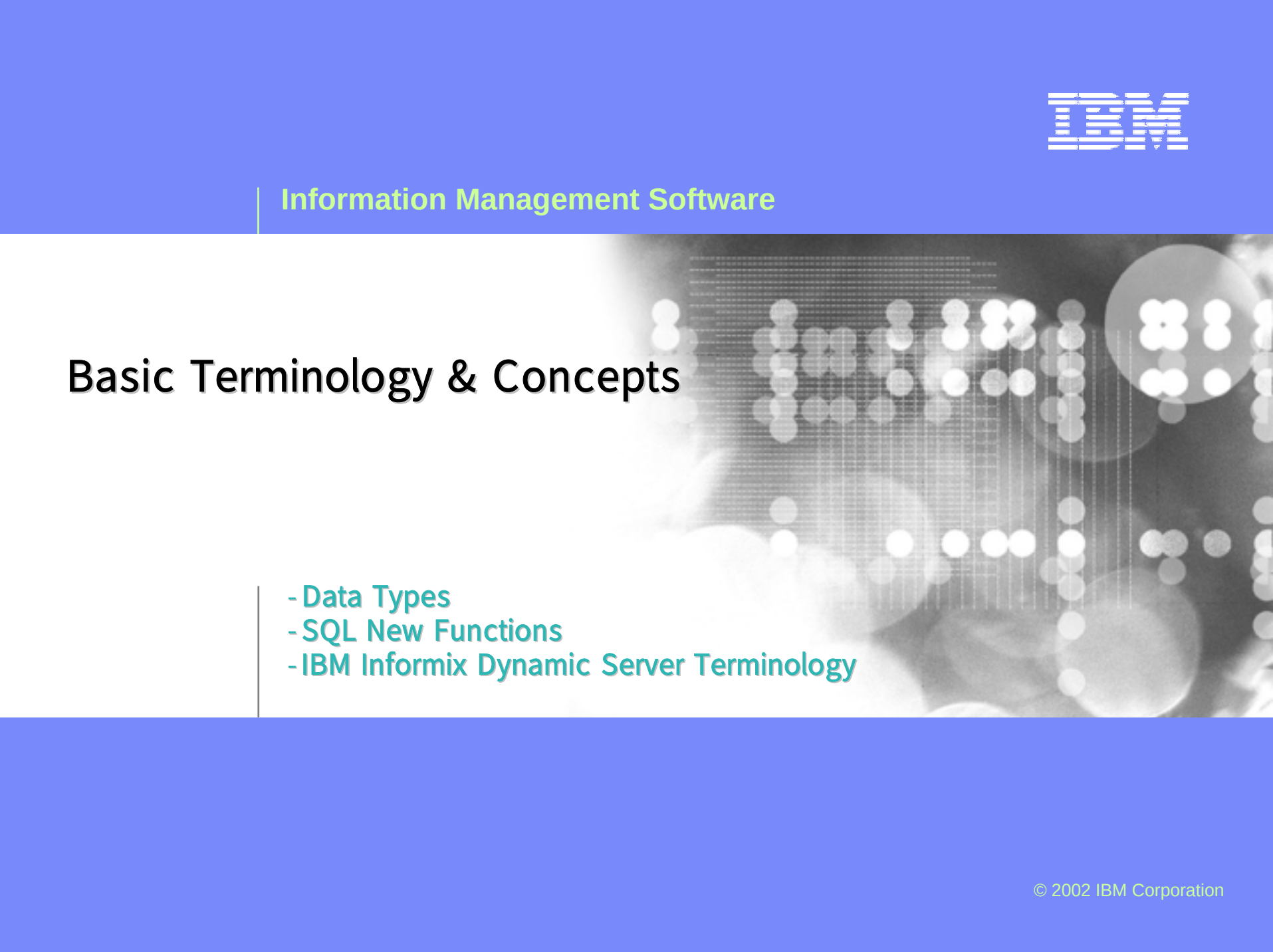
Information Management Software

IBM Informix Programmer's Guide

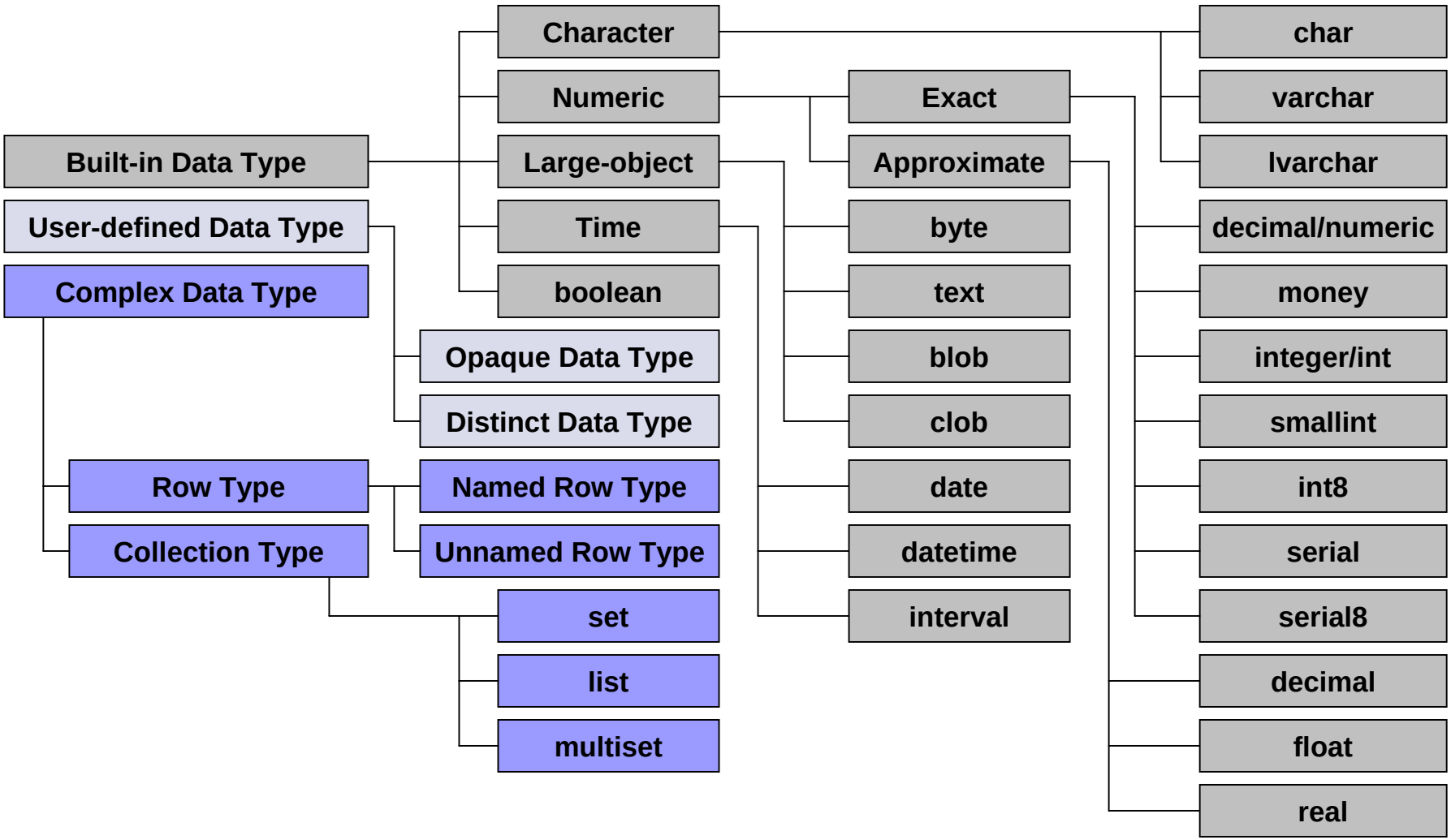
2003. 03.

