DATA 30 (Section B)

Aye Hninn Khine, Ph.D.,
Visiting Faculty
Parami University

PARAMI UNIVERSITY

SQL identifiers

- SQL identifiers are used to identify objects in the database, such as table names, view names, and columns.
- 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.
- Example, table name “Property for Rent” is not valid since it contains spaces.

SQL Data Type

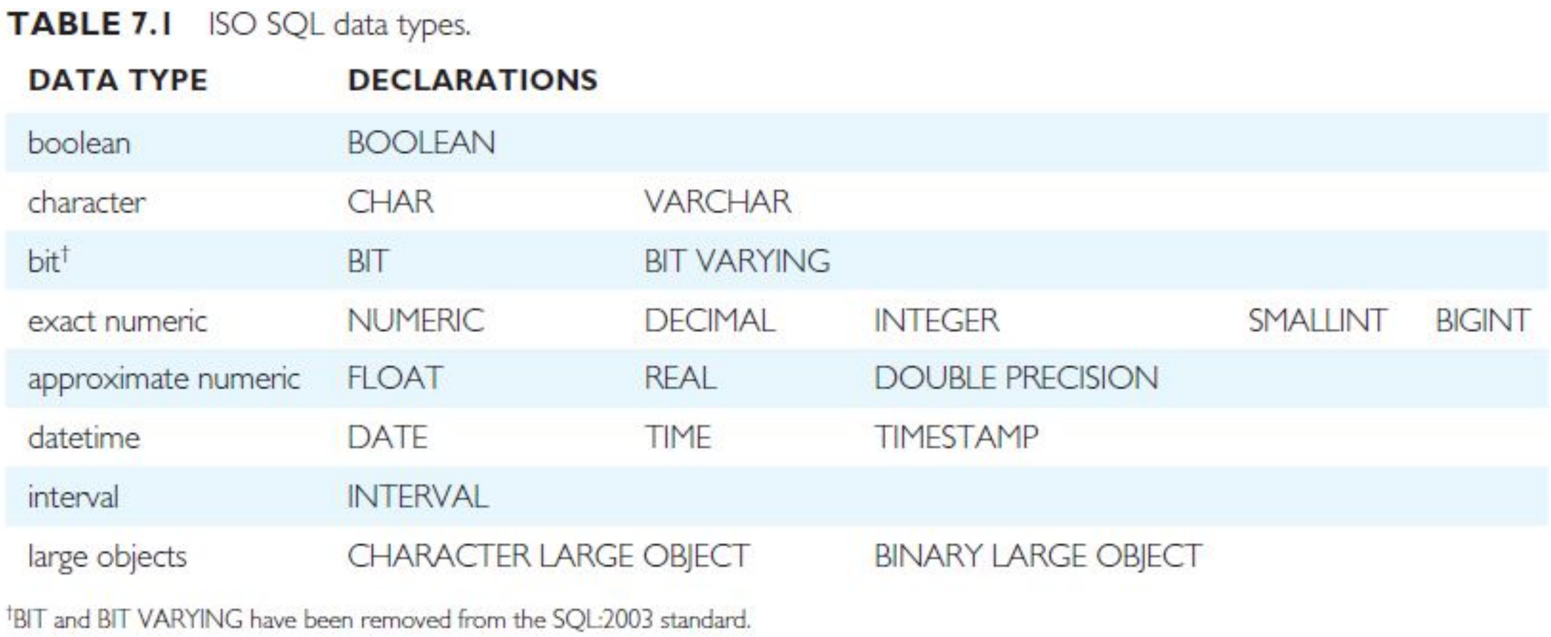


TABLE 7.1 ISO SQL data types.

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.

TO DO

Explore each data type to understand more details.

NUMERIC TYPES: SMALLINT < INT < BIGINT
DECIMAL/NUMERIC (exact), REAL/DOUBLE (approx)
TEXT TYPES: CHAR (fixed) | VARCHAR (variable) | TEXT (no limit)
DATE/TIME TYPES: DATE | TIME | TIMESTAMP | TIMESTAMPTZ
BOOLEAN: TRUE/FALSE
OTHER: BYTEA | UUID | JSONB | ARRAY | ENUM

SQL (Sub Query Language)

- SQL is the most widely used database query language.
- It is designed for retrieving and managing data in a relational database.
- SQL can be used to perform different types of operations in the database such as accessing data, describing data, manipulating data and setting users roles and privileges (permissions).

SQL Languages

Data Definition Language (DDL) - The SQL DDL category provides commands for defining, deleting and modifying tables in a database. Use the following commands in this category. (CREATE, ALTER, DELETE)

Data Query Language (DQL) The SQL DQL commands provide the ability to query and retrieve data from the database. Use the following command in this category. (SELECT Command)

Data Manipulation Language (DML) The SQL DML commands provide the ability to query, delete and update data in the database. Use the following commands in this category. (UPDATE)

Data Control Language (DCL) You use DCL to deal with the rights and permissions of users of a database system. You can execute SQL commands to perform different types of operations such as create and drop tables. To do this, you need to have user rights set up. This is called user privileges. This category deals with advanced functions or operations in the database. Note that this category can have a generic description of the two main commands. Use the following commands in this category, (GRANT)

SQL Operations

C - CREATE

R - READ

U - UPDATE

D - DELETE

Data Definition Language (DDL)

CREATE Command

Purpose: To create the database or tables inside the database

DROP Command

Purpose: To delete a database or a table inside the database.

ALTER Command

Purpose: To change the structure of the tables in the database such as changing the name of a table, adding a primary key to a table, or adding or deleting a column in a table.

TRUNCATE Command

Purpose: To remove all records from a table, which will empty the table but not delete the table itself.

COMMENT Command

Purpose: To add comments to explain or document SQL statements by using double dash (`--`) at the start of the line. Any text after the double dash will not be executed as part of the SQL statement. These comments are not there to build the database. They are only for your own use.

