - `OPEN` cursor-name
  - `FETCH FROM` cursor-name `INTO` VAR1, VAR2, ...
  - `CLOSE` cursor-name
- Often used in stored procedure

4.6 Working with SQL procedures

You can create SQL procedures in the db2 database server as a way to convert business or system logic to a process that uses SQL and runs on the db2 database server.

4.6.1 Creating SQL prodcedures

- Use the `CREATE PROCEDURE` statement to create a new SQL procedure or recreate an existing SQL procedure by using the `REPLACE` keyword.
- You can indicate input (IN), output (OUT), or both input and output (INOUT) parameters enclosed in parenthesis and separately by comma.

4.6.2 SQL Procedural Language

See my page example

4.6.3 Calling SQL procedures

See my page example

4.7 Working with user-defined functions

- Use user-defined functions (UDFs) when you want to extend the capabilities of existing DB2 built-in functions.
- UDFs provide a way to generate consistent output for given input. It also helps to have some of the logic perform in the data server and thus improve application performance.
- type of UDFs:
  - Scalar functions
  - Aggregate functions
  - Row functions: return one row
  - Table functions: return a table (result set)

4.7.1 Creating user-defined functions

See my page example

4.7.2 Calling user-defined functions

See my page example

4.8 Retrieving data using XQuery

You can use DB2 databases to store well-formed XML data in a column of a table and use SQL, XQuery, or a combination of SQL and XQuery to retrieve the XML data from the database.

4.9 Working with temporal tables

4.9.1 Concept revisit

- basic concepts:
  - System time: tracks when changes are made to the tables
  - Business time: tracks for which time the row is valid
- table types:
  - System-period temporal tables:

    - System-period temporal tables use a historical version of data. Use a system-period temporal table to store the current version of data. The associated history table automatically stores the updated and deleted data.
    - must include: *a*) a SYSTEM_TIME period column to capture the begin and end times when the data in a row is current *b*) a transaction start-ID column to capture the time when execution started for a transaction that impacts the row
  - Application-period temporal tables:
    - Use an application-period temporal table to manage data based on time criteria that defines the time period when data is valid.
    - Must include: *a*) a BUSINESS_TIME period column to capture the begin and end times when the data in a row is valid
  - Bitemporal tables:
    - combine the capability of tracking history of system-period temporal tables and the time-specific storage of application-period tables

4.9.2 Examples

See my page example