Agenda

- Basic Terminology & Concepts
- ESQL programming highlights
- Concurrency Control
- Query Optimization
- Homepage Supports



IBM Informix Data Types



Built-in Data Types

- CHARACTER
  - CHAR (n)
  - VARCHAR (m,r)
  - LVARCHAR
- BOOLEAN
- NUMERIC
  - SMALLINT
  - INTEGER / INT8
  - FLOAT
  - SMALLFLOAT
  - DECIMAL (p,s)
  - MONEY (p,s)
  - SERIAL / SERIAL8

Type	최소	최대	크기
smallint	$-(2^{15}-1)$: -32,767	$2^{15}-1$: 32,767	2 byte
integer	$-(2^{31}-1)$: -2,147,483,647	$2^{31}-1$: 2,147,483,647	4 byte
int8	$-(2^{63}-1)$	$2^{63}-1$	8 byte (64bit) 10 byte (32bit)
smallfloat	최대 8자리 유효자리수		4 byte
float	최대 16자리 유효자리수		8 byte
decimal money	precision, scale을 지정 최대 32자리 유효자리수 money는 화폐기호 ($DBMONEY) 출력 기본값 : decimal(16,0), money(16,2)		scale이 홀수: (p+4)/2 byte 짝수: (p+3)/2 byte

Built-in Data Types

- TIME
  - DATE
  - DATETIME
  - INTERVAL

입력값		99.06.04	02.06.04	02.12.31
DBCENTURY 설정에 따라 저장된 값 ※ 오늘날짜 : 2002.06.04	P	1999.06.04	1902.06.04	1902.12.31
	F	2099.06.04	2102.06.04	2002.12.31
	C	1999.06.04	2002.06.04	2002.12.31
	R	2099.06.04	2002.06.04	2002.12.31

P : past
F : Future
C : Closest
R : Present (Default)

Built-in Data Types

- LARGE-OBJECTS
  - Simple Large-Objects
    - TEXT
    - BYTE
  - Smart Large-Objects
    - BLOB
    - CLOB

Complex Data Types / User Defined Data Types

- Complex Data Types
  - Collection Data Types
    - SET
    - MULTISET
    - LIST
  - Row Data Types
    - Named row types
    - Unnamed row types
- User-Defined Data Types
  - Distinct Types
  - Opaque Types

SQL New Functions

- CASE WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  ELSE resultn
  END

- NVL(value1, value2)
  같은 의미 : CASE value1
  WHEN null THEN value2
  ELSE value1
  END

- DECODE(value1, value2, value3, value4, ... valuen)
  같은 의미 : CASE value1
  WHEN value2 THEN value3
  ...
  ELSE valuen
  END

SQL New Functions

- TO_DATE(string, [,fmt])
- TO_CHAR(date, [,fmt])

format 문자	의미
%Y	4자리 연도
%m	월
%B	월 (January, February, March, April, ...)
%d	일
%A	요일 (Monday, Tuesday, Wednesday, ...)
%R	24시간:분
%H	24시간
%M	분
%S	초
%Fn	소수점 이하 n자리 초

SQL New Functions

- TRIM({ LEADING | TRAILING | BOTH } trim_expr FROM source_expr)
- TRIM(source_expr) = TRIM(TRAILING ‘ ’ FROM source_expr)

- LPAD (string, length, [,pad])
- RPAD (string, length, [,pad])

- UPPER
- LOWER
- INITCAP

SQL New Functions

- REPLACE (string, old [,new])
- SUBSTR(string,start [,length])
  start : Positive - Counts forward
  Negative - Counts backward
  0 - equivalent to 1
- SUBSTRING(string FROM start [FOR length])
  Start : Positive - Counts forward
  Negative - Counts backward from one position before the first character
  0 - counts from one position before the first character

IBM Informix Dynamic Server Terminology

- Chunk
- Page
- Extent
- Tblspace
- Dbspace
- Blobspace
- Smart blobspace

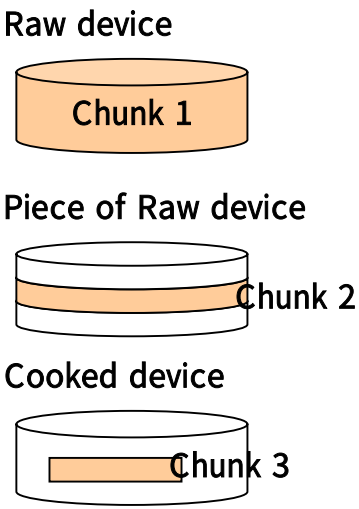
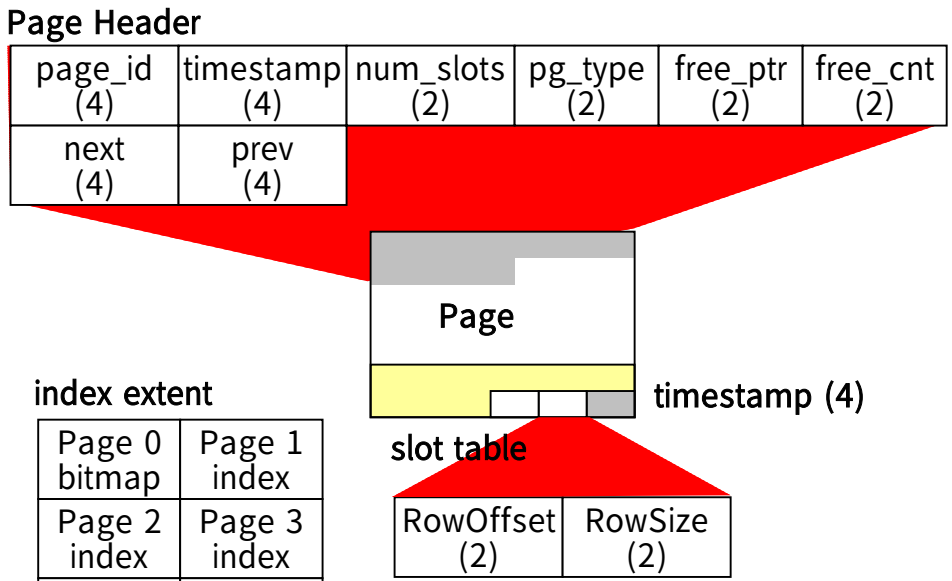


Table extent

Page 0 bitmap	Page 1 data
Page 2 data	Page 3 data
Page 4 reminder	Page 5 blob
Page 6 free	Page 7 free

index extent

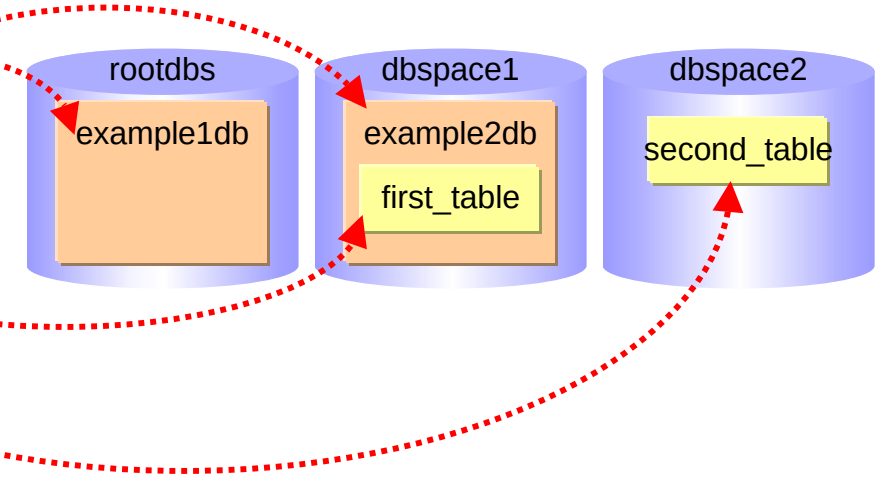
Page 0 bitmap	Page 1 index
Page 2 index	Page 3 index
Page 4 index	Page 5 free
Page 6 free	Page 7 free