Simple CRUD Commands

```
-- Create a new table
CREATE TABLE Students (
    student_id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT,
    major VARCHAR(50)
);

-- Insert new records
INSERT INTO Students (name, age, major)
VALUES ('Alice', 20, 'Computer Engineering'),
       ('Bob', 22, 'Electrical Engineering');
```

```
-- Select all rows
SELECT * FROM Students;

-- Select specific columns
SELECT name, major FROM Students;

-- Select with condition
SELECT * FROM Students
WHERE age > 20;
```

Simple CRUD Commands

```
-- Update specific student's major
UPDATE Students
SET major = 'Mechanical Engineering'
WHERE name = 'Bob';

-- Update multiple fields
UPDATE Students
SET age = 21, major = 'Software Engineering'
WHERE student_id = 1;
```

```
-- Delete a single record
DELETE FROM Students
WHERE name = 'Alice';

-- Delete all rows (be careful!)
DELETE FROM Students;
```

Data Definition Language (DDL)

```
CREATE TABLE table_name (column_name1
datatype(size), column_name2 datatype(size),
column_name3 datatype(size));
```

```
DROP TABLE table_name;
```

```
ALTER TABLE table_name ADD (column_name
datatype(size));  – add new column
```

```
ALTER TABLE table_name ADD primary key
(column_name);
```

```
TRUNCATE TABLE table_name;
```

```
--Retrieve all data from a table
SELECT * FROM table_name;
```

Integrity Enhancement in DDL

- required data (NULL/NOT NULL)
- domain constraints
- entity integrity
- referential integrity
- general constraints

Required Data

- Some columns must contain a valid value; they are not allowed to contain nulls.
- For example, every member of staff must have an associated job position

- position VARCHAR(10) NOT NULL

Domain Constraints

- Every column has a domain; in other words, a set of legal.
- For example, the sex of a member of staff is either ‘M’ or ‘F’, so the domain of the column sex of the Staff table is a single character string consisting of either ‘M’ or ‘F
- sex **CHAR NOT NULL CHECK** (sex **IN** (‘M’, ‘F’)) – Column Constraint
- **CREATE DOMAIN** DomainName [**AS**] datatype **[DEFAULT** defaultOption] **[CHECK** (searchCondition)]
- **CREATE DOMAIN** SexType **AS CHAR DEFAULT** ‘M’ **CHECK (VALUE IN** (‘M’, ‘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**

Domain Constraints

- 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));
- Domains can be removed from the database using the DROP DOMAIN statement:
- **DROP DOMAIN** DomainName **[RESTRICT | CASCADE]**

Entity Constraints

- The primary key of a table must contain a unique, nonnull value for each.
- For example, each row of the PropertyForRent table has a unique value for the property number propertyNo, which uniquely identifies the property represented by that row

- **PRIMARY KEY**(propertyNo)

- To define a **composite primary key**, we specify multiple column names in the PRIMARY KEY clause, separating each by a comma.
- For example, to define the primary key of the Viewing table, which consists of the columns clientNo and propertyNo, we include the clause:

- **PRIMARY KEY**(clientNo, propertyNo)

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 standard supports definition of FKs with **FOREIGN KEY** clause in **CREATE** and **ALTER TABLE**
- FOREIGN KEY(branchNo) REFERENCES Branch
- 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

- For example, in the PropertyForRent table the staff number staffNo is a foreign key referencing the Staff table.

- We can specify a deletion rule such that 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** (staffNo) **REFERENCES** Staff **ON DELETE SET NULL**

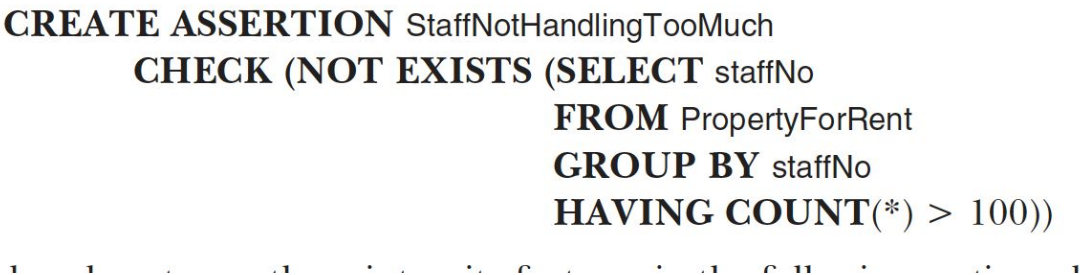
General Constraints

- Updates to tables may be constrained by enterprise rules governing the real-world transactions that are represented by the.
- For example, DreamHome may have a rule that prevents a member of staff from managing more than 100 properties at the same time

Format

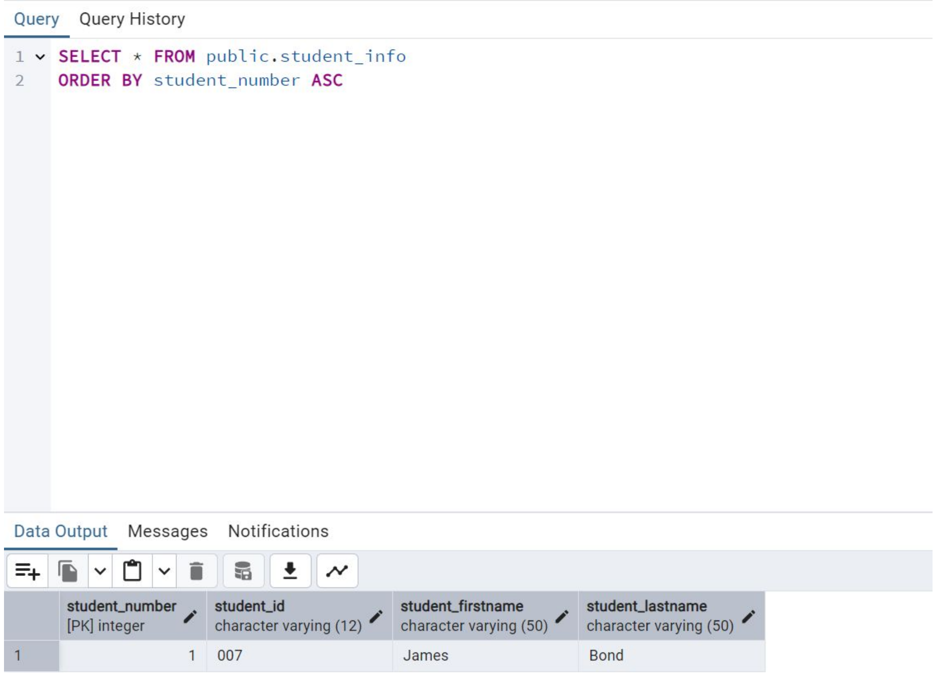
- *CREATE ASSERTION AssertionName CHECK (searchCondition)*

