Chapter 7

SQL – Data Definition

Chapter 7 - Objectives

- Data types supported by SQL standard.
- Purpose of integrity enhancement feature of SQL.
- How to define integrity constraints using SQL.
- How to use the integrity enhancement feature in the CREATE and ALTER TABLE statements.

Chapter 7 - Objectives

- Purpose of views.
- How to create and delete views using SQL.
- How the DBMS performs operations on views.
- Under what conditions views are updatable.
- Advantages and disadvantages of views.
- How the ISO transaction model works.
- How to use the GRANT and REVOKE statements as a level of security.

SQL Identifiers

- Used to identify objects in the database, such as table names, view names, and columns
- Consists of the uppercase letters A . . . Z, the lowercase letters a . . . z, the digits 0 . . . 9, and the underscore (_) character
- Can be no longer than 128 characters.
- Must start with a letter
- Cannot contain spaces

SQL Scalar Data Types

- **ISO SQL data types:**

DATA TYPE	DECLARATIONS				
boolean	BOOLEAN				
character	CHAR	VARCHAR			
bit†	BIT	BIT VARYING			
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT	BIGINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION		
datetime	DATE	TIME	TIMESTAMP		
interval	INTERVAL				
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT		

+ BIT and BIT VARYING have been removed from the SQL:2003 standard.

SQL Scalar Data Types

- Boolean data
  - consists of the truth values TRUE and FALSE
- Character data
  - branchNo CHAR(4) (fixed length of 4 chars)
  - address VARCHAR(30) (up to max. 30 chars)
- Bit data
  - Defining bit strings
  - Format similar to that of character data type
  - E.g. bitString BIT(4)

SQL Scalar Data Types

- Exact numeric data
  - Define numbers with an exact representation
  - Consists of a precision and a scale
  - E.g. rooms SMALLINT (-32,768 ~ 32767)
  - E.g. salary DECIMAL(7,2) (7 sig. digits, 2 dec. places)
- Approximate numeric data
  - Define numbers without an exact representation
  - Notation is similar to scientific notation
  - E.g. 10E3, +5.2E6, –0.2E–4

SQL Scalar Data Types

- Datetime data
  - Define points in time
  - DATE stores the YEAR, MONTH, and DAY fields
  - TIME stores the HOUR, MINUTE, and SECOND fields
  - TIMESTAMP is used to store date and times
  - E.g. viewDate DATE
- Interval data
  - Used to represent periods of time
  - Consists of a contiguous subset of the fields: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

SQL Scalar Data Types

- **Large objects**
  - **A data type that holds a large amount of data, such as a long text file or a graphics file. Three different kinds of large object types are defined in SQL:**
  - **Binary Large Object (BLOB) is a binary string that does not have a character set or collation association**
  - **Character Large Object (CLOB) and National Character Large Object (NCLOB), both character strings.**

Integrity Enhancement Feature

- Consists of constraints we wish to impose to protect the database from becoming inconsistent.
- Consider five types of integrity constraints:
  - required data
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints.

Integrity Enhancement Feature

Required Data

position VARCHAR(10) NOT NULL

(***every member of staff must have a job position***)

Domain Constraints

(a) CHECK

sex CHAR NOT NULL CHECK (sex IN ('M', 'F'))

(***The sex of a member of staff is either 'M' or 'F'***)

Integrity Enhancement Feature

(b) CREATE DOMAIN

CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]

For example:

```
CREATE DOMAIN SexType AS CHAR
        DEFAULT 'M'
        CHECK (VALUE IN ('M', 'F'));
sex     SexType       NOT NULL
```

Integrity Enhancement Feature

- *searchCondition* can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
                 FROM Branch));
```

- Domains can be removed using DROP DOMAIN:

```
DROP DOMAIN DomainName [RESTRICT | CASCADE]
```

RESTRICT: Refuse to drop the domain if there are any dependent objects. This is the default.

CASCADE: Automatically drop objects that depend on the domain (such as table columns).

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 alternate keys using UNIQUE:

  UNIQUE(telNo)

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 the FK contains a value, that value must refer to an existing, valid row in the parent table.
- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE statements:

FOREIGN KEY(branchNo) REFERENCES Branch

(*Define branchNo in PropertyForRent table*)

Referential Integrity

- Any INSERT/UPDATE attempting to create FK value in child table without matching PK value in parent is rejected.
- Action taken attempting to update/delete a PK 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

Referential Integrity

CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.

SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are not specified as NOT NULL.

SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.

NO ACTION: Reject delete from parent. Default.

Referential Integrity

FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL

(If a staff record is deleted from the* Staff *table, the values of the corresponding* staffNo *column in the* PropertyForRent *table are set to NULL.)

FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE

(*If an owner number is updated in the* PrivateOwner *table, the corresponding column(s) in the* PropertyForRent *table are set to the new value.*)

General Constraints

- Updates to tables may be constrained by enterprise rules governing the real-world transactions
- Can use CHECK and UNIQUE clauses in CREATE and ALTER TABLE statements
- Can also use:

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)
```

General Constraints

DreamHome may have a rule that prevents a member of staff from managing more than 100 properties at the same time.

```
CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS    (SELECT staffNo
                      FROM PropertyForRent
                      GROUP BY staffNo
                      HAVING COUNT(*) > 100))
```

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**

Data Definition

- Relations and other database objects exist in an *environment*.
- Each environment contains one or more *catalogs*, and each catalog consists of a set of schemas.
- A schema is a named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.

CREATE SCHEMA

CREATE SCHEMA Name
[AUTHORIZATION CreatorId]

- **For example:**

CREATE SCHEMA DreamHome AUTHORIZATION DZhu

DROP SCHEMA

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 the order defined previously. If any of these operations fail, DROP SCHEMA fails.**

CREATE TABLE

After creating the database structure, we can create the table structures:

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

CREATE TABLE

- Creates a table called TableName with one or more columns of the specified *dataType*.
- With NOT NULL, system rejects any attempt to insert a null in the 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.

Example 7.1 - CREATE TABLE

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
        CHECK (VALUE IN (SELECT ownerNo FROM PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
        CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN BranchNumber AS CHAR(4)
        CHECK (VALUE IN (SELECT branchNo FROM Branch));
CREATE DOMAIN PropertyNumber AS VARCHAR(5);
CREATE DOMAIN Street AS VARCHAR(25);
CREATE DOMAIN City AS VARCHAR(15);
CREATE DOMAIN Postcode AS VARCHAR(8);
CREATE DOMAIN PropertyType AS CHAR(1)
        CHECK(VALUE IN ('B', 'C', 'D', 'E', 'F', 'M', 'S'));
CREATE DOMAIN PropertyRooms AS SMALLINT;
        CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PropertyRent AS DECIMAL(6,2)
        CHECK(VALUE BETWEEN 0 AND 9999.99);
```

To be continued…

Example 7.1 - CREATE TABLE

```
CREATE TABLE PropertyForRent(
    propertyNo      PropertyNumber      NOT NULL,
    street          Street              NOT NULL,
    city            City                NOT NULL,
    postcode        PostCode,
    type            PropertyType        NOT NULL DEFAULT 'F',
    rooms           PropertyRooms       NOT NULL DEFAULT 4,
    rent            PropertyRent        NOT NULL DEFAULT 600,
    ownerNo         OwnerNumber         NOT NULL,
    staffNo         StaffNumber
                    CONSTRAINT StaffNotHandlingTooMuch
                      CHECK (NOT EXISTS (SELECT staffNo
                                         FROM PropertyForRent
                                         GROUP BY staffNo
                                         HAVING COUNT(*) > 100)),
```

To be continued …

Example 7.1 - CREATE TABLE