Logical Concepts / Physical Concepts

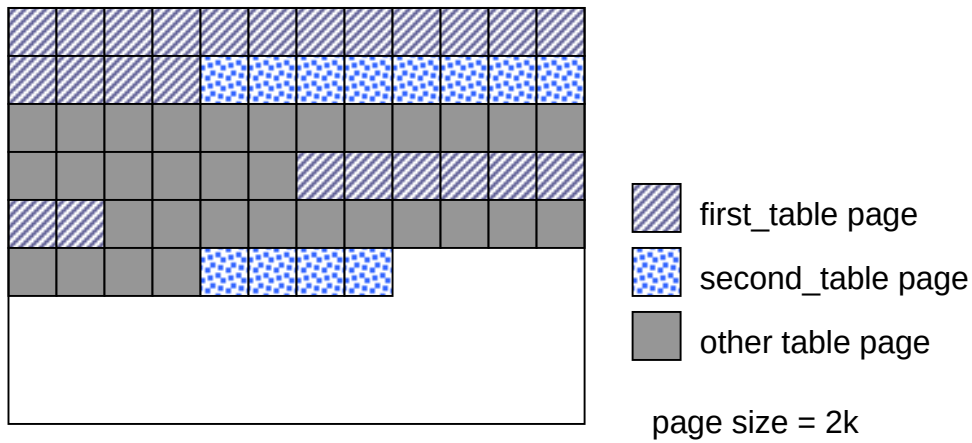
- Logical Concepts

```
create database example1db;
create database example2db in dbspace1;
create table first_table (
    tab_id          int ,
    name            char(20) ,
    address         char(50)
) ;
create table second_table (
    tab_id          int ,
    name            char(20) ,
    address         char(50)
) in dbspace2 ;
```



- Physical Concepts

```
create table first_table (
    tab_id          int ,
    name            char(20) ,
    address         char(50)
) extent size 32 next size 16 ;
create table second_table (
    tab_id          int ,
    name            char(20) ,
    address         char(50)
) extent size 16 next size 8 ;
```

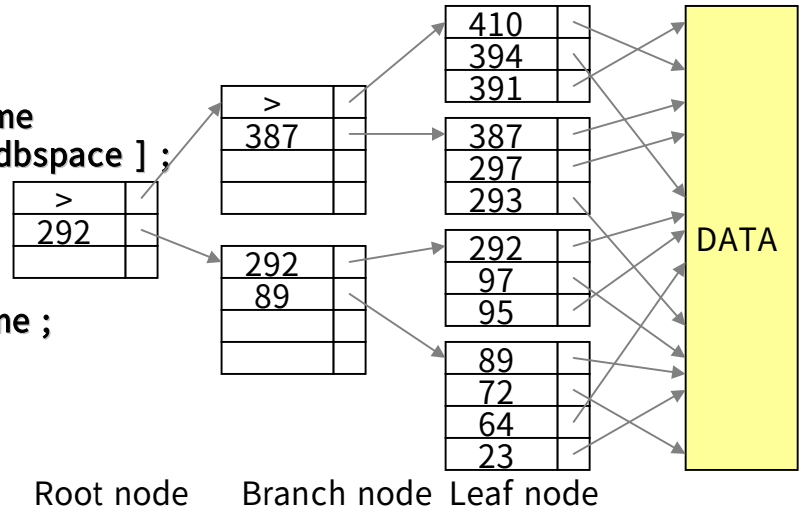


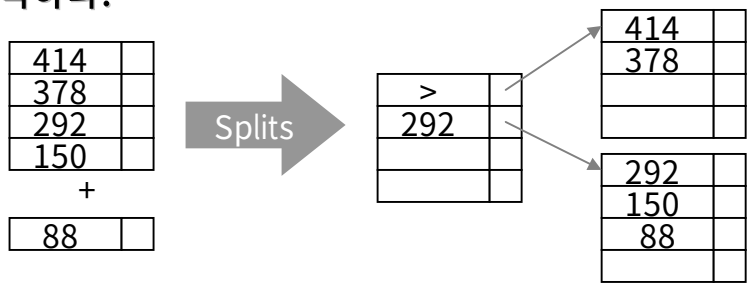
Data Page monitoring

- 테이블 생성
  CREATE TABLE monitoring (col1 SERIAL, col2 CHAR(2000));
  CREATE TABLE dummy (col1 INT);
- 데이터 입력 (8건)
  INSERT INTO monitoring VALUES (0,'data');
- 컬럼 추가
  ALTER TABLE monitoring ADD col3 INT;
- 데이터 입력 (1건)
  INSERT INTO monitoring VALUES (0,'data');
- Monitor
  oncheck –pT stores_demo:monitoring

Index Page

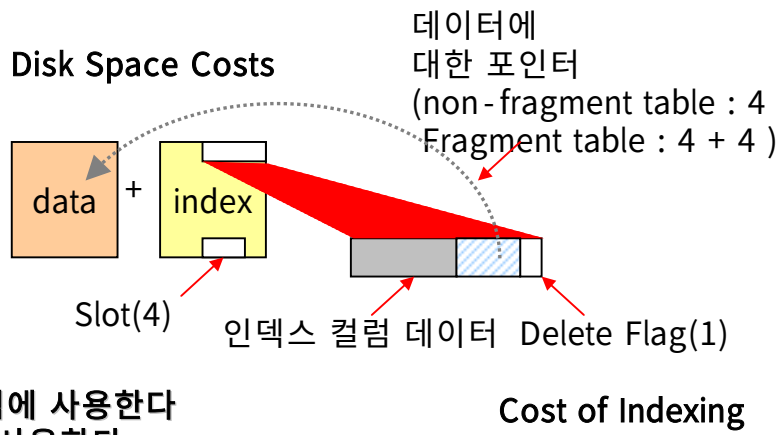
- 인덱스 종류
  - Unique
  - Duplicate
  - Composite
  - Cluster
- 인덱스 생성 기본 구문
  - CREATE [UNIQUE] [CLUSTER] INDEX index_name
    ON table_name (column_name_list) [IN dbspace] ;
- 인덱스 변경
  - ALTER INDEX index_name TO CLUSTER ;
  - RENAME INDEX old_index_name TO new_index_name ;
- 인덱스 삭제
  - DROP INDEX index_name ;
- 다음과 같은 이유로 인덱스 키는 작게 만드는 것이 바람직하다.
  - 하나의 페이지에 저장되는 인덱스 키가 많아진다
  - 인덱스 페이지의 노드 레벨이 감소한다