General constraint 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))
```

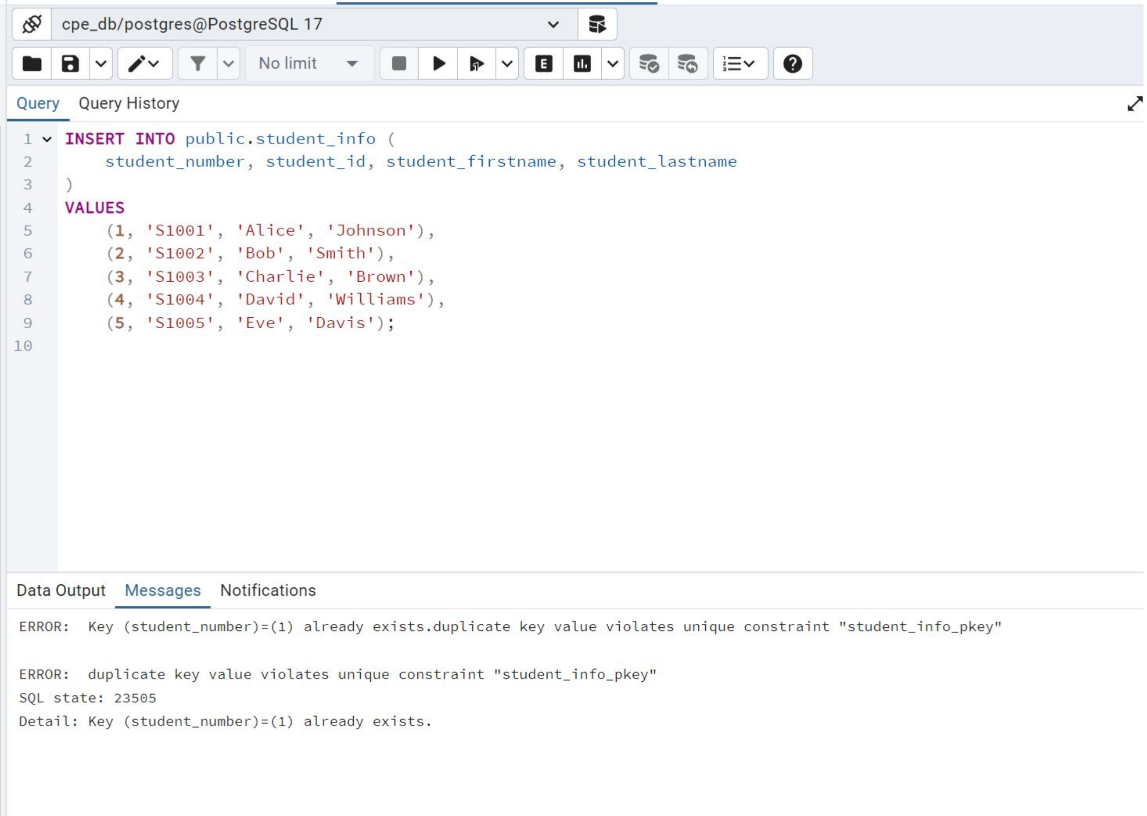
Unique Constraints



```
SELECT * FROM public.student_info
ORDER BY student_number ASC
```

	student_number [PK] integer	student_id character varying (12)	student_firstname character varying (50)	student_lastname character varying (50)
1	1	007	James	Bond

Unique Constraints

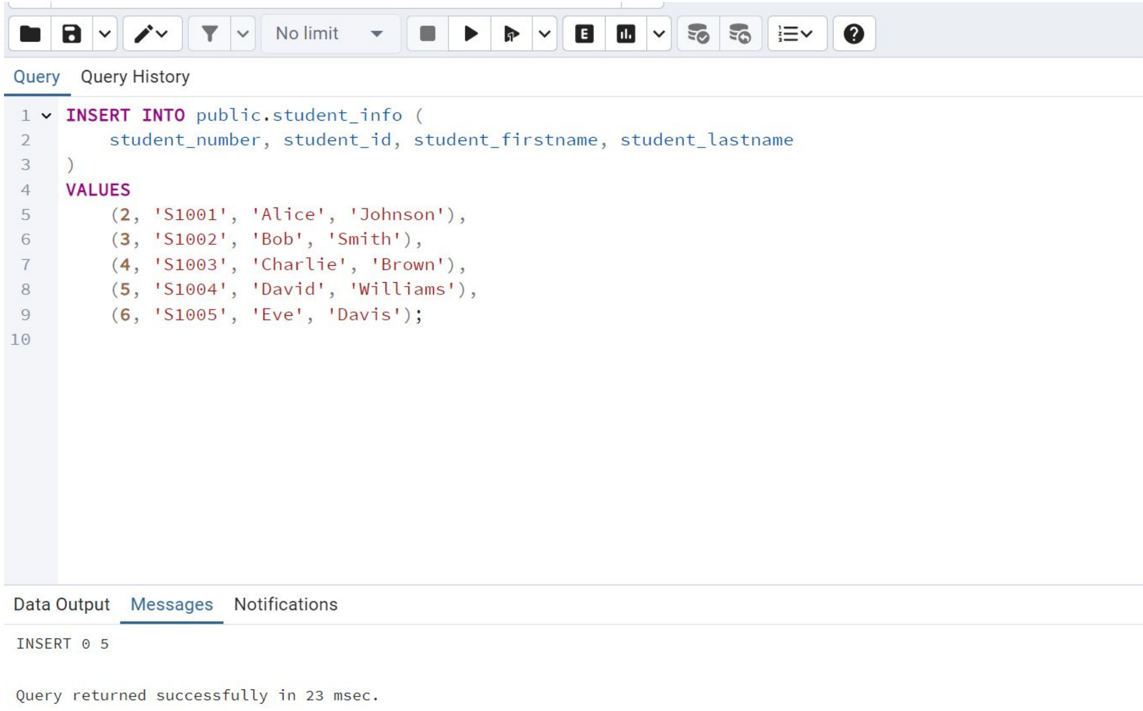


```
INSERT INTO public.student_info (
    student_number, student_id, student_firstname, student_lastname
)
VALUES
    (1, 'S1001', 'Alice', 'Johnson'),
    (2, 'S1002', 'Bob', 'Smith'),
    (3, 'S1003', 'Charlie', 'Brown'),
    (4, 'S1004', 'David', 'Williams'),
    (5, 'S1005', 'Eve', 'Davis');
```

```
ERROR:  Key (student_number)=(1) already exists.duplicate key value violates unique constraint "student_info_pkey"

ERROR:  duplicate key value violates unique constraint "student_info_pkey"
SQL state: 23505
Detail: Key (student_number)=(1) already exists.
```

Unique Constraints



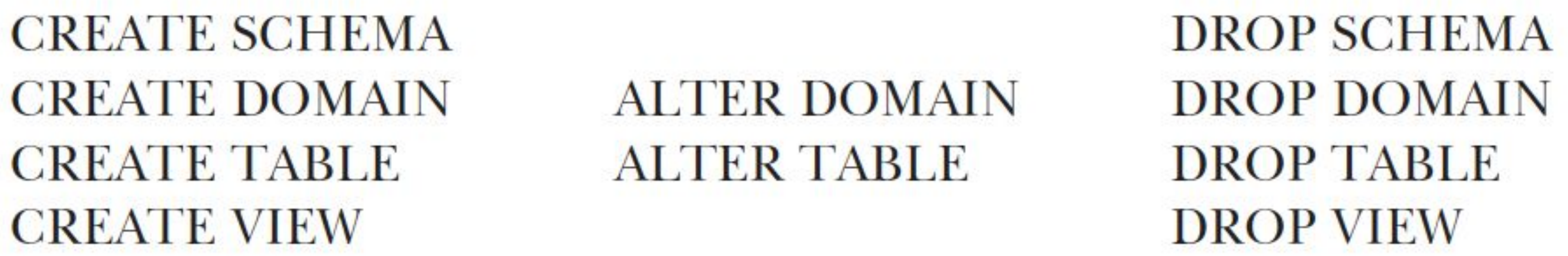
```
INSERT INTO public.student_info (
    student_number, student_id, student_firstname, student_lastname
)
VALUES
    (2, 'S1001', 'Alice', 'Johnson'),
    (3, 'S1002', 'Bob', 'Smith'),
    (4, 'S1003', 'Charlie', 'Brown'),
    (5, 'S1004', 'David', 'Williams'),
    (6, 'S1005', 'Eve', 'Davis');
```

```
INSERT 0 5

Query returned successfully in 23 msec.
```

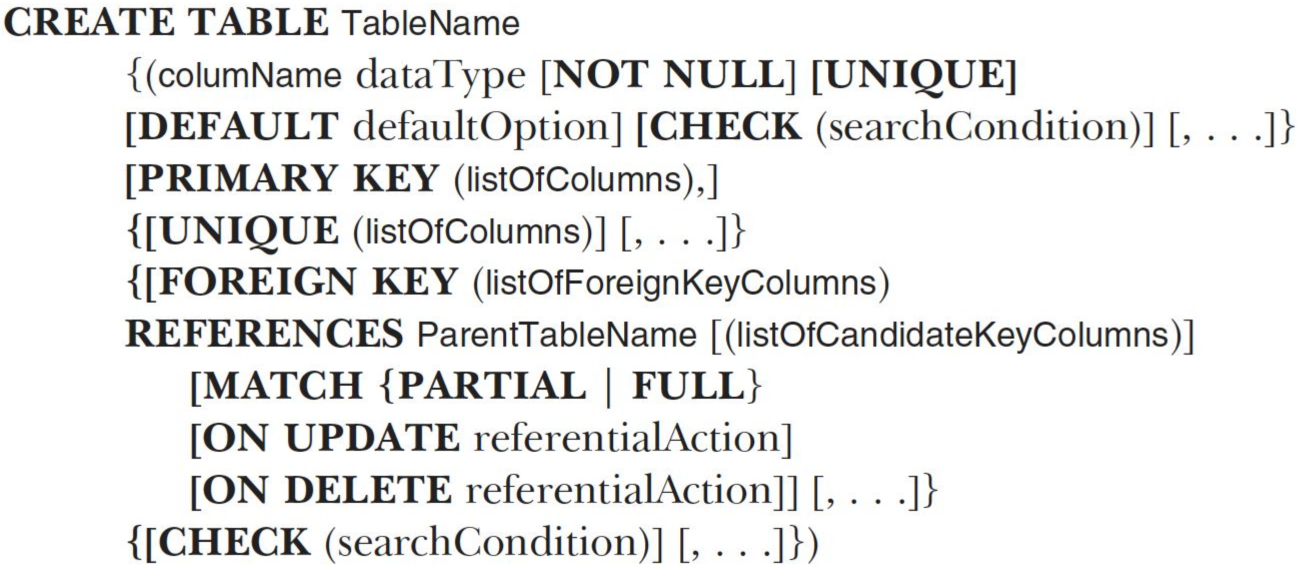
Data Definition Language

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



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

CREATE TABLE



```
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)] [, . . .]})
```

CREATE TABLE

- Creates a table 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

Changing a table definition (ALTER TABLE)

- 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

- Change the Staff table by removing the default of ‘Assistant’ for the position column and setting the default for the sex column to female (‘F’).
- ALTER TABLE Staff ALTER position DROP DEFAULT;
- ALTER TABLE Staff ALTER gender SET DEFAULT 'F';

Removing a table

- Removing a Table (DROP TABLE)
- 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)
- DROP TABLE TableName [RESTRICT | CASCADE]

Creating an Index

- An index is a structure that provides accelerated access to the rows of a table based on the values of one or more columns
- The presence of an index can significantly improve the performance of a query.
- However, as indexes may be updated by the system every time the underlying tables are updated, additional overheads may be incurred.
- Indexes are usually created to satisfy particular search criteria after the table has been in use for some time and has grown in size.

- CREATE [UNIQUE] INDEX IndexName ON TableName (columnName [ASC | DESC] [, . . .])

Creating an Index

- For the Staff and PropertyForRent tables, we may want to create at least the following indexes:
- CREATE UNIQUE INDEX StaffNolnd ON Staff (staffNo);
- CREATE UNIQUE INDEX PropertyNoInd ON PropertyForRent (propertyNo);

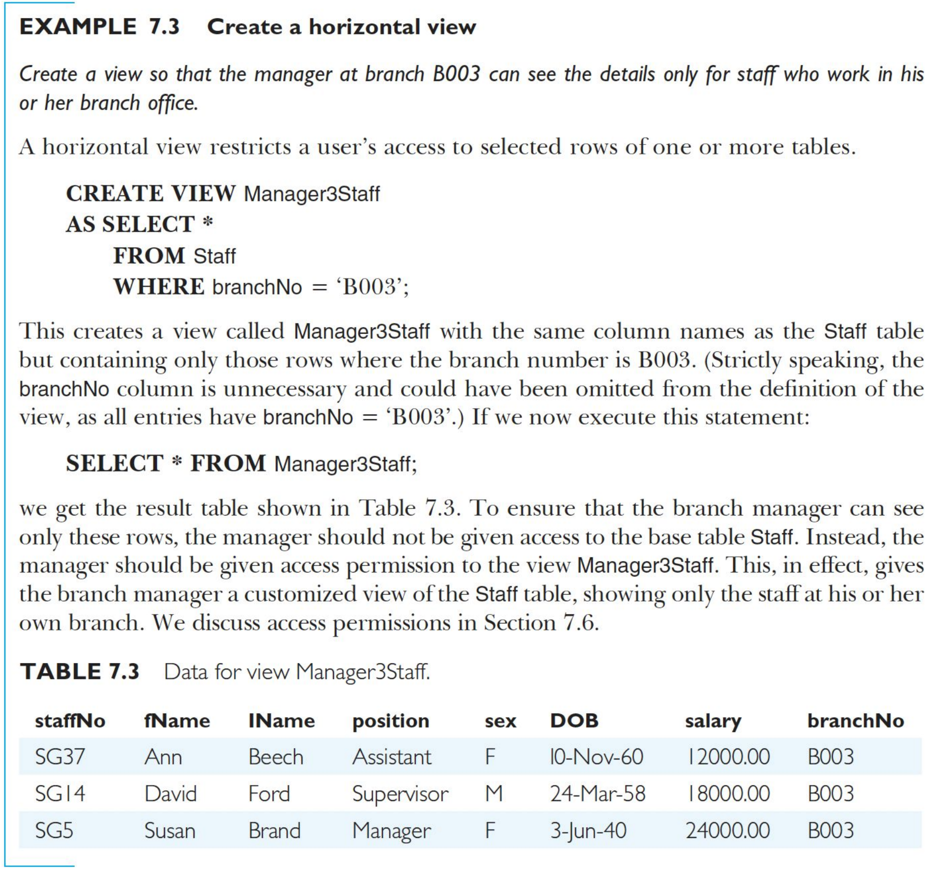
Views

- The dynamic result of one or more relational operations operating on the base relations to produce another relation.
- A view is a virtual relation that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.
- To the database user, a view appears just like a real table, with a set of named columns and rows of data.
- However, unlike a base table, a view does not necessarily exist in the database as a stored set of data values.
- Instead, a view is defined as a query on one or more base tables or views

Views

- *The format of the CREATE VIEW statement is:*
- *CREATE VIEW ViewName [(newColumnName [, . . .])] AS subselect [WITH [CASCADED | LOCAL] CHECK OPTION]*

Horizontal View



EXAMPLE 7.3 Create a horizontal view

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

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

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

This creates a view called Manager3Staff with the same column names as the Staff table but containing only those rows where the branch number is B003. (Strictly speaking, the branchNo column is unnecessary and could have been omitted from the definition of the view, as all entries have branchNo = ‘B003’.) If we now execute this statement:

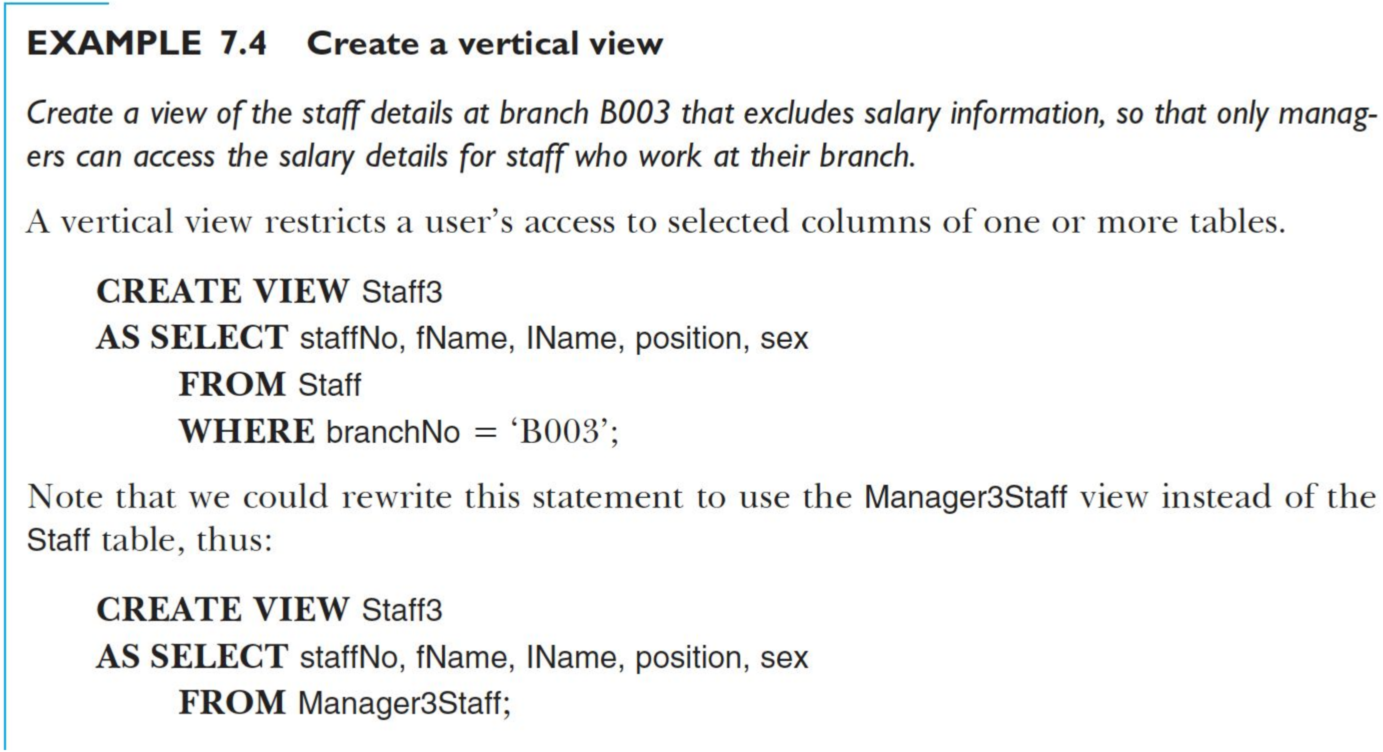
```
SELECT * FROM Manager3Staff;
```

we get the result table shown in Table 7.3. To ensure that the branch manager can see only these rows, the manager should not be given access to the base table Staff. Instead, the manager should be given access permission to the view Manager3Staff. This, in effect, gives the branch manager a customized view of the Staff table, showing only the staff at his or her own branch. We discuss access permissions in Section 7.6.

TABLE 7.3 Data for view Manager3Staff.

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

Vertical View



EXAMPLE 7.4 Create a vertical view

Create a view of the staff details at branch B003 that excludes salary information, so that only managers can access the salary details for staff who work at their branch.

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

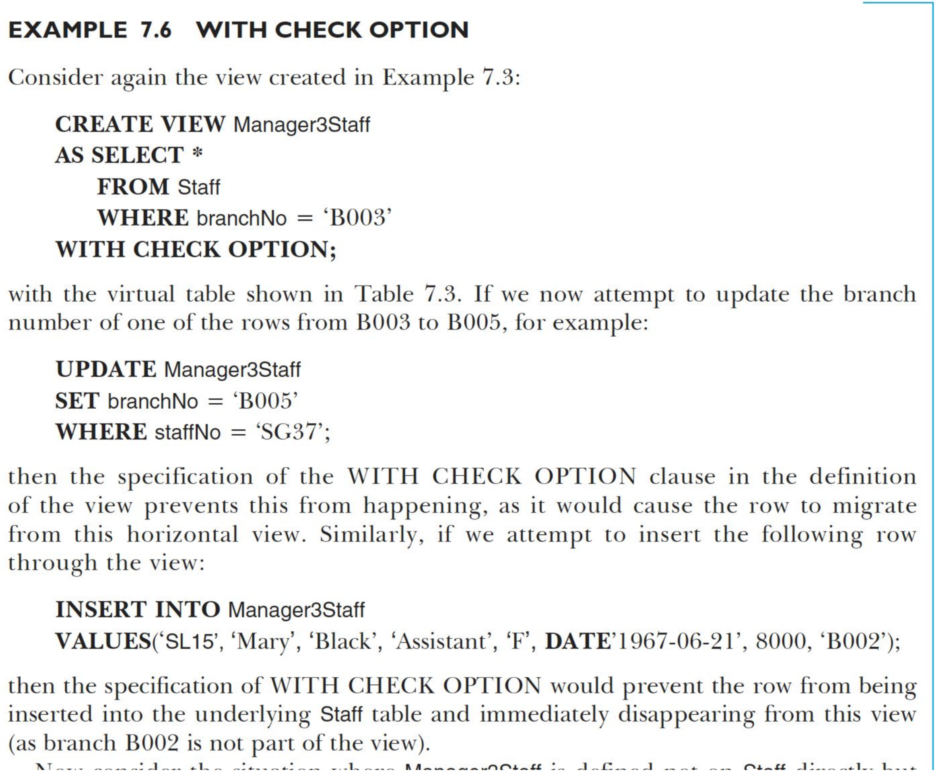
```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
   FROM Staff
   WHERE branchNo = 'B003';
```

Note that we could rewrite this statement to use the Manager3Staff view instead of the Staff table, thus:

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
   FROM Manager3Staff;
```

With Check Option

- Rows exist in a view, because they satisfy the WHERE condition of the defining query. If a row is altered such that it no longer satisfies this condition, then it will disappear from the view.
- Similarly, new rows will appear within the view when an insert or update on the view causes them to satisfy the WHERE condition.



EXAMPLE 7.6 WITH CHECK OPTION

Consider again the view created in Example 7.3:

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

with the virtual table shown in Table 7.3. If we now attempt to update the branch number of one of the rows from B003 to B005, for example:

```
UPDATE Manager3Staff
SET branchNo = 'B005'
WHERE staffNo = 'SG37';
```

then the specification of the WITH CHECK OPTION clause in the definition of the view prevents this from happening, as it would cause the row to migrate from this horizontal view. Similarly, if we attempt to insert the following row through the view:

```
INSERT INTO Manager3Staff
VALUES('SL15', 'Mary', 'Black', 'Assistant', 'F', DATE'1967-06-21', 8000, 'B002');
```

then the specification of WITH CHECK OPTION would prevent the row from being inserted into the underlying Staff table and immediately disappearing from this view (as branch B002 is not part of the view).

Now consider the situation where Manager2Staff is defined not on Staff directly but

Transactions

- The ISO standard defines a transaction model based on two SQL statements: COMMIT and ROLLBACK.
- A COMMIT statement ends the transaction successfully, making the database changes permanent. A new transaction starts after COMMIT with the next transaction-initiating statement.
- A ROLLBACK statement aborts the transaction, backing out any changes made by the transaction. A new transaction starts after ROLLBACK with the next transaction-initiating statement.
- Changes made by transactions are not visible to other concurrently executing transactions until transaction completes.

Properties of Transactions

- There are properties that all transactions should possess.
- The four basic, or so called **ACID**, properties that define a transaction are (Haerder and Reuter, 1983):
- **A** – **Atomicity** (All or Nothing)
- **C** – **Consistency** (Data Must Be Correct)
- **I** – **Isolation** (No Mixing Transactions)
- **D** – **Durability** (Changes Are Permanent)

- Imagine you are **transferring money** from Alice’s account to Bob’s account in a bank database.

🔹 A – Atomicity (All or Nothing)

- The transaction must be **fully completed** or **fully canceled**—no in-between states.
- If Alice's account is debited but Bob's isn't credited, the entire transaction is **rolled back**.
- 🔸 **Example:**

✅ Alice sends $100 to Bob → Success! ✅

❌ If the system crashes after deducting $100 from Alice but before adding it to Bob, it **undoes everything** (no partial transactions).

🔹 C – Consistency (Data Must Be Correct)

- The database **must stay in a valid state** before and after the transaction.
- If Alice had $500 before sending $100 to Bob, the total money in both accounts **should still be correct** after the transfer.
- 🔸 **Example:**

  ✅ Alice: **$500** → **$400**, Bob: **$200** → **$300** (Valid 👍)

  ❌ Alice: **$500** → **$400**, Bob: **$200** → **$350** (Invalid ❌ extra $50 appeared!)

🔹 I – Isolation (No Mixing Transactions)

- Multiple transactions happening at the same time **should not interfere** with each other.
- If Bob checks his balance while Alice’s transfer is still in progress, he should **see the correct final amount**, not an intermediate state.
- 🔸 **Example:**

  ✅ Alice transfers $100 while Bob withdraws $50 → Both work **independently** and correctly.

🔹 D – Durability (Changes Are Permanent)

- Once a transaction is committed, it **stays in the database** even if the system crashes.
- If Alice’s transfer is successful, the bank **won’t lose** the data even if the power goes out.
- 🔸 **Example:**

✅ System crashes? 💥 No worries! The transaction is safely **saved to disk**.

Granting Privileges to Other Users

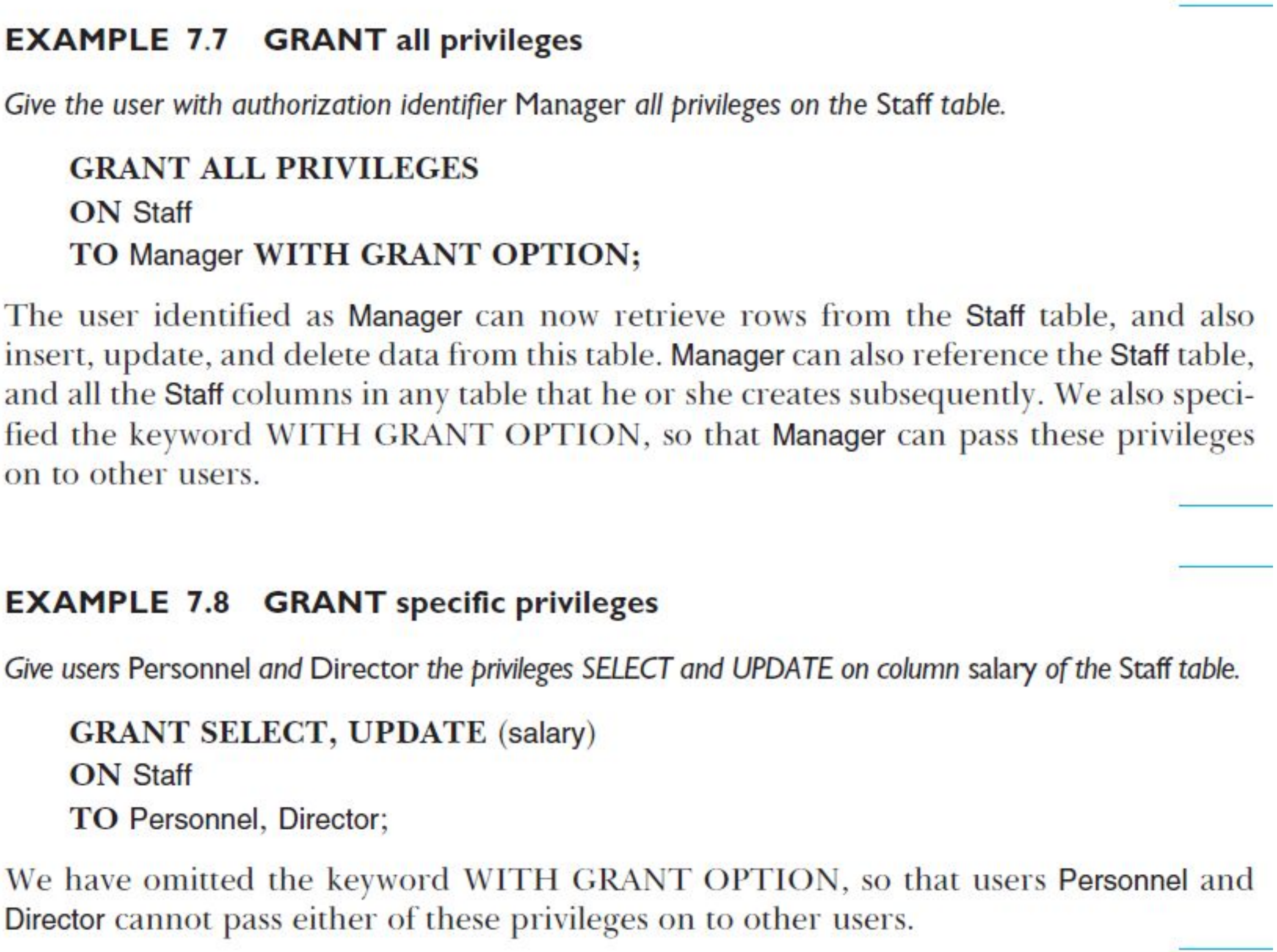
- The GRANT statement is used to grant privileges on database objects to specific users.

- Normally the GRANT statement is used by the owner of a table to give other users access to the data.

- The format of the GRANT statement is:

- **GRANT** {PrivilegeList | **ALL PRIVILEGES}** **ON** ObjectName **TO** {AuthorizationldList | **PUBLIC}**

  [WITH GRANT OPTION]



EXAMPLE 7.7 GRANT all privileges

Give the user with authorization identifier Manager *all privileges on the* Staff *table.*

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

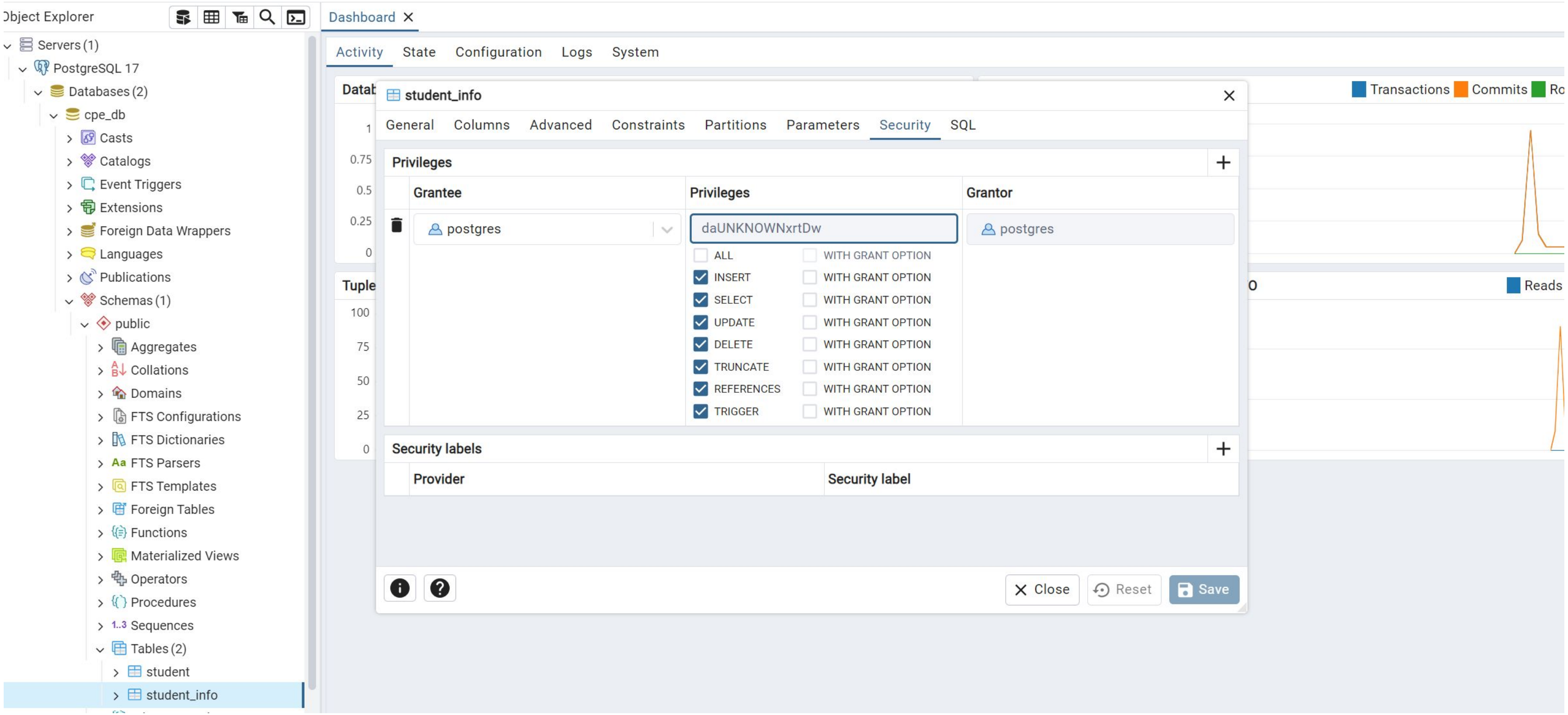
The user identified as Manager can now retrieve rows from the Staff table, and also insert, update, and delete data from this table. Manager can also reference the Staff table, and all the Staff columns in any table that he or she creates subsequently. We also specified the keyword WITH GRANT OPTION, so that Manager can pass these privileges on to other users.

EXAMPLE 7.8 GRANT specific privileges

Give users Personnel *and* