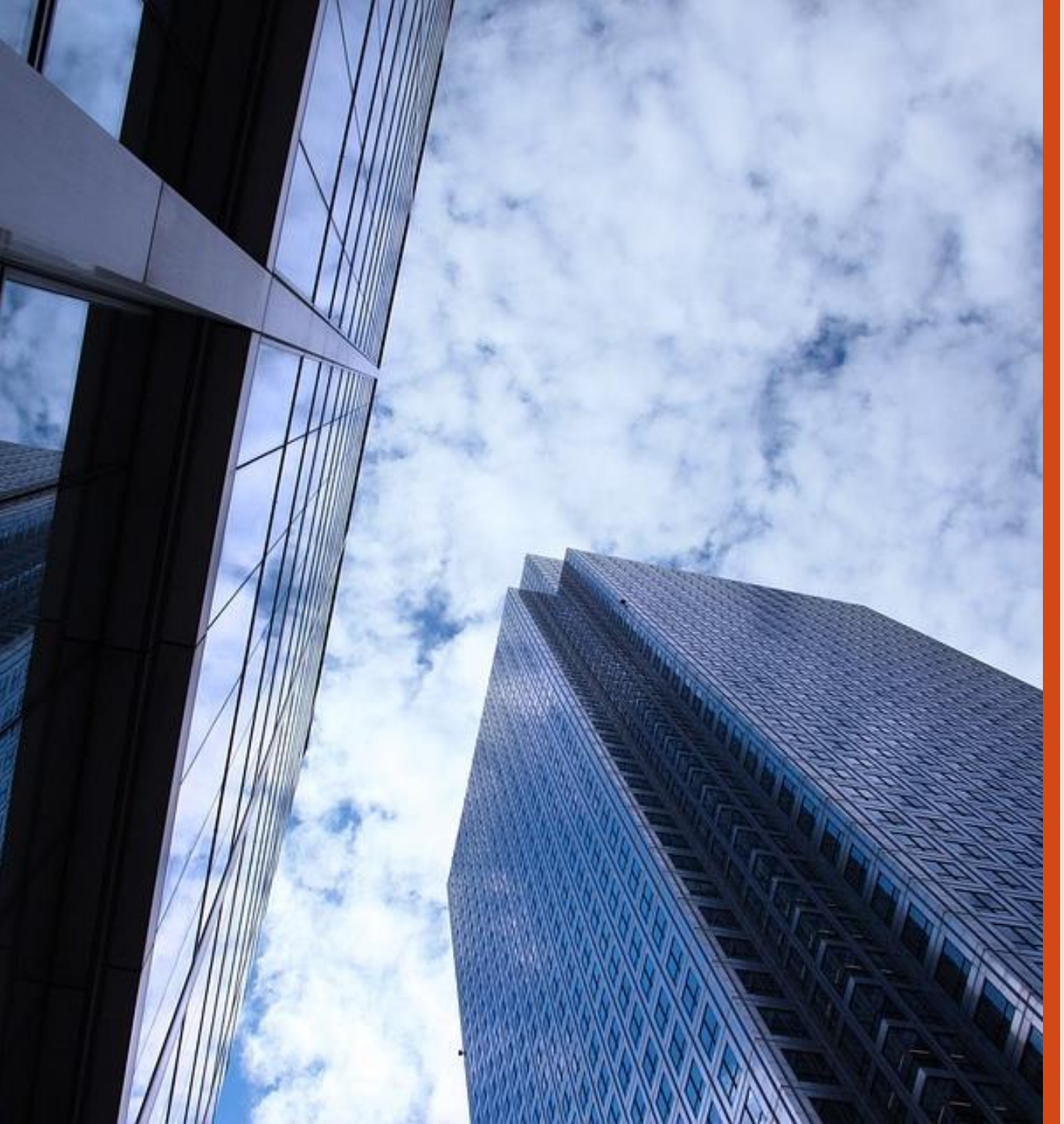

SQL
PART 02

Jayathma Chathurangani

ejc@ucsc.cmb.ac.lk

OUTLINE

- **ISO SQL Data Types**
- **ISO SQL Scalar Operators**
- **Integrity Control**
- **Data Definition Commands (DDL)**

1

ISO SQL DATA TYPES

1.1 SQL IDENTIFIERS

- SQL identifiers are used to **identify objects in the database, such as table names, view names, and columns**.
- The characters that can be used in a user-defined SQL identifier must appear in a character set.
- The ISO standard provides a default character set, which consists of the uppercase letters A . . . Z, the lowercase letters a . . . z, the digits 0 . . . 9, and the underscore (_) character.
- It is also possible to specify an alternative character set.
- Following Restrictions are imposed on identifiers
  - an identifier can be no longer than 128 characters (most dialects have a much lower limit than this);
  - an identifier must start with a letter;
  - an identifier cannot contain spaces.

1.2 ISO SQL DATA TYPES - BOOLEAN DATA

- Declaration: **BOOLEAN**
- **Boolean data consists of the distinct truth values TRUE** and **FALSE**.
- Unless prohibited by a NOT NULL constraint, boolean data also supports the **UNKNOWN** truth value as the NULL value. Null value cannot be hold.
- The value TRUE is greater than the value FALSE.
- Any comparison involving the NULL value or an UNKNOWN truth value returns an UNKNOWN result.
- When a Boolean column contains NULL, logical operations involving it follow three-valued logic (3VL):TRUE AND UNKNOWN → UNKNOWN
  - FALSE AND UNKNOWN → FALSE
  - TRUE OR UNKNOWN → TRUE
  - FALSE OR UNKNOWN → UNKNOWN
  - NOT UNKNOWN → UNKNOWN

1.3 ISO SQL DATA TYPES - CHARACTER DATA

- **Character data consists of a sequence of characters from an implementation defined character set**, that is, it is defined by the vendor of the particular SQL dialect.
- Thus, the exact characters that can appear as data values in a character type column will vary.
- ASCII and EBCDIC are two sets in common use today.
- When a character string column is defined, a length can be specified to indicate the maximum number of characters that the column can hold (default length is 1).
- A character string may be defined as having a fixed or varying length.
- If the string is defined to be a **fixed length** and we enter a string with fewer characters than this length, the string is padded with blanks on the right to make up the required size.
- If the string is defined to be of a **varying length** and we enter a string with fewer characters than this length, only those characters entered are stored, thereby using less space.

branchNo **CHAR**(4)

address **VARCHAR**(30)

1.4 ISO SQL DATA TYPES - BIT DATA

- The bit data type is used to define bit strings, that is, **a sequence of binary digits (bits), each having either the value 0 or 1**.
- The format for specifying the bit data type is similar to that of the character data type
- For example, to hold the fixed length binary string "0011", we declare a column *bitString*, as:

```
bitString BIT(4)
```

1.5 ISO SQL DATA TYPES - EXACT NUMERIC DATA

- The exact numeric data type is used to **define numbers with an exact representation**.
- The number consists of digits, an optional decimal point, and an optional sign.
- An exact numeric data type consists of a precision and a scale.
- The precision gives the total number of significant decimal digits, that is, the total number of digits, including decimal places but excluding the point itself. The scale gives the total number of decimal places.
- NUMERIC and DECIMAL store numbers in decimal notation. INTEGER is used for large positive or negative whole numbers. SMALLINT is used for small positive or negative whole numbers and BIGINT for very large whole numbers.

NUMERIC [precision [, scale]]
DECIMAL [precision [, scale]]
INTEGER
SMALLINT
BIGINT

INTEGER can be abbreviated to **INT** and **DECIMAL** to **DEC**

rooms **SMALLINT**

salary **DECIMAL**(7,2)

1.5 ISO SQL DATA TYPES - EXACT NUMERIC DATA (CONTD)

INTEGER (or INT)

•Used for whole numbers without a fractional part.

•Supports a range of values depending on the database implementation (typically 32-bit or 64-bit integers).

```
CREATE TABLE ExampleTable (EmployeeID INTEGER);
INSERT INTO ExampleTable (EmployeeID) VALUES (123);
```

SMALLINT

•A smaller version of INTEGER.

•Requires less storage and supports a smaller range of values (usually 16-bit integers).

```
CREATE TABLE ExampleTable (Age SMALLINT);
INSERT INTO ExampleTable (Age) VALUES (25);
```

BIGINT

•Used for very large integers.

•Supports a wider range of values (usually 64-bit integers).

```
CREATE TABLE ExampleTable (Population BIGINT);
INSERT INTO ExampleTable (Population) VALUES (9876543210);
```

1.5 ISO SQL DATA TYPES - EXACT NUMERIC DATA (CONTD)

NUMERIC(p, s)

- A fixed-precision number type.
- p specifies the total number of digits (precision).
- s specifies the number of digits after the decimal point (scale)

DECIMAL(p, s)

- Similar to NUMERIC, but implementations may allow slightly more flexibility in storage.

Example
CREATE TABLE ExampleTable (Salary NUMERIC(6, 2) -- Up to 6 digits, 2 after the decimal point);
12345.67 → **??**

1.6 ISO SQL DATA TYPES - APPROXIMATE NUMERIC DATA

- The approximate numeric data type is used for defining numbers that do not have an exact representation, such as real numbers.
- Approximate numeric, or floating point, notation is similar to scientific notation, in which a number is written as a mantissa times some power of ten (the exponent).
- For example, 10E3, +5.2E6, −0.2E–4.

FLOAT [precision]
REAL
DOUBLE PRECISION

1.6 ISO SQL DATA TYPES - APPROXIMATE NUMERIC DATA (CONTD)

FLOAT

•A general floating-point data type.
•Can store a wide range of values, with precision dependent on the implementation or specified precision.
•For MySQL, -3.4E+38 to 3.4E+38 (32-bit floating-point standard, varies with the implementation).

```
CREATE TABLE ExampleTable (    Measurement FLOAT(10) -- Approximate precision with 10 bits);
INSERT INTO ExampleTable (Measurement) VALUES (12345.678);
```

REAL (REAL is a synonym for FLOAT(24))

•A single-precision floating-point type.
•Typically, equivalent to FLOAT with a predefined precision (implementation-dependent, often 24 bits).
•Range approximately between -3.4 X 10^{38} to +3.4 X 10^{38} (-3.4E+38 to 3.4E+38)

```
CREATE TABLE ExampleTable (    Temperature REAL);
INSERT INTO ExampleTable (Temperature) VALUES (36.6);
```

DOUBLE PRECISION (It is equivalent to FLOAT(53) in most databases. For MySQL is Double)

•A double-precision floating-point type
•Provides more precision compared to REAL (implementation-dependent, often 53 bits).
•Range approximately between -1.79 X 10^{308} to +1.79 X 10^{308} (-1.8E+308 to 1.8E+308)

1.7 ISO SQL DATA TYPES - DATETIME DATA

- The datetime data type is **used to define points in time** to a certain degree of accuracy.
- Examples are dates, times, and times of day.
- **DATE** is used to store calendar dates using the YEAR, MONTH, and DAY fields.
- **TIME** is used to store time using the HOUR, MINUTE, and SECOND fields.
- **TIMESTAMP** is used to store date and times. The time Precision is the number of decimal places of accuracy to which the SECOND field is kept.
- If not specified, TIME defaults to a precision of 0 (that is, whole seconds), and TIMESTAMP defaults to 6 (that is, microseconds).
- The WITH TIME ZONE keyword controls the presence of the TIMEZONE_HOUR and TIMEZONE_MINUTE fields.

```
DATE
TIME [timePrecision] [WITH TIME ZONE]
TIMESTAMP [timePrecision] [WITH TIME ZONE]
```

```
viewDate DATE
```

Data Type	Range
date	0001-01-01 to 9999-12-31
time	00:00:00.0000000 to 23:59:59.9999999

1.8 ISO SQL DATA TYPES - INTERVAL DATA

- The interval data type **is used to represent periods of time**.
- MySQL does not have a built-in INTERVAL data type for storing time intervals or durations. But support supports INTERVAL in expressions to manipulate dates and times.
- Every interval data type consists of a contiguous subset of the fields: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.
- Two classes: The **year–month** class may contain only the YEAR and/ or the MONTH fields (YYYY-MM); the **day–time** class may contain only a contiguous selection from DAY, HOUR, MINUTE, SECOND (DD HH:MM:SS).
- In all cases, *startField* has a leading field precision that defaults to 2.
- For example:
  - **INTERVAL YEAR**(2) **TO MONTH**

represents an interval of time with a value between
0 years 0 months, and 99 years 11 months.
(This allows a range of 00 to 99 years for the YEAR compone

  - **INTERVAL HOUR TO SECOND**(4)

represents an interval of time with a value between
99 hours 59 minutes 59.9999 seconds (the fractional precision of second is 4).

INTERVAL {{startField **TO** endField} singleDatetimeField}
startField = **YEAR** | **MONTH** | **DAY** | **HOUR** | **MINUTE**
[(intervalLeadingFieldPrecision)]
endField = **YEAR** | **MONTH** | **DAY** | **HOUR** | **MINUTE** | **SECOND**
[(fractionalSecondsPrecision)]
singleDatetimeField = startField | **SECOND**
[(intervalLeadingFieldPrecision [, fractionalSecondsPrecision])]

*If the precision is **not explicitly specified**, the default behavior is determined by the database implementation, which is commonly **2 digits**.

1.9 ISO SQL DATA TYPES - LARGE OBJECTS

- A **large object** is a data type that holds a large amount of data, such as a long text file or a graphics file.
- Three different types of large object data types are defined in SQL:
  - **Binary Large Object (BLOB),** 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.

1.10 ISO SQL DATA TYPES - SUMMARY

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.

Above Table shows the SQL scalar data types defined in the ISO standard. Sometimes, for manipulation and conversion purposes, the data types *character* and *bit* are collectively referred to as **string** data types, and *exact numeric* and *approximate numeric* are referred to as **numeric** data types, as they share similar properties.

2

ISO SQL SCALAR OPERATORS

2.1 ISO SQL SCALAR OPERATORS - DEFINITION

- SQL provides a number of built-in scalar operators and functions that can be used to construct a scalar expression, that is, an expression that evaluates to a scalar value.
- Apart from the obvious arithmetic operators (+, −, *, /), some other operators are shown next.

2.2 ISO SQL SCALAR OPERATORS

OPERATOR	MEANING
OCTET_LENGTH	Returns the length of a string in octets (bit length divided by 8). For example, **OCTET_LENGTH**(×'FFFF') returns 2.
CHAR_LENGTH	Returns the length of a string in characters (or octets, if the string is a bit string). For example, **CHAR_LENGTH** ('Beech') returns 5.
CAST	Converts a value expression of one data type into a value in another data type. For example, **CAST**(5.2E6 AS INTEGER).
||	Concatenates two character strings or bit strings. For example, fName || lName.
CURRENT_USER or **USER**	Returns a character string representing the current authorization identifier (informally, the current user name).
SESSION_USER	Returns a character string representing the SQL-session authorization identifier.
SYSTEM_USER	Returns a character string representing the identifier of the user who invoked the current module.
LOWER	Converts uppercase letters to lowercase. For example, **LOWER**(**SELECT** fName **FROM** Staff **WHERE** staffNo = 'SL21') returns 'john'.
UPPER	Converts lower-case letters to upper-case. For example, **UPPER**(**SELECT** fName **FROM** Staff **WHERE** staffNo = 'SL21') returns 'JOHN'.
TRIM	Removes leading (**LEADING**), trailing (**TRAILING**), or both leading and trailing (**BOTH**) characters from a string. For example, **TRIM**(**BOTH** '*' **FROM** '*** Hello World ***') returns 'Hello World'.
POSITION	Returns the position of one string within another string. For example, **POSITION** ('ee' **IN** 'Beech') returns 2.
SUBSTRING	Returns a substring selected from within a string. For example, **SUBSTRING** ('Beech' **FROM** 1 **TO** 3) returns the string 'Bee'.

2.2 ISO SQL SCALAR OPERATORS - CONTINUED

CASE	Returns one of a specified set of values, based on some condition. For example, **CASE** type **WHEN** 'House' **THEN** 1 **WHEN** 'Flat' **THEN** 2 **ELSE** 0 **END**
CURRENT_DATE	Returns the current date in the time zone that is local to the user.
CURRENT_TIME	Returns the current time in the time zone that is the current default for the session. For example, **CURRENT_TIME**(6) gives time to microseconds precision.
CURRENT_ TIMESTAMP	Returns the current date and time in the time zone that is the current default for the session. For example, **CURRENT_TIMESTAMP**(0) gives time to seconds precision.
EXTRACT	Returns the value of a specified field from a datetime or interval value. For example, **EXTRACT**(**YEAR FROM** Registration.dateJoined).
ABS	Operates on a numeric argument and returns its absolute value in the same most specific type. For example, **ABS**(−17.1) returns 17.1.
MOD	Operates on two exact numeric arguments with scale 0 and returns the modulus (remainder) of the first argument divided by the second argument as an exact numeric with scale 0. For example, **MOD**(26, 11) returns 4.
LN	Computes the natural logarithm of its argument. For example, **LN**(65) returns 4.174 (approx).
EXP	Computes the exponential function, that is, *e*, (the base of natural logarithms) raised to the power equal to its argument. For example, **EXP**(2) returns 7.389 (approx).

2.2 ISO SQL SCALAR OPERATORS - CONTINUED

POWER	Raises its first argument to the power of its second argument. For example, **POWER**(2,3) returns 8.
SQRT	Computes the square root of its argument. For example, **SQRT**(16) returns 4.
FLOOR	Computes the greatest integer less than or equal to its argument. For example, **FLOOR**(15.7) returns 15.
CEIL	Computes the least integer greater than or equal to its argument. For example, **CEIL**(15.7) returns 16.

3

INTEGRITY CONTROL

Consists of constraints that we wish to impose in order to protect the database from becoming inconsistent which can be defined in the **CREATE** and **ALTER TABLE** statements

3.1 REQUIRED DATA

- Some columns must contain a valid value; they are not allowed to contain nulls.
- A null is distinct from blank or zero, and is used to represent data that is either not available, missing, or not applicable.
- The ISO standard provides the NOT NULL column specifier in the CREATE and ALTER TABLE statements to provide this type of constraint.
- When NOT NULL is specified, the system rejects any attempt to insert a null in the column. If NULL is specified, the system accepts nulls.
- The ISO default is NULL.
- Example:

  position **VARCHAR(10) NOT NULL**

3.2 DOMAIN CONSTRAINTS

- Every column has a domain; in other words, a set of legal values.
- Three types are there.
- The ISO standard provides two mechanisms for specifying domains in the CREATE and ALTER TABLE statements.

Type 01:

- **CHECK** clause, which allows a constraint to be defined on a column or the entire table.

  CHECK (searchCondition**)**

- In a column constraint, the CHECK clause can reference only the column being defined.
- Thus, to ensure that the column sex can be specified only as ‘M’ or ‘F’, we could define the column as:

  sex **CHAR NOT NULL CHECK (sex IN (‘**M**’, ‘**F**’))**

3.2 DOMAIN CONSTRAINTS (CONTINUED)

Type 02 (Preferred Way) :

- ISO standard allows domains to be defined more explicitly using the **CREATE DOMAIN** statement:
  CREATE DOMAIN DomainName **[AS] dataType**
  [DEFAULT defaultOption**]**
  [CHECK (searchCondition**)]**
- A domain is given a name, DomainName, a data type, an optional default value, and an optional CHECK constraint. This is not the complete definition, but it is sufficient to demonstrate the basic concept.
- Thus, for the previous example, we could define a domain for sex as:
  CREATE DOMAIN SexType **AS CHAR**
  DEFAULT ‘M’
  CHECK (VALUE IN (‘M’, ‘F’**));**
- This definition creates a domain SexType that consists of a single character with either the value ‘M’ or ‘F’. When defining the column sex, we can now use the domain name SexType in place of the data type CHAR:
  sex SexType **NOT NULL**

3.2 DOMAIN CONSTRAINTS (CONTINUED)

Type 02 (Preferred Way) - Continued:

- Domains can be removed from the database using the **DROP DOMAIN** statement:

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

- The drop behavior, RESTRICT or CASCADE, specifies the action to be taken if the domain is currently being used.
- If **RESTRICT** is specified and the domain is used in an existing table, view, or assertion definition, the drop will fail.
- In the case of **CASCADE**, any table column that is based on the domain is *automatically changed to use the domain's underlying data type*, and any constraint or default clause for the domain is replaced by a column constraint or column default clause, if appropriate.

3.2 DOMAIN CONSTRAINTS (CONTINUED)

Type 03:

- The searchCondition can involve a table lookup.
- For example, we can create a domain BranchNumber to ensure that the values entered correspond to an existing branch number in the Branch table, using the statement:

  CREATE DOMAIN BranchNumber **AS CHAR(4)**
  CHECK (VALUE IN (SELECT branchNo **FROM** Branch**));**

3.3 ENTITY INTEGRITY

- The primary key of a table must contain a unique, nonnull value for each row
- The ISO standard supports entity integrity with the **PRIMARY KEY** clause in the CREATE and ALTER TABLE statements.

  PRIMARY KEY(propertyNo**)**
- To define a composite primary key,

  PRIMARY KEY(clientNo, propertyNo**)**
- The PRIMARY KEY clause can be specified only once per table. However, it is still possible to ensure uniqueness for any alternate keys in the table using the keyword **UNIQUE**. *Every column that appears in a UNIQUE clause must also be declared as NOT NULL*. There may be as many UNIQUE clauses per table as required.

  clientNo **VARCHAR(5) NOT NULL,**
  propertyNo **VARCHAR(5) NOT NULL,**
  UNIQUE (clientNo, propertyNo**)**
- SQL rejects any INSERT or UPDATE operation that attempts to create a duplicate value within each candidate key (that is, primary key or alternate key).

3.4 REFERENTIAL INTEGRITY

- A foreign key is a column, or set of columns, that links each row in the child table containing the foreign key to the row of the parent table containing the matching candidate key value.
- Referential integrity means that, if the foreign key contains a value, that value must refer to an existing, valid row in the parent table
- The ISO standard supports the definition of foreign keys with the **FOREIGN KEY** clause in the CREATE and ALTER TABLE statements.
- SQL rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table without a matching candidate key value in the parent table.
- The action SQL takes for any UPDATE or DELETE operation that attempts to update or delete a candidate key value in the parent table that has some matching rows in the child table is dependent on the referential action specified using the **ON UPDATE** and **ON DELETE** subclauses of the FOREIGN KEY clause.

3.4 REFERENTIAL INTEGRITY (CONTINUED)

- When the user attempts to delete a row from a parent table, and there are one or more matching rows in the child table, SQL supports four options regarding the action to be taken:
  1. **CASCADE**: Delete the row from the parent table and automatically delete the matching rows in the child table.
  2. **SET NULL:** Delete the row from the parent table and set the foreign key value(s) in the child table to NULL. This option is valid only if the foreign key columns do not have the NOT NULL qualifier specified.
  3. **SET DEFAULT**: Delete the row from the parent table and set each component of the foreign key in the child table to the specified default value. This option is valid only if the foreign key columns have a DEFAULT value specified.
  4. **NO ACTION**: Reject the delete operation from the parent table. This is the default setting if the ON DELETE rule is omitted.

3.4 REFERENTIAL INTEGRITY (CONTINUED)

- SQL supports the same options when the candidate key in the parent table is updated.
- With CASCADE, the foreign key value(s) in the child table are set to the new value(s) of the candidate key in the parent table. In the same way, the updates cascade if the updated column(s) in the child table reference foreign keys in another table.
- Example:

FOREIGN KEY (staffNo**) REFERENCES** Staff **ON DELETE SET NULL**

FOREIGN KEY (ownerNo**) REFERENCES** PrivateOwner **ON UPDATE CASCADE**

3.5 GENERAL CONSTRAINTS

- Updates to tables may be constrained by enterprise rules governing the real-world transactions that are represented by the updates.
- The ISO standard allows general constraints to be specified using the CHECK and UNIQUE clauses (discussed earlier) of the CREATE and ALTER TABLE statements and the **CREATE ASSERTION** statement.
- The CREATE ASSERTION statement is an integrity constraint that is not directly linked with a table definition.

  CREATE ASSERTION AssertionName
  CHECK (searchCondition**)**

- This statement is very similar to the CHECK clause discussed earlier.
- The CHECK clause in an assertion specifies the condition that the database must always ensure is true globally across the database. This means the database evaluates the CHECK condition whenever any relevant change occurs in the tables referenced by the assertion.

3.5 GENERAL CONSTRAINTS (CONTINUED)

- However, when a general constraint involves more than one table, it may be preferable to use an ASSERTION rather than duplicate the check in each table or place the constraint in an arbitrary table.
- For example, to define the general constraint that prevents a member of staff from managing more than 100 properties at the same time, we could write:

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

***Assertions are not widely supported:*
Many database systems (e.g., MySQL, PostgreSQL) do not support CREATE ASSERTION.
In such cases, you would need to enforce this rule using triggers or application logic.

4

DATA DEFINITION (DDL)

4.1 INTRODUCTION

- The SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- In this section, we briefly examine how to create and destroy schemas, tables, and indexes. We discuss how to create and destroy views in a latter section.
- Extra Reading: The ISO standard also allows the creation of character sets, collations, and translations. Refer Cannan and Otten, 1993.
- The main SQL data definition language statements are:

CREATE SCHEMA, DROP SCHEMA, CREATE DOMAIN, ALTER DOMAIN, DROP DOMAIN, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE VIEW DROP VIEW

- These statements are used to create, change, and destroy the structures that make up the conceptual schema.
- Although not covered by the SQL standard, the following two statements are provided by many DBMSs:

CREATE INDEX, DROP INDEX

4.2 CREATING A DATABASE – CREATE SCHEMA

- The process of creating a database differs significantly from product to product.
  - In multi-user systems, the authority to create a database is usually reserved for the DBA.
  - In a single-user system, a default database may be established when the system is installed and configured and others can be created by the user as and when required.
- The ISO standard does not specify how databases are created, and each dialect generally has a different approach.
- According to the ISO standard, relations and other database objects exist in an environment.
- Among other things, each environment consists of one or more **catalogs**, and **each catalog consists of a set of schemas**.
- A **schema** is a named collection of database objects that are in some way related to one another (all the objects in the database are described in one schema or another).
- The **objects** in a schema can be tables, views, domains, assertions, collations, translations, and character sets.
- All the objects in a schema have the same owner and share a number of defaults.

4.2 CREATING A DATABASE (CONTINUED)

- The schema definition statement has the following (simplified) form:

  CREATE SCHEMA [Name **| AUTHORIZATION** CreatorIdentifier**]**

- Therefore, if the creator of a schema SqlTests is Smith, the SQL statement is:

  CREATE SCHEMA SqlTests **AUTHORIZATION** Smith**;**

4.3 DROPPING A DATABASE – DROP SCHEMA

- Schema can be destroyed using the DROP SCHEMA statement, which has the following form:

  DROP SCHEMA Name **[RESTRICT | CASCADE]**

- If **RESTRICT** is specified, which is the default if neither qualifier is specified, the schema must be empty or the operation fails.
- If **CASCADE (Be Careful!)** is specified, the operation cascades to drop all objects associated with the schema in the order defined previously. If any of these drop operations fail, the DROP SCHEMA fails.
- It should be noted, however, that the CREATE and DROP SCHEMA statements are not always supported.

4.4 CREATING A TABLE – CREATE TABLE

- Having created the database structure, we may now create the table structures for the base relations to be stored in the database

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

4.4 CREATING A TABLE (CONTINUED)

- **Example**

```
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);
```

```
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)),
    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);
```

4.5 CHANGING A TABLE DEFINITION – ALTER TABLE

- The ISO standard provides an **ALTER TABLE** statement for changing the structure of a table once it has been created.
  - 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.

4.5 CHANGING A TABLE DEFINITION (CONTINUED)

- The basic format of the statement is:

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]

4.5 CHANGING A TABLE DEFINITION (CONTINUED)

- The **DROP COLUMN** clause specifies the name of the column to be dropped from the table definition, and has an optional qualifier that specifies whether the DROP action is to cascade or not:
  - **RESTRICT**: The DROP operation is rejected if the column is referenced by another database object (for example, by a view definition). This is the default setting.
  - **CASCADE**: The DROP operation proceeds and automatically drops the column from any database objects it is referenced by. This operation cascades, so that if a column is dropped from a referencing object, SQL checks whether that column is referenced by any other object and drops it from there if it is, and so on.

- Example: *For Default values and Domain Constraints*

```
ALTER TABLE Staff
        ALTER position DROP DEFAULT;
ALTER TABLE Staff
        ALTER sex SET DEFAULT 'F';
ALTER TABLE PropertyForRent
        DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
        ADD prefNoRooms PropertyRooms;
```

4.6 DROPPING A TABLE – DROP TABLE

- Over time, the structure of a database will change; new tables will be created and some tables will no longer be needed. We can remove a redundant table from the database using the DROP TABLE statement, which has the format:

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

- **RESTRICT**: The DROP operation is rejected if there are any other objects that depend for their existence upon the continued existence of the table to be dropped.
- **CASCADE**: The DROP operation proceeds and SQL automatically drops all dependent objects (and objects dependent on these objects).
- For example, to remove the PropertyForRent table we use the command:

DROP TABLE PropertyForRent**;**

- Note, however, that this command removes not only the named table, but also all the rows within it. To simply remove the rows from the table but retain the table structure, use the DELETE statement instead.

4.7 TRUNCATING A TABLE – TRUNCATE TABLE

- **TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure**.
- In DML commands, we will study about the DELETE command which is also more or less same as the TRUNCATE command.

TRUNCATE TABLE table_name

- DELETE command is slower than TRUNCATE command.
- The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.
- TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.

4.8 RENAMING A TABLE – RENAME TABLE

- RENAME command is used to set a new name for any existing table.

RENAME TABLE old_table_name **to** new_table_name

4.8 COMMENTING

- Comments in SQL are similar to comments in other programming languages such as Java, C++, Python, etc.
- They are primarily used to define a code section for easier understanding.
- Comments can be
  - Single line,

```
--Select all:
SELECT * FROM Customers;
```

  - Inline and

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

  - Multi-line

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

WRAP UP

THANK YOU!