Database Systems 2

Lecture 5

Advanced SQL 1

Data Definition

Lecture - Objectives

- Data types
- The Dataset Hierarchy
- Create Table Statement
- Constraints
- Alter Table Statement
- Named Constraints
- Drop Table Statement

ISO SQL Data Types (SQL 92)

Table 6.1 ISO SQL data types.

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Connolly T and Begg C., 2004. *Database Systems*. 4th ed. Addison Wesley

SQL 2003 Datatypes

Table 2-8. SQL2003 categories and datatypes

Category	Example datatypes and abbreviations	Description
BINARY	*BINARY LARGE OBJECT (BLOB)*	This datatype stores binary string values in hexadecimal format. Binary string values are stored without reference to any character set and without any length limit.
BOOLEAN	*BOOLEAN*	This datatype stores truth values (either *TRUE* or *FALSE*).
CHARACTER string types	*CHAR* *CHARACTER VARYING (VARCHAR)*	These datatypes can store any combination of characters from the applicable character set. The varying datatypes allow variable lengths, while the other datatypes allow only fixed lengths. Also, the variable-length datatypes automatically trim trailing spaces, while the other datatypes pad all open space.
	NATIONAL CHARACTER (NCHAR) *NATIONAL CHARACTER VARYING (NCHAR VARYING)*	The national character datatypes are designed to support a particular implementation-defined character set.
	CHARACTER LARGE OBJECT (CLOB)	*CHARACTER LARGE OBJECT* and *BINARY LARGE OBJECT* are collectively referred to as *large object string types*.
	NATIONAL CHARACTER LARGE OBJECT (NCLOB)	Same as *CHARACTER LARGE OBJECT*, but supports a particular implementation-defined character set.
DATALINK	*DATALINK*	Defines a reference to a file or other external data source that is not part of the SQL environment.
INTERVAL	*INTERVAL*	Specifies a set of time values or span of time.
COLLECTION	*ARRAY* *MULTISET*	*ARRAY* was offered in SQL99, and *MULTISET* was added in SQL2003. Whereas an *ARRAY* is a set-length, ordered collection of elements, *MULTISET* is a variable-length, unordered collection of elements. The elements in an *ARRAY* and a *MULTISET* must be of a predefined datatype.

Kline K, 2004, ***SQL in a Nutshell,*** 2nd ed. O'Reilly

Table 2-8. SQL2003 categories and datatypes (continued)

Category	Example datatypes and abbreviations	Description
NUMERIC	*INTEGER (INT)* *SMALLINT* *BIGINT* *NUMERIC(p,s)* *DEC[IMAL](p,s)* *FLOAT(p,s)* *REAL* *DOUBLE PRECISION*	These datatypes store exact numeric values (integers or decimals) or approximate (floating-point) values. *INT*, *BIGINT*, and *SMALLINT* store exact numeric values with a predefined precision and a scale of zero. *NUMERIC* and *DEC* store exact numeric values with a definable precision and a definable scale. *FLOAT* stores approximate numeric values with a definable precision, while *REAL* and *DOUBLE PRECISION* have predefined precisions. You may define a precision (*p*) and scale (*s*) for a *DECIMAL*, *FLOAT*, or *NUMERIC* datatype to indicate the total number of allowed digits and the number of decimal places, respectively. *INT*, *SMALLINT*, and *DEC* are sometimes referred to as *exact numeric types*, while *FLOAT*, *REAL*, and *DOUBLE PRECISION* are sometimes called *approximate numeric types*.
TEMPORAL	*DATE* *TIME* *TIME WITH TIME ZONE* *TIMESTAMP* *TIMESTAMP WITH TIME ZONE*	These datatypes handle values related to time. *DATE* and *TIME* are self-explanatory. Datatypes with the *WITH TIME ZONE* suffix also include a time zone offset. The *TIMESTAMP* datatypes are used to store a value that represents a precise moment in time. Temporal types are also known as *datetime types*.
XML	*XML*	Stores XML data and can be used wherever a SQL datatype is allowed (e.g., for a column of a table, a field in a row, etc.). Operations on the values of an XML type assume a tree-based internal data structure. The internal data structure is based on the XML Information Set Recommendation (Infoset), using a new document information item called the *XML root information item*.

Not everyone adheres to the standards...