```
    branchNo        BranchNumber                    NOT NULL,
    PRIMARY KEY (propertyNo),
    FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL
                                        ON UPDATE CASCADE,
FOREIGN KEY (ownerNo) REFERENCES PrivateOwner ON DELETE NO
                                        ACTION ON UPDATE CASCADE,
FOREIGN KEY (branchNo) REFERENCES Branch ON DELETE NO
                                        ACTION ON UPDATE CASCADE);
```

ALTER TABLE

It changes the structure of a table once it has been created. It consists of six options to:

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

ALTER TABLE Statement Format

ALTER TABLE tableName

[ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]

[DEFAULT defaultOption] [CHECK (searchCondition)]]

[DROP [COLUMN] columnName [RESTRICT | CASCADE]]

[ADD [CONSTRAINT [constraintName]] tableConstraintDefinition]

[DROP CONSTRAINT constraintName [RESTRICT | CASCADE]]

[ALTER [COLUMN] SET DEFAULT defaultOption]

[ALTER [COLUMN] DROP DEFAULT]

Example 7.2(a) - ALTER TABLE

Change Staff table by removing default of 'Assistant' for position column and setting default for sex column to female ('F').

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

Example 7.2(b) - ALTER TABLE

Remove constraint from PropertyForRent that staff are not allowed to handle more than 100 properties at a time. Add new column to Client table.

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
    ADD prefNoRooms PRooms;
```

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).**

Views

View

Dynamic result of one or more relational operations operating on base relations to produce another relation.

- **Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.**

Views

- Contents of a view are defined as a query on one or more base relations. The DBMS stores the definition of the view in the database.
- There are two approaches to view referencing.
- With view resolution, any operation on view is automatically translated into the defined query operations on the base relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

SQL - CREATE VIEW

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
    AS subselect
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.
- If omitted, each column takes name of corresponding column in *subselect*.

SQL - CREATE VIEW

- List must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

Example 7.3 - Create Horizontal View

A horizontal view restricts a user's access to selected rows of one or more tables.

Create a view so that manager at branch B003 can only see details for staff working in his or her branch office.

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example 7.4 - Create Vertical View

A vertical view restricts a user's access to selected columns of one or more tables.

Create view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
    FROM Staff
    WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Example 7.5 - Grouped and Joined Views

Create a view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
 AS SELECT s.branchNo, s.staffNo, COUNT(*)
      FROM Staff s, PropertyForRent p
      WHERE s.staffNo = p.staffNo
      GROUP BY s.branchNo, s.staffNo;
```

Example 7.3 - Grouped and Joined Views

Data for view StaffPropCnt:

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

SQL - DROP VIEW

- **Statement Format:**

  DROP VIEW ViewName [RESTRICT | CASCADE]

- **Causes definition of view to be deleted from database.**
- **For example:**

  DROP VIEW Manager3Staff;

SQL - DROP VIEW

- With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With RESTRICT (default), if any other objects depend for their existence on continued existence of the view being dropped, command is rejected.

View Resolution

Count number of properties managed by each member at branch B003.

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

View Resolution

(a) View column names in SELECT list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

(b) View names in FROM are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```

View Resolution

(c) WHERE from user query is combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

(d) GROUP BY and HAVING clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

(e) ORDER BY copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```

View Resolution

(f) Final merged query is now executed to produce the result:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
                  branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Restrictions on Views

SQL imposes several restrictions on creation and use of views.

(a) If column in view is based on an aggregate function:

- Column may appear only in SELECT and ORDER BY clauses of queries that access view.
- Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.

Restrictions on Views

- For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

(b) Grouped view may never be joined with a base table or a view.

- For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

View Updatability

- All updates to a base table are immediately reflected in all views that encompass that base table.
- Similarly, may expect that if view is updated then base table(s) will reflect change.

View Updatability

- However, consider again view StaffPropCnt.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, we do not know which properties they are; i.e. do not know primary keys!

View Updatability

- If change definition of view and replace count with actual property numbers:

```
CREATE VIEW StaffPropList (branchNo,
                           staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo;
```

View Updatability

- Now try to insert the record:

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

- Still problem, because in PropertyForRent all columns except postcode/staffNo are not allowed nulls.
- However, have no way of giving remaining non-null columns values.

View Updatability

- ISO specifies that a view is updatable if and only if:

  - DISTINCT is not specified.

  - Every element in SELECT list of defining query is a column name and no column appears more than once.

  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.

  - WHERE clause doesn't include nested SELECTs referencing the table in the FROM clause.

  - No GROUP BY or HAVING clause in the defining query .

  - Also, every row added through view must not violate integrity constraints of base table.

WITH CHECK OPTION

- Rows exist in a view because they satisfy WHERE condition of defining query.
- If a row changes and no longer satisfies condition, it disappears from the view.
- New rows appear within view when insert/update on view cause them to satisfy WHERE condition.
- Rows that enter or leave a view are called *migrating rows*.
- WITH CHECK OPTION prohibits a row migrating out of the view.

WITH CHECK OPTION

- LOCAL/CASCADED apply to view hierarchies.
- With LOCAL, any row insert/update on this view and any view directly or indirectly defined on this view must not cause row to disappear from the view unless row also disappears from underlying derived view/table.
- With CASCADED (default), any row insert/ update on this view and on any view directly or indirectly defined on this view must not cause row to disappear from the view.

Example 7.6 - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = 'B003'
WITH CHECK OPTION;
```

- Cannot update branch number of a row from B003 to B002 as this would cause the row to migrate from view.
- Also cannot insert a row into view with a branch number that does not equal B003.

Example 7.6 - WITH CHECK OPTION

- Now consider the following:

```
CREATE VIEW LowSalary
 AS        SELECT * FROM Staff WHERE salary > 9000;

CREATE VIEW HighSalary
 AS        SELECT * FROM LowSalary
           WHERE salary > 10000
 WITH LOCAL CHECK OPTION;

CREATE VIEW Manager3Staff
 AS        SELECT * FROM HighSalary
           WHERE branchNo = 'B003';
```

Example 7.6 - WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.
- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

Example 7.6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary.
- To prevent anomalies like this, each view should be created using WITH CASCADED CHECK OPTION.

Advantages of Views

- Data independence
  - Even if the underlying source tables are changed, the view can present a consistent, unchanging structure of the database
- Currency
  - Changes to any of the base tables in the defining query are immediately reflected in the view.
- Improved security
  - Each user can access the database only through a small set of views that contain the data appropriate for that user

Advantages of Views

- **Reduced complexity**
  - A view can simplify queries by drawing data from several tables into a single table, transforming multi-table queries into single-table queries.
- **Convenience**
  - Users are presented with only the part of the database that they need to see.
- **Customization**
  - Views customize the appearance of the database so that the same underlying base tables can be seen by different users in different ways.

Advantages of Views

- Data integrity
  - If the WITH CHECK OPTION clause of the CREATE VIEW statement is used, then SQL ensures that no row that fails to satisfy the WHERE clause of the defining query is ever added to any of the underlying base table(s) through the view, thereby ensuring the integrity of the view.

Disadvantages of Views

- **Update restriction**
  - In some cases, a view cannot be updated.
- **Structure restriction**
  - The structure of a view is determined at the time of its creation. If columns are later added to the base table, then these columns will not appear in the view, unless the view is dropped and recreated.
- **Performance**
  - A view defined by a complex, multi-table query may take a long time to process, as the view resolution must join the tables together every time the view is accessed.

View Materialization

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- View materialization stores view as temporary table when view is first queried.
- Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.

View Maintenance

- It is the process of updating a materialized view in response to changes to the underlying data.
- View maintenance aims to apply only those changes necessary to keep view current.
- Consider following view:

```
CREATE VIEW StaffPropRent(staffNo)
AS SELECT DISTINCT staffNo
        FROM PropertyForRent
        WHERE branchNo = 'B003' AND rent > 400;
```

staffNo
SG37
SG14

View Materialization

- If insert row into PropertyForRent with rent ≤400 then view would be unchanged.
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view.
- If insert row for property PG54 at branch B003 with staffNo = SG37 and rent = 450, then no new row would need to be added to materialized view because SG37 already exists in materialized view.
- If delete property PG24, row should be deleted from materialized view.
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21).