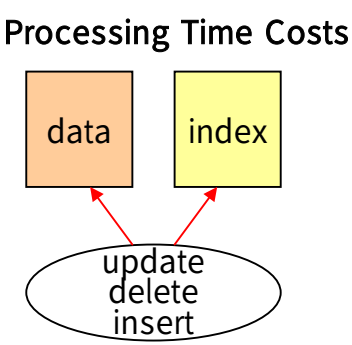




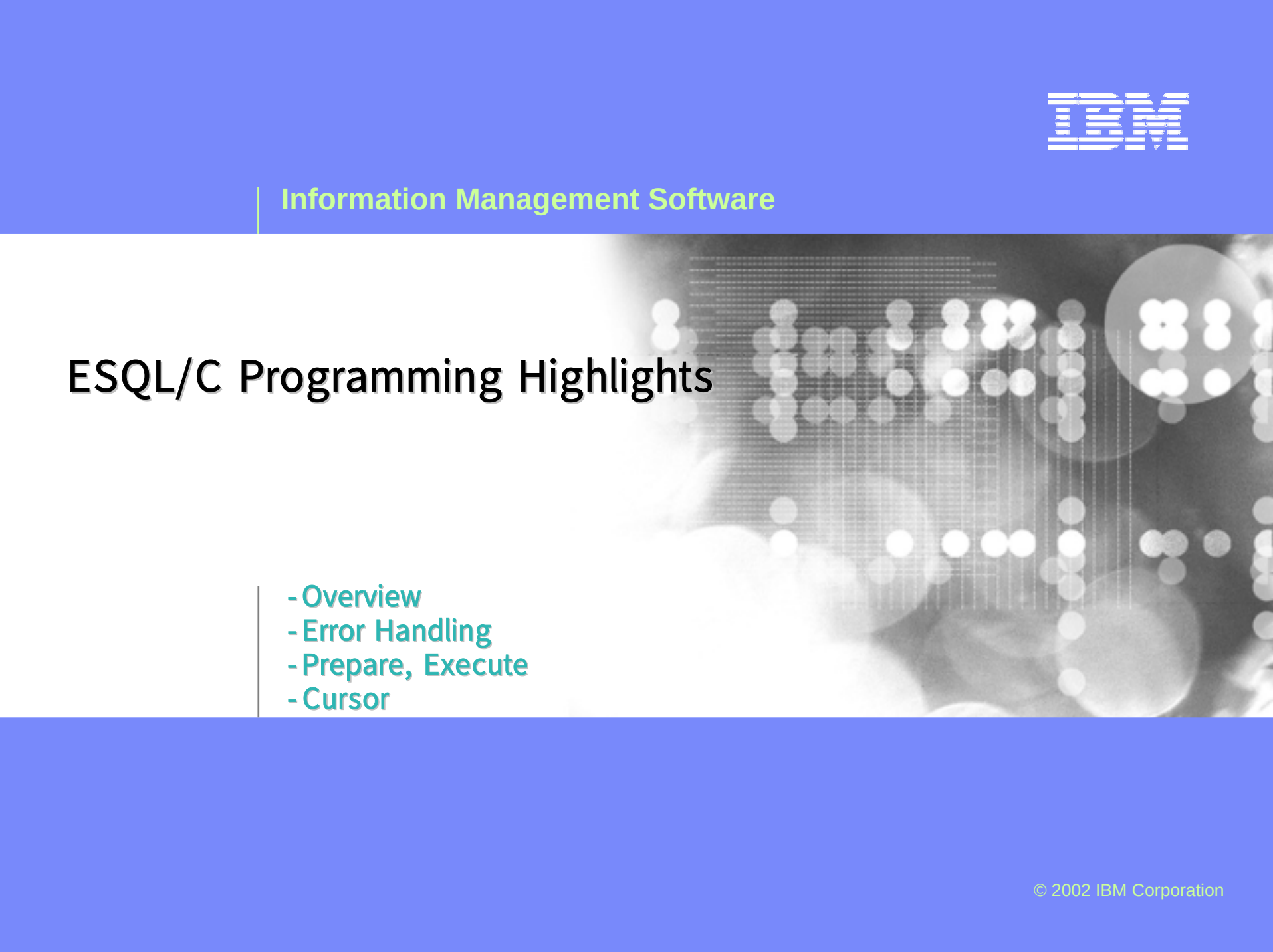
Index Page

- 인덱스 사용의 장점
  - 데이터 페이지의 입출력 횟수가 감소한다
  - 인덱스 데이터에 대하여 정렬 작업이 생략된다
  - 데이터의 유일성을 보장할 수 있다 (Unique index)
  - Key-only 검색으로 데이터 페이지를 읽지 않을 수 있다
- 인덱스 소요 비용
  - 디스크 사용 공간
  - DML 문장 수행시간





- 인덱스 사용 가이드
  - 선택적인 필터 컬럼, 조인 컬럼에 사용한다
  - 정렬에 자주 사용되는 컬럼에 사용한다
  - 중복값이 많은 컬럼에는 사용하지 않는다
  - 주요 인덱스의 개수를 제한한다
  - 키값의 크기를 작게 한다
  - Composite 인덱스로 유일성을 증가시킨다
  - Cluster 인덱스로 데이터 조회 속도를 증가시킨다
  - 대량의 update, delete, insert 작업 전에 미리 인덱스를 disable한다



Information Management Software

ESQL/C Programming Highlights

- Overview
- Error Handling
- Prepare, Execute
- Cursor

Overview

- ESQL/C
  - C 프로그램 내에 sql 구문을 바로 넣어 그 실행 결과를 프로그램에서 다양하게 사용할 수 있도록 제공되는 어플리케이션 개발 tool
  - ESQL/C의 주된 component는 preprocessor로서 ESQL/C code를 C code로 변환하여 C compiler에게 넘겨준다.

- ESQL/C프로그램 작성 규칙
  - ESQL/C preprocessor가 다른 C code와 구별할 수 있게 하기 위해 SQL은 “$” 혹은 “EXEC SQL”으로 시작
  - SQL구문의 끝에는 “;”를 붙임.
  - SQL 구문 안의 변수(host 변수라 부름)는 변수 이름 앞에 “:”를 붙임.
  - 주석은 표준 C의 주석인 “/* */”사용

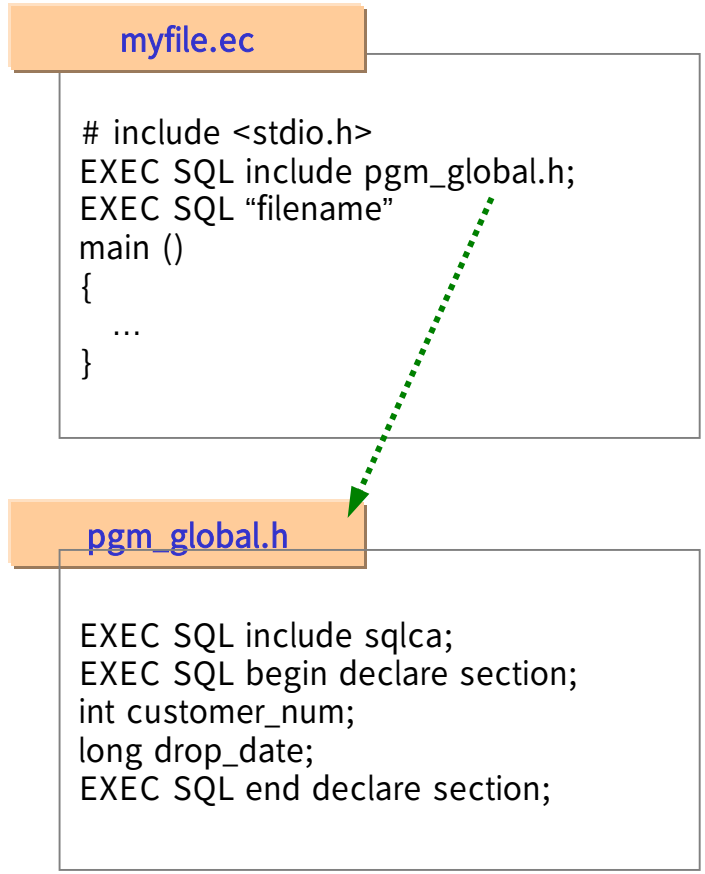
- ESQL/C 프로그램 컴파일
  - esql [-e] [preprocessor 옵션] [cc 매개변수] [-o 실행파일] 소스.ec [링크옵션]

Overview

- include 에 지정된 파일을 찾는 순서
  - 현재 디렉토리
  - 컴파일시 -I 옵션으로 지정한 디렉토리
  - $INFORMIXDIR/incl/esql
  - /usr/include

- 다음과 같은 preprocessor 구문을 사용할 수 있다
  - define, undef
  - ifdef, elseif, else, endif, ifndef
  - 예제

```
EXEC SQL define USERTRANSACTIONS;
EXEC SQL ifdef USERTRANSACTIONS;
EXEC SQL begin work;
EXEC SQL endif;
```



Host variables

- 호스트 변수 : SQL 데이터를 저장하는 C 변수
- 대소문자 구분
- SQL문장안에서 컬럼명과 구분하기 위하여 호스트변수 앞에 :기호 사용
- 선언
  - 달러 기호 사용
    ```
    $ CHAR *desc;
    ```
  - EXEC SQL 키워드 사용
    ```
    EXEC SQL begin declare section;
    char *desc;
    EXEC SQL end declare section;
    ```
- 사용
  - SQL문 내에서
    ```
    EXEC SQL update customer set zipcode = :zipcode ;
    ```
  - C 구문 안에서
    ```
    gets (dbname);
    EXEC SQL database :dbname;
    ```
- external 이나 static으로 선언하지 않은 경우 자동적으로 local 범위

```
$ extern short StockNumS ;

func1()
$ double StockNumS ;

  $ {
  $ long StockNumS ;
    :
  $ }

  func2()
  {
  $ insert into stock ...
     values ($StockNumS);
    :
  }
```

Database Connections

- ESQL/C 구문
  - CONNECT TO 데이터베이스명
  - DISCONNECT ALL
- CONNECT구문의 옵션
  - Connection 이름
    - EXEC SQL CONNECT TO ‘stores@munish’ AS ‘munich_con’ ;
  - User 절
    - EXEC SQL CONNECT TO ‘stores@munish’ USER :uid USING :passwd ;
  - Default 절 (데이터베이스를 오픈하지 않고 서버에 연결)
    - EXEC SQL CONNECT TO DEFAULT ;
    - Default절 사용 후 다음과 같은 문장이 필요
      - ✓ DATABASE
      - ✓ CREATE DATABASE
      - ✓ START DATABASE
- 연결 전환 (switching)
  - 구문

    EXEC SQL SET CONNECTION ‘munish_con’ ;
    EXEC SQL SET CONNECTION :connect_id ;
    EXEC SQL SET CONNECTION DEFAULT;
- 연결 종료 (disconnecting)
  - 열린 데이터베이스를 닫고 connection을 종료한다

    EXEC SQL DISCONNECT :connect_id ;
    EXEC SQL DISCONNECT CURRENT ;
    EXEC SQL DISCONNECT DEFAULT ;
    EXEC SQL DISCONNECT ALL ;
  - default로 connect했거나 connection이 하나 일 경우 DEFAULT 사용

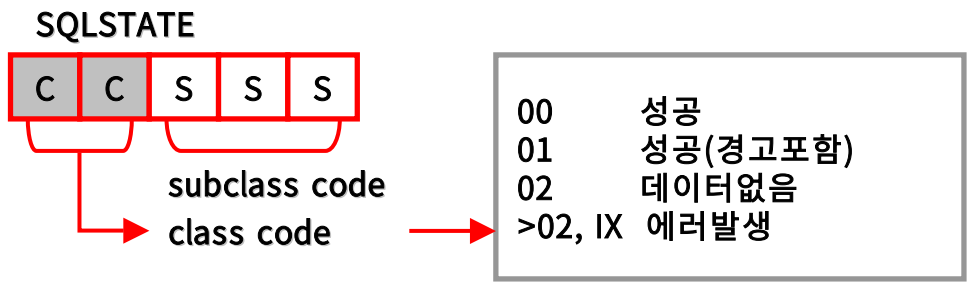
Error Handling

- SQL문을 실행한 후 서버에서 SQLCA라 불리는 영역에 feedback정보를 남긴다
  - errors
  - performance
  - warnings
- SQLCODE 는 sqlca.sqlcode 값과 같다
  - SQLCODE < 0 : 에러 코드 값
  - SQLCODE = 0 : 정상적 처리
  - SQLCODE = 100 : SQLNOTFOUND, 찾는 행이 없음
  - SQLCODE = 1~99 : Dynamic SQL
- 에러 메시지를 얻기 위한 함수
  - rgetmsg, rgetlmsg
- SQL문이 실행되고 다음과 같은 4가지 중 하나의 상태가 된다
  - 성공
  - 성공, 찾을 수 있는 데이터는 없음
  - 성공, 경고(warning) 메시지가 있음
  - 실패
- 성공 외의 나머지 상태에 대한 처리 방법을 exception처리로 제어한다
  - WHENEVER exception action
    - exception : NOT FOUND, SQLWARNING, SQLERROR
    - action : GO TO label, CALL function, STOP, CONTINUE
  - 디폴트 action은 CONTINUE
- WHENEVER문의 scope는 한 소스 파일에서 global 성격을 가진다

```
struct sqlca_s {
    long  sqlcode ;
    char sqlerrm[72] ;
    char sqlerrp[8] ;
    long sqlerrd[6] ;
    struct sqlcaw_s {
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
        char sqlwarn0 ;
    } sqlwarn ;
} ;
extern struct sqlca_s sqlca;
```

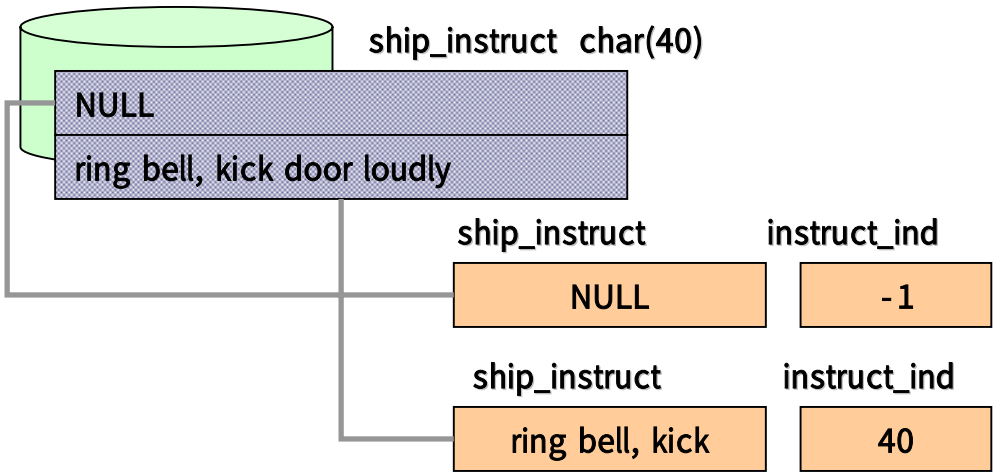
Error Handling

- GET DIAGNOSTICS 사용
  - GET DIAGNOSTICS 문은 여러 개의 SQL exception을 처리할 수 있고, ANSI 및 X/OPEN 표준으로 컴파일된다
  - SQLCA 는 계속 사용 가능하다
  - 다음 세가지 분류로 사용한다
    - SQLSTATE
    - GET DIAGNOSTICS 문
    - GET DIAGNOSTICS EXCEPTION 문
  - GET DIAGNOSTICS 의 정보
    - MORE : 모든 exception이 반환되었음을 나타냄
    - NUMBER : exception의 갯수
    - ROW_COUNT : SQL에 의해 처리된 행 수
  - EXCEPTION 의 정보
    - RETURNED_SQLSTATE
    - SERVER_NAME
    - CONNECTION_NAME
    - CLASS_ORIGIN 등



Error Handling

- Indicator 사용
  - 데이터를 호스트변수에 저장할 때 다음과 같은 Exception이 발생할 수 있다
    - NULL값을 fetch 하였다
    - 컬럼 데이터에 비해 호스트 변수의 크기가 작다
  - 사용법
    - :host변수:indicator변수
    - $host변수$indicator변수
    - :host변수 INDICATOR indicator변수 (ANSI 표준)



Prepare, Execute

- Prepare / Execute 구문
  PREPARE prepare_statement_id FROM { “quoted_string” | :host_variable } ;
  EXECUTE prepare_statement_id [USING :host_variables] ;

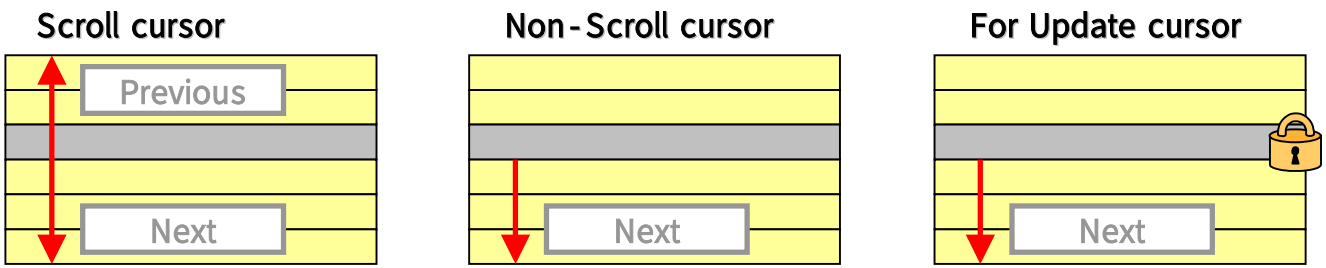
- 단일 SQL문 처리 예
  EXEC SQL insert into customer (customer_num, fname, lname, company)
  values (0, :fname, :lname, ‘MPS Corp’);
  EXEC SQL insert into items values (:itemsrec);
  ※ itemsrec 변수는 items테이블의 컬럼 순서로 선언된 structure 형

- PREPARE / EXECUTE 사용
  EXEC SQL PREPARE ins_p from
  “insert into customer (customer_num, fname, lname, company)
  values (0,?,?,?)” ;
  EXEC SQL EXECUTE ins_p using :fname, :lname, :company;

- PREPARE를 사용하는 이유
  - PREPARE 시점에 SQL문장에 대하여 미리 parsing되고 query plan이 생성된다
  - EXECUTE 시점에는 처리해야 할 값만 넘겨주면 즉시 실행된다
  - 하나의 DML문장에서 여러번 값을 바꿔서 넘겨주어야 할 때 PREPARE에서 한 번만 query plan을 생성하므로 유용하게 사용할 수 있다

Cursor

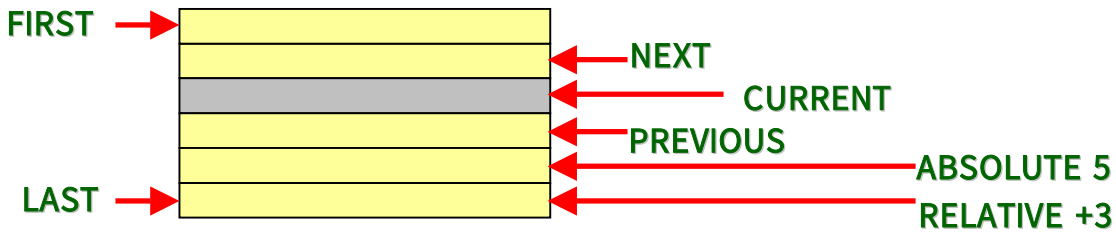
- Cursor의 이해
  - Cursor 는 select구문에 의해 선택된 row들을 가리키고 있는 일종의 marker라고 볼 수 있음.
  - 여러 개의 row를 가져오는 select문의 결과에 대해 처리하기 위해서 필요함.
  - Cursor의 종류
    - Select cursor
      - ✓ Scroll cursor
      - ✓ Non-scroll cursor
      - ✓ For update cursor
    - Insert cursor



Scroll Cursor

- 구문
  - DECLARE
    DECLARE cursor_id SCROLL CURSOR [WITH HOLD] FOR select_stmt ;
  - OPEN
    OPEN cursor_id ;
  - FETCH
    FETCH [position] cursor_id [INTO host_variable_list] ;

    Position : FIRST, LAST, CURRENT, NEXT, PREVIOUS, ABSOLUTE n, RELATIVE n



  - CLOSE
    CLOSE cursor_id ;
  - FREE
    FREE cursor_id ;

Scroll Cursor

- WITH HOLD 구문
  - WITH HOLD구문을 사용한 커서는 transaction이 종료되어도 close되지 않는다.
- Fetch Buffer Size
  - 환경변수 : export FET_BUF_SIZE=30000
  - 전역변수 : EXEC SQL include sqlhdr ;
    FetBufSize=30000 ;
- Deferred Prepare
  - 환경변수 : export IFX_DEFERRED_PREPARE=1 (enable)
    export IFX_DEFERRED_PREPARE=0 (disable)
  - SQL구문 : EXEC SQL set deferred_prepare ;
    EXEC SQL set deferred_prepare enabled ;
    EXEC SQL set deferred_prepare disabled ;
- Open-Fetch-Close
  - 환경변수 : export OPTOFC=1 (enable)
    export OPTOFC=0 (disable)
- Autofree
  - 환경변수 : export IFX_AUTOFREE=1 (enable)
    export IFX_AUTOFREE=0 (disable)
  - SQL구문 : EXEC SQL set autofree;
    EXEC SQL set autofree enabled ;
    EXEC SQL set autofree disabled ;

```
declare cursor
open
begin work
:
commit work;
fetch
close
free
```

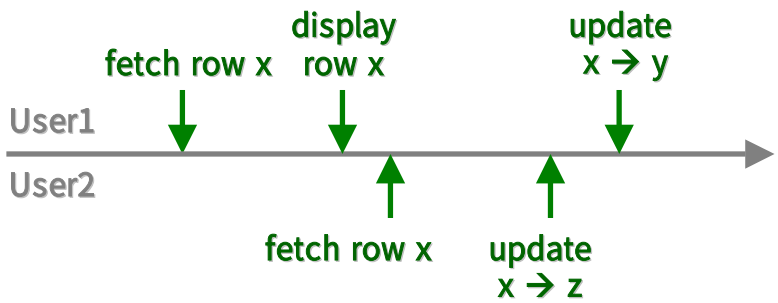
```
declare cursor with hold
open
begin work
:
commit work;
fetch
close
free
```

```
EXEC SQL free stmt_id;
EXEC SQL close cur_id;
EXEC SQL free cur_id;
```

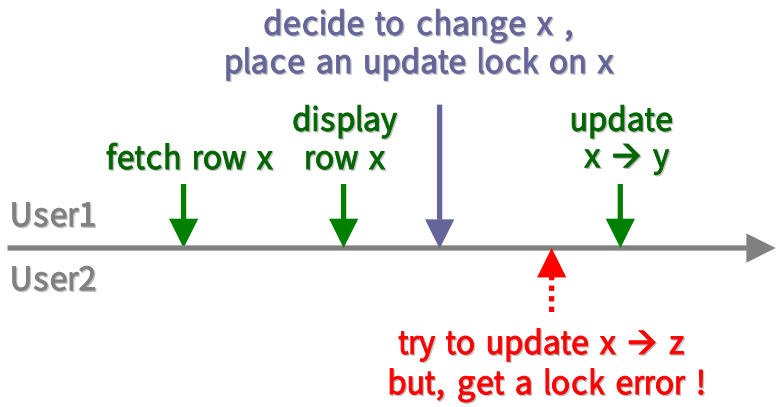
```
EXEC SQL set autofree;
        :
EXEC SQL close cur_id;
```

For Update Cursor

- 구문
  - DECLARE
    DECLARE cursor_id CURSOR [WITH HOLD] FOR select_stmt FOR UPDATE ;
  - OPEN
    OPEN cursor_id ;
  - FETCH
    FETCH cursor_id [INTO host_variable_list] ;
  - UPDATE
    UPDATE ... WHERE CURRENT OF cursor_id ;
  - CLOSE
    CLOSE cursor_id ;
  - FREE
    FREE cursor_id ;



No Lock Used



Using For Update Cursor

Insert Cursor

- 구문
  - DECLARE
    DECLARE cursor_id CURSOR FOR insert_stmt ;
  - OPEN
    OPEN cursor_id ;
  - PUT
    PUT cursor_id [FROM host_variable_list] ;
  - FLUSH
    FLUSH cursor_id ;
  - CLOSE
    CLOSE cursor_id ;
  - FREE
    FREE cursor_id ;

Concurrency Control

- Isolation
- Lock

Isolation – Read Concurrency Control

- ANSI SQL-92 Transaction Isolation (SET TRANSACTION)
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable Read

- ANSI표준
- access mode지원
- 트랜잭션당 한번만
- 트랜잭션이 끝날 때까지 유효
- 트랜잭션에서만 사용

- Informix Isolation (SET ISOLATION)
  - Dirty Read
  - Committed Read
  - Cursor Stability
  - Repeatable Read

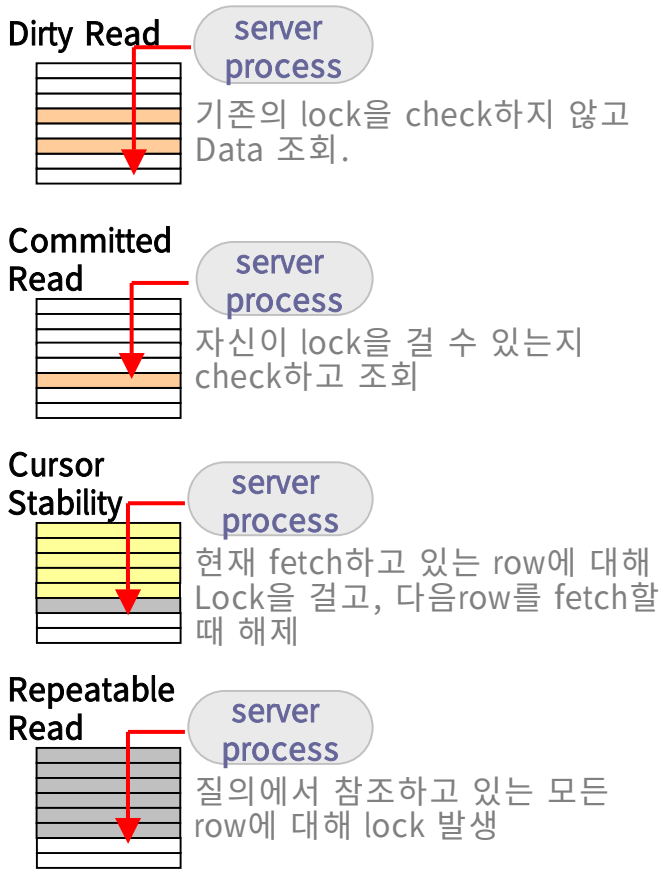
- non-ANSI
- 트랜잭션에서 전환가능
- 트랜잭션이 끝나거나 다른 SET ISOLATION구문을 실행할 때까지 유효
- 로깅모드 데이터베이스에서 언제든 사용 가능

- Access Mode
  - SET TRANSACTION READ WRITE ;
  - SET TRANSACTION READ ONLY ;

ANSI Levels	SET TRANSACTION	SET ISOLATION
Read Uncommitted	READ UNCOMMITTED	DIRTY READ
Read Committed	READ COMMITTED	COMMITTED READ
N/A	N/A	CURSOR STABILITY
Repeatable Read	REPEATABLE READ	REPEATABLE READ
Serializable	SERIALIZABLE	REPEATABLE READ

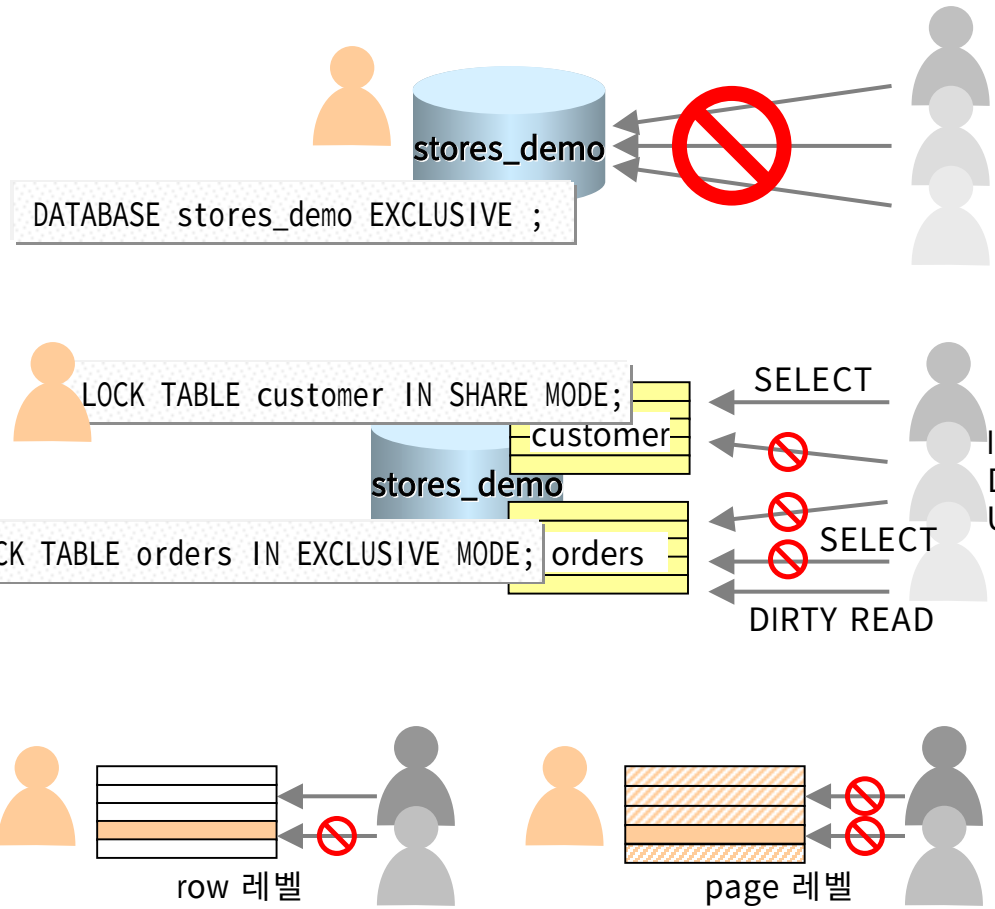
Isolation – Read Concurrency Control

- Read Uncommitted / Dirty Read
  - SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ;
  - SET ISOLATION TO DIRTY READ ;
- Read Committed / Committed Read
  - SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
  - SET ISOLATION TO COMMITTED READ ;
- Cursor Stability
  - SET ISOLATION TO CURSOR STABILITY ;
- Serializable / Repeatable
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
  - SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
  - SET ISOLATION TO REPEATABLE READ ;



Lock – Update Concurrency Control

- Lock의 적용 단위
  - Database 레벨
  - Table 레벨
  - Page 레벨
  - Row 레벨
  - Key 레벨

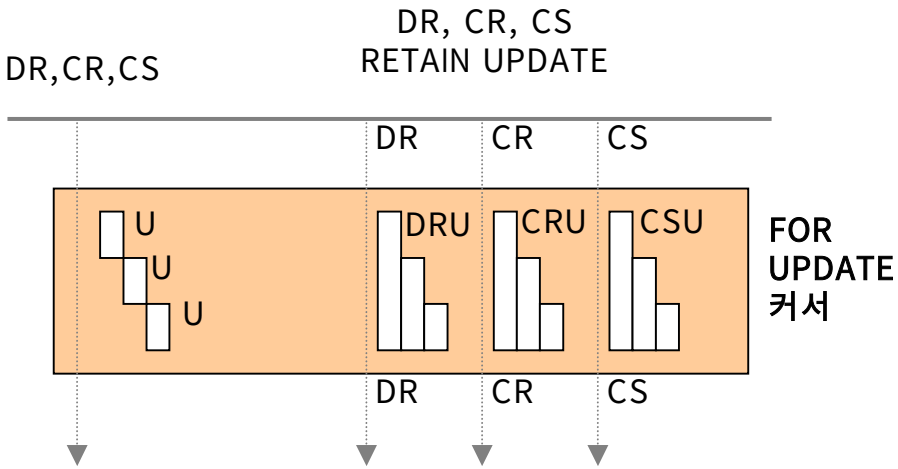


Lock – Update Concurrency Control

- Lock의 종류
  - Shared Lock
  - Exclusive Lock
  - Update Lock
- Lock 모드 설정
  - SET LOCK MODE TO NOT WAIT;
  - SET LOCK MODE TO WAIT 20;
  - SET LOCK MODE TO WAIT;

Lock – Update Concurrency Control

- Retain Update Lock
  - SET ISOLATION TO DIRTY READ RETAIN UPDATE LOCKS;
  - SET ISOLATION TO COMMITTED READ RETAIN UPDATE LOCKS;
  - SET ISOLATION TO CURSOR STABILITY RETAIN UPDATE LOCKS;



- Key value locking
  - 인덱스가 있는 데이터를 update,delete,insert 할 때 인덱스 Key에 locking하는 기법

key value	rowid	delete flag

0 : Not deleted
1 : Deleted

Query Optimization

- Cost Based Query Optimizer
- Explain
- Directives
- Update Statistics

Cost-based Query Optimizer

- Query Plan
  - Optimizer가 선택하는 access plan과 join plan으로 결정
  - 조회할 페이지를 줄이고 불필요한 정렬 작업을 최소화하도록 선택

- Access Plan 작성 단계
  - 각 필터를 통해 예상되는 데이터 출력 건수 산정 (selectivity)
  - 인덱스 사용 가능 여부 검사
    - filter 컬럼
    - ORDER BY / GROUP BY 컬럼
  - 최적의 access 방안 선택
    - sequentially
    - by an index

- Join Plan 작성 단계
  - 테이블간의 join 조합 구성
  - 각 조합별로 I/O 및 CPU cost 산정
  - 부하가 많은 (cost가 크다. expensive) 조합은 plan에서 제거

Explain - Viewing the Query Plan

- Query plan 확인
  - SET EXPLAIN { ON | OFF } ;
- EXPLAIN을 ON시키고 SQL구문을 실행하면 query plan을 확인 할 수 있는 텍스트 파일이 생성된다.
  - 예상 비용 (estimate cost)
  - 테이블 접근 순서
  - 임시테이블 사용 정보
  - access method / join method
- Explain 파일
  - UNIX : ${PWD}/sqexplain.out 또는 ${HOME}/sqexplain.out
  - NT : %INFORMIXDIR% \ sqexpln \ username.out

Explain - examples

- Sequential Scan with Temporary Table
- Sequential Scan with Filter
- Key-only Index Scan

Explain - examples

- Index Scan with Lower Index Filter
- Index Scan with Lower and Upper Index Filters

Explain - examples

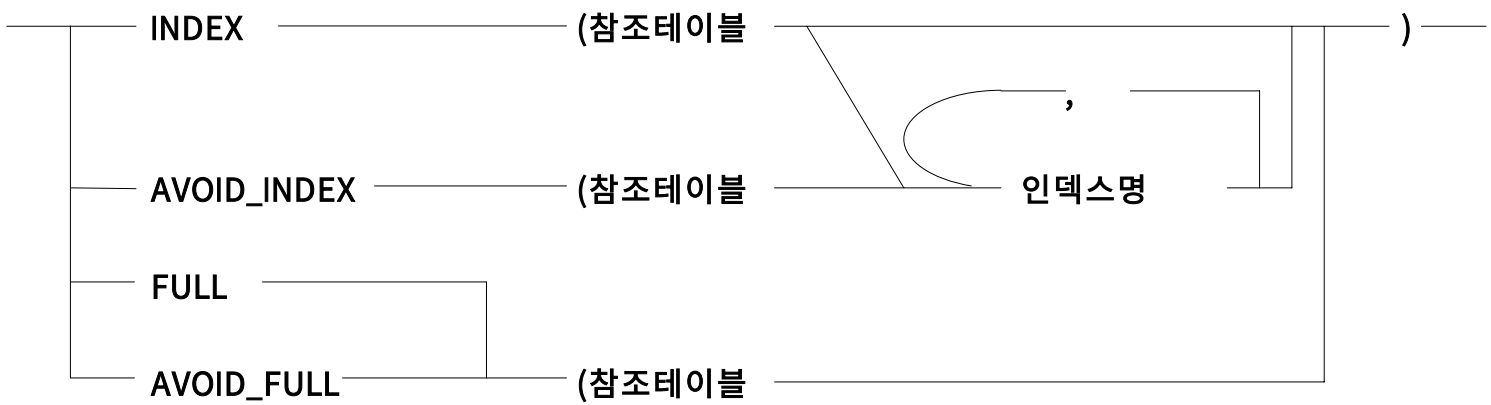
- Dynamic Hash Join
- Key-First Index Scan

Directives

- SQL에 직접적으로 optimizing 방법을 명시하는 구문을 optimizer directives 라고 한다
  - Positive Directives
  - Negative Directives
- Optimizer Directives의 종류
  - Access method directives
  - Join order directives
  - Join method directives
  - Optimization goal directives
  - Explain diretivies
- SQL comment 문자 뒤에 + 기호를 붙여서 directives를 나타낸다
  - --+ directives 구문
  - {+ directives 구문 }
  - /*+ directives 구문 */

Directives

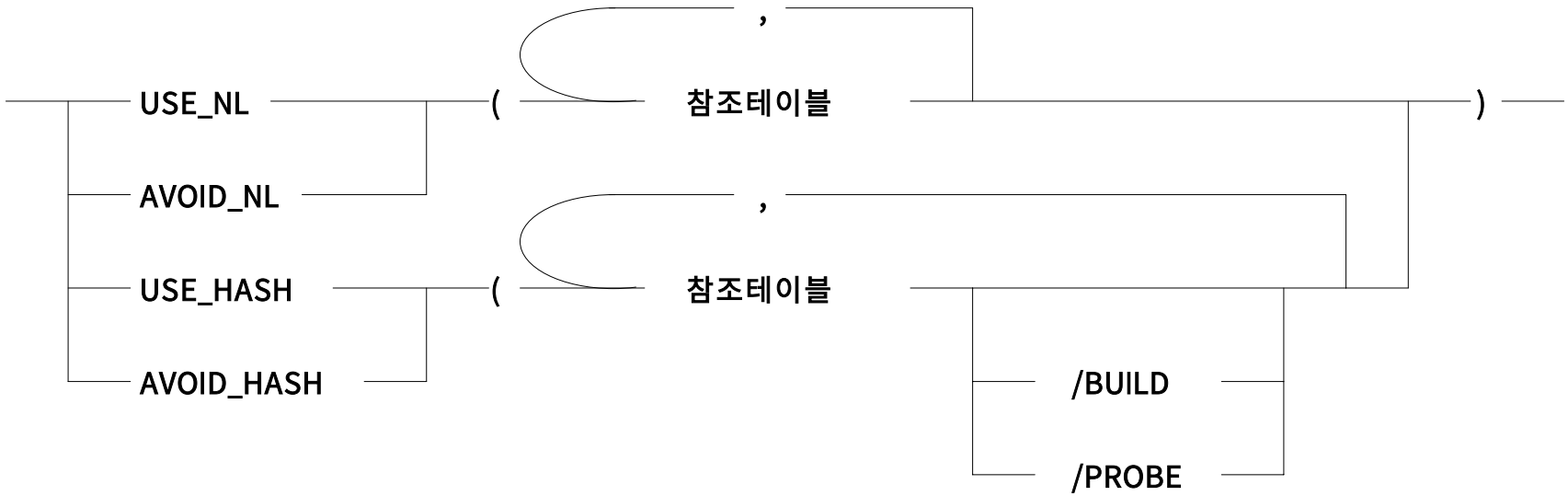
- Access Method Directives
  - INDEX
  - AVOID_INDEX
  - FULL
  - AVOID_FULL



※ 참조테이블 : table, synonym, alias

Directives

- Join Method Directives
  - USE_NL
  - AVOID_NL
  - USE_HASH
  - AVOID_HASH



Directives

- Join Order Directives
  - ORDERED
- Optimization Goal Directives
  - FIRST_ROWS
  - ALL_ROWS (default)
- Explain Directives
  - EXPLAIN
  - EXPLAIN AVOID_EXECUTE

Update Statistics

- UPDATE STATISTICS구문을 사용하여 시스템 카타로그 테이블의 통계 정보를 갱신한다
- 구문
  - 데이터베이스의 모든 오브젝트에 대하여 수행
    UPDATE STATISTICS [LOW | MEDIUM | HIGH] ;
  - 특정 테이블과 그 테이블에 생성된 인덱스에 대하여 수행
    UPDATE STATISTICS [LOW | MEDIUM | HIGH] FOR TABLE [table_name] ;
  - 특정 컬럼에 대하여 수행
    UPDATE STATISTICS [LOW | MEDIUM | HIGH] FOR TABLE table_name (column_name) ;

Update Statistics Guideline

- 데이터베이스 전체 또는 테이블별로 LOW 모드로 수행하는 것이 일반적
- LOW 모드로 수행하였을 때 만족스럽지 못하다면 다음 단계 수행
  - 대용량 테이블의 경우
    1. 테이블별로 MEDIUM 모드, distribution only 옵션으로 수행
    2. 인덱스의 첫번째 컬럼에 대하여 HIGH 모드 수행
    3. Multi-column으로 생성된 인덱스 고려
    4. 기타 컬럼에 대하여 컬럼 단위로 LOW 모드 수행
  - 소규모 테이블의 경우
    테이블 단위로 HIGH 모드 수행

- 다음과 같은 경우 반드시 UPDATE STATISTICS를 수행해야 한다
  - 데이터를 테이블에 대량 로드 한 후
  - 데이터 수정이 상당히 많아서 데이터 분포가 변경 되었을 때
  - 테이블의 대부분의 행에서 입력 또는 삭제 작업이 일어났을 때
  - 인덱스를 추가하거나 삭제 또는 변경하였을 때

Homepage Supports

http://www.ibm.com/kr/informix

http://www.ibm.com/kr/informix

- 특정 날짜가 몇 월 몇 째주인지 알고 싶을 때 사용할 수 있는 stored procedure.
- PHP와 연동 중 –439에러가 발생하는 원인
- JDBC를 사용한 application의 메모리 증가를 최소화 하기 위한 방안
- 현재 lock을 가지고 있는 테이블 또는 세션을 확인
- dbspace의 사용 상태를 sysmaster query를 통하여 모니터링
- 각 테이블 별 extent 개수 및 사용량
- 청크 안에 저장된 테이블 정보
- unique 컬럼으로 사용할 컬럼에 duplication 데이터 삭제 방법
- 특정일 기준으로 몇 달 전,후 날짜 출력
- 두 시간 차이를 표현 하거나 두 날짜 사이를 일 수로 표현
- 등등...