Table 2-9. Comparison of platform-specific datatypes

Vendor datatype	MySQL	Oracle	PostgreSQL	SQL Server	SQL2003 datatype
BFILE		Y			None
BIGINT	Y		Y	Y	*BIGINT*
BINARY	Y			Y	*BLOB*
BINARY_FLOAT		Y			*FLOAT*
BINARY_DOUBLE		Y			*DOUBLE PRECISION*
BIT	Y		Y	Y	None
BIT VARYING, VARBIT			Y		None
BLOB	Y	Y			*BLOB*
BOOL, BOOLEAN	Y		Y		*BOOLEAN*
BOX			Y		None
BYTEA			Y		*BLOB*
CHAR, CHARACTER	Y	Y	Y	Y	*CHARACTER*
CHAR FOR BIT DATA					None
CIDR			Y		None
CIRCLE			Y		None
CLOB		Y			*CLOB*
CURSOR				Y	None
DATALINK					*DATALINK*
DATE	Y	Y	Y	Y	*DATE*
DATETIME	Y			Y	*TIMESTAMP*

This is just an extract....

Data Definition

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.

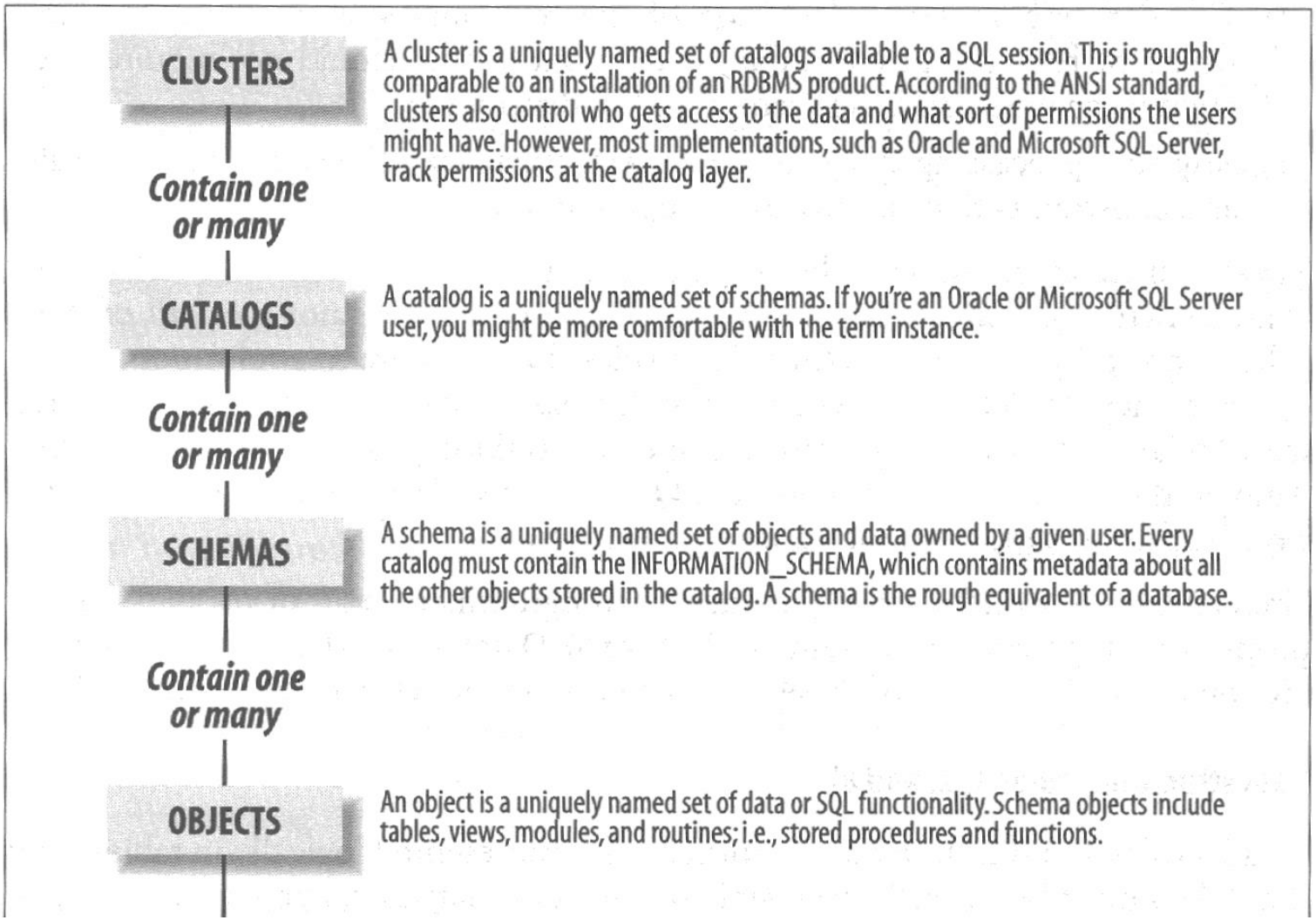
- Main SQL DDL statements are:

`CREATE SCHEMA`	`DROP SCHEMA`
`CREATE/ALTER DOMAIN`	`DROP DOMAIN`
`CREATE/ALTER TABLE`	`DROP TABLE`
`CREATE VIEW`	`DROP VIEW`

- Many DBMSs also provide:

`CREATE INDEX`	`DROP INDEX`

SQL 2003 Dataset Hierarchy



CREATE SCHEMA

```
CREATE SCHEMA [Name | AUTHORIZATION CreatorId ]

DROP SCHEMA Name [RESTRICT | CASCADE ]
```

- With RESTRICT (default), schema must be empty or operation fails.
- With CASCADE, operation cascades to drop all objects associated with schema in order defined above. If any of these operations fail, DROP SCHEMA fails.

Oracle doesn't follow the standard here

CREATE SCHEMA

Purpose

Use the `CREATE SCHEMA` statement to create multiple tables and views and perform multiple grants in your own schema in a single transaction.

To execute a `CREATE SCHEMA` statement, Oracle Database executes each included statement. If all statements execute successfully, then the database commits the transaction. If any statement results in an error, then the database rolls back all the statements.

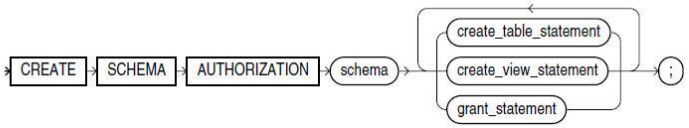
> **Note:** This statement does not actually create a schema. Oracle Database automatically creates a schema when you create a user (see CREATE USER on page 17-7). This statement lets you populate your schema with tables and views and grant privileges on those objects without having to issue multiple SQL statements in multiple transactions.

Prerequisites

The `CREATE SCHEMA` statement can include `CREATE TABLE`, `CREATE VIEW`, and `GRANT` statements. To issue a `CREATE SCHEMA` statement, you must have the privileges necessary to issue the included statements.

Syntax

***create_schema*::=**



From Oracle SQL Reference Doc E17118-05.pdf

CREATE TABLE

Note:

Language syntax definitions tend to use a variant of EBNF notation. (Extended Backus-Naur Form).

A term in italics is defined elsewhere

{ } means "repeated 1 or more times "

[ ] means "is optional"

| means "either - or"

First, the short, abbreviated version.

CREATE TABLE

```
CREATE TABLE TableName
(
    { colName dataType [NOT NULL]
                       [UNIQUE]
                       [DEFAULT defaultOption]
                       [CHECK searchCondition]
                       [,...]
    }
    [PRIMARY KEY (listOfColumns),]
    [ {
        FOREIGN KEY (listOfFKColumns)
                       REFERENCES ParentTableName [(listOfCKColumns)]
             [ON UPDATE referentialAction ]
             [ON DELETE referentialAction ]

        [,…]
    }   ]
)
```

A repeating list of column definitions, each one optionally followed by inline constraints, and followed by a comma (if there is another line).

An optional out-line definition of a Primary Key constraint (not repeating, because there can only be one)

An optional repeating list of Foreign Key constraints, possibly including referential action clauses. Separated by commas

Red indicated a keyword or symbol which actually appears in the SQL command.

CREATE TABLE

- Creates a table with one or more columns of the specified *dataType*.
- With NOT NULL, system rejects any attempt to insert a row with a null in that column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action

```
CREATE [GLOBAL] [TEMPORARY] TABLE table_name
[ ( {column | attribute} [SORT] [DEFAULT expression] [{column_constraint |
      inline_ref_constraint} ] |
   {table_constraint_clause | table_ref_constraint} |
   {GROUP log_group (column [NO LOG] [,...]) [ALWAYS] | DATA
      (constraints [,...]) COLUMNS} ) ]
[ON COMMIT {DELETE | PRESERVE} ROWS]
[table_constraint_clause]
{  [physical_attributes_clause] [TABLESPACE tablespace_name]
      [storage_clause] [[NO]LOGGING] |
   [CLUSTER (column [,...] )] |
   {[ORGANIZATION
     {HEAP [physical_attributes_clause][TABLESPACE
        tablespace_name][storage_clause]
         [COMPRESS | NOCOMPRESS] [[NO]LOGGING] |
      INDEX [physical_attributes_clause][TABLESPACE
         tablespace_name][storage_clause]
         [PCTTHRESHOLD int] [COMPRESS [int] | NOCOMPRESS]
         [MAPPING TABLE | NOMAPPING][...] [[NO]LOGGING]
         [[INCLUDING column] OVERFLOW
            [physical_attributes_clause][TABLESPACE tablespace_name]
            [storage_clause] [[NO]LOGGING] } ] |
      EXTERNAL ( [TYPE driver_type] ) DEFAULT DIRECTORY directory_name
         [ACCESS PARAMETERS {USING CLOB subquery | ( opaque_format ) } ]
         LOCATION ( [directory_name:]'location_spec' [,...] )
         [REJECT LIMIT {int | UNLIMITED} ] } }
[{ENABLE | DISABLE} ROW MOVEMENT]
[[NO]CACHE] [[NO]MONITORING] [[NO]ROWDEPENDENCIES]
[PARALLEL int | NOPARALLEL] [NOSORT] [[NO]LOGGING]]
[COMPRESS [int] | NOCOMPRESS]
[{ENABLE | DISABLE} [[NO]VALIDATE]
   {UNIQUE (column [,...] | PRIMARY KEY | CONSTRAINT constraint_name} ]
   [USING INDEX  {index_name | CREATE_INDEX_statement} ] [EXCEPTIONS INTO]
   [CASCADE] [{KEEP | DROP} INDEX] ]  |
[partition_clause]
[AS subquery]
```

- This is the FULL syntax for the Oracle CREATE TABLE command

Other reference sources

See the online Oracle documentation:

http://docs.oracle.com/cd/E17952_01/refman-5.1-en/create-table.html

Or for the syntax diagram approach:

http://docs.oracle.com/cd/E11882_01/server.112/e41084/statements_7002.htm#i2095331

Example 1 - CREATE TABLE

```
CREATE TABLE PropertyForRent
(
  propertyNo          VARCHAR(5),
   rooms              SMALLINT,
  rent                DECIMAL(6,2),
  ownerNo             VARCHAR(5),
  staffNo             VARCHAR(5),
  branchNo            CHAR(4),
  PRIMARY KEY (propertyNo),
  FOREIGN KEY (staffNo) REFERENCES staff(staffNo)
);
```

These are called out-of-line constraints.

Example 2 - CREATE TABLE

```
CREATE TABLE PropertyForRent
(
  propertyNo          VARCHAR(5) PRIMARY KEY,
   rooms              SMALLINT,
  rent                DECIMAL(6,2),
  ownerNo             VARCHAR(5),
  staffNo             VARCHAR(5) REFERENCES staff(staffNo),
  branchNo            CHAR(4)
  );
```

These are called inline constraints.

CONSTRAINTS

Constraints in Tables

Why do we need constraints?

Do the constraints create the join, or does the SELECT query create the join?

You don't need to create the constraints at the same time as you create the table.

You can use ALTER TABLE to add them later.

What Types of Constraint Are There?

- There are five types of integrity constraints:

a. Required data.
b. Domain constraints.
c. Entity integrity.
d. Referential integrity.
e. Enterprise constraints.

Integrity Enhancement Feature

a. Required Data

```
position VARCHAR(10) NOT NULL
```

b. Domain Constraints

(aka Check Constraints)

```
sex     VARCHAR(1)  CHECK (sex IN ('M', 'F'))
```

c. Entity Integrity

- Primary key of a table must contain a unique, non-null value for each row.

- ISO standard supports PRIMARY KEY clause in CREATE and ALTER TABLE statements:

```
PRIMARY KEY(staffNo)
PRIMARY KEY(clientNo, propertyNo)
```

- Can only have one PRIMARY KEY clause per table. Can still ensure uniqueness for other fields using UNIQUE:

```
telno     CHAR(5)     UNIQUE,
```

d. Referential Integrity

- FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.

- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.

- ISO SQL standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

```
FOREIGN KEY(branchNo) REFERENCES Branch(branchNo)
```

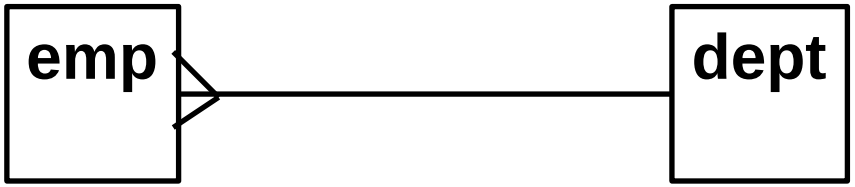
d. Referential Integrity

- Any INSERT/UPDATE that attempts to create FK value in child table without matching candidate key value in parent is rejected.

- Action taken that attempts to update/delete a candidate key value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

CASCADE	SET NULL
SET DEFAULT	NO ACTION

Delete Cascade function



Assuming all your **primary and foreign key** constraints are in place and **enabled**:

What happens if you try to delete deptno = 10 from your dept table?

Delete Cascade function

DELETE FROM mydept
WHERE deptno = 10;

You should get something like the following error message.

ERROR at line 1:
ORA-02292: integrity constraint (user.SYS_Cnnnnnn) violated - child record Found

This means that if you delete deptno 10 from the dept table, all the records in the emp table with the value 10 in the deptno foreign key will invalidate referential integrity, thus Oracle will not let you delete the record.

Oracle provides referential integrity during deletion, BUT we can use DELETE CASCADE – so when a master row is deleted, all detail rows are deleted as well.

FOREIGN KEY (deptno) REFERENCES mydept(deptno)
ON DELETE CASCADE;

Now if you delete deptno 10, Oracle will delete all records in emp which are related to that record.

You could also have said ON DELETE SET NULL

or ON DELETE SET DEFAULT

```
FOREIGN KEY (staff_no) REFERENCES staff(staff_no)
ON DELETE CASCADE
```

OR

```
FOREIGN KEY (staff_no) REFERENCES staff(staff_no)
ON DELETE SET DEFAULT
```

OR

```
FOREIGN KEY (staff_no) REFERENCES staff(staff_no)
ON DELETE SET NULL
```

OR

```
FOREIGN KEY (staff_no) REFERENCES staff(staff_no)
ON DELETE NO ACTION
```

e. Enterprise Constraints

Rules of the business or enterprise being modelled.

eg. dist_id should be two letters, followed by two letters or numbers.

You can usually use a CHECK constraint to implement these.

```
CREATE TABLE distributors
  (
    dist_id   CHAR(4) PRIMARY KEY
        CHECK  (dist_id  LIKE  '[A-Z][A-Z][A-Z][A-Z]'  OR
                dist_id LIKE '[A-Z][A-Z][0-9][0-9]'),
  dist_name   VARCHAR(40)
  );
```

Example of Constraints

```
CREATE TABLE PropertyForRent
(
   propertyNo   VARCHAR(5)    PRIMARY KEY,
   rooms        SMALLINT      CHECK(VALUE BETWEEN 1 AND 15)  NOT NULL  DEFAULT 4,
   rent         DECIMAL(6,2)  CHECK(VALUE BETWEEN 0 AND 9999.99) NOT NULL DEFAULT 600,
   ownerNo      VARCHAR(5)    CHECK (VALUE IN (SELECT ownerNo FROM Owner)) NOT NULL,
   staffNo      VARCHAR(5)    CHECK (VALUE IN (SELECT staffNo FROM Staff)),
   branchNo     CHAR(4)       CHECK (VALUE IN (SELECT BranchNo FROM Branch)) NOT NULL
);
```

Do I also need to add this constraint?

```
FOREIGN KEY (staffNo) REFERENCES Staff(StaffNo)
```

What am I actually saying about ownerNo and branchNo above?

ALTER TABLE

ALTER TABLE

- Add a new column to a table.
- Drop a column from a table.
- Set a default for a column.
- Drop a default for a column.

You can also:

- Add a new table constraint.
- Drop a table constraint.

Example - ALTER TABLE

```
ALTER TABLE Staff
      ADD middle_name VARCHAR(30);

ALTER TABLE Staff
      DROP middle_name VARCHAR(30);


ALTER TABLE Staff
      ALTER position DROP DEFAULT;

ALTER TABLE Staff
      ALTER sex SET DEFAULT 'F';
```

Adding Primary and Foreign keys to existing Tables

```
ALTER TABLE myemp
ADD PRIMARY KEY (empno);

ALTER TABLE myemp
ADD FOREIGN KEY(deptno) REFERENCES mydept(deptno);
```

NOTE:

The primary key must be created first on mydept, before the foreign key can be created.

MORE ON CONSTRAINTS

Checking the Constraints

You can see a list of all the constraints set by querying the data dictionary table **user_constraints**:

```
SELECT constraint_name, constraint_type, table_name
FROM user_constraints;
```

You should see that the constraint has a name allocated to it in the format SYS_Cnnnn.

This name is **automatically assigned** if you do not name your constraints.

The constraint also has a type:

- P = primary key
- U = unique key
- R = foreign key
- C = check Not Null.

Information about Constraints

You may also have noticed that just using the data dictionary table USER_CONSTRAINTS does not show you which column the constraint applies to.

You need to use another table as well, USER_CONS_COLUMNS.

```
SELECT  SUBSTR(a.constraint_name,1,15)   AS name,
        SUBSTR(constraint_type,1,1)      AS type,
        SUBSTR(a.table_name,1,15)        AS table_name,
        SUBSTR(column_name,1,15)         AS column_name,
        SUBSTR(r_constraint_name,1,15)   AS references

FROM user_constraints a, user_cons_columns b
WHERE a.constraint_name = b.constraint_name
ORDER BY a.table_name;
```

Naming the Integrity Constraints

To simplify the handling of **constraints** you can **name them**.

Add the word CONSTRAINT and then **the name** you require and then follow it with **the specific constraint** you wish to implement:

A meaningful name for a constraint would be something like:

staff_pk or branch_fk1 or employee_chk2

Adding Constraints to existing Tables

```
ALTER TABLE myemp
ADD CONSTRAINT myemp_pk  PRIMARY KEY (empno);

ALTER TABLE myemp
ADD CONSTRAINT myemp_fk
    FOREIGN KEY(deptno) REFERENCES mydept(deptno);

ALTER TABLE myemp
ADD CONSTRAINT myemp_sal_check CHECK (sal < 200000);
```

NOTE:

The primary key must be created first on mydept, before the foreign key can be created.

Naming the Constraints

Alternatively, you can name out-of-line constraints when the table is created:

```
CREATE TABLE testemp2
(
  empno NUMBER(2)
  name CHAR(10),
  deptno NUMBER(2),
  salary NUMBER(7,2)
  ni_no CHAR(9),

CONSTRAINT sal_check      CHECK (salary < 100000),
CONSTRAINT testemp2_pk    PRIMARY KEY (empno),
CONSTRAINT testemp2_fk    FOREIGN KEY (deptno) REFERENCES mydept(deptno)

);
```

Naming the Constraints

or you can also name inline constraints, but this makes the script very difficult to read – not recommended:

```
CREATE TABLE testemp2
(
  empno NUMBER(2)     CONSTRAINT testemp2_pk    PRIMARY KEY,

  name CHAR(10),

  deptno NUMBER(2)    CONSTRAINT testemp2_fk    FOREIGN KEY (deptno) REFERENCES
                                                      mydept(deptno),

  salary NUMBER(7,2)  CONSTRAINT sal_check      CHECK (salary < 100000),

  ni_no CHAR(9)       CONSTRAINT ni_unique      UNIQUE

);
```

You can disable and then enable constraints should you need to, this is useful if performing large table inserts:

```
ALTER TABLE testemp2 DISABLE PRIMARY KEY;

ALTER TABLE testemp1
DISABLE CONSTRAINT testemp2_fk;

ALTER TABLE testemp2 ENABLE PRIMARY KEY;
```

You can also delete constraints:

ALTER TABLE testemp2 DROP PRIMARY KEY;

ALTER TABLE testemp2 DROP CONSTRAINT testemp2_fk1;

DROP TABLE

DROP TABLE

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

e.g. `DROP TABLE PropertyForRent;`

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

Exercise

- Use named constraints.

Creating a composite key

```
CREATE TABLE orderline
    (
        order_no NUMBER(2),
        product_no NUMBER(4),
        qty NUMBER (6),
        CONSTRAINT ord_line_pk PRIMARY KEY (order_no, product_no)
    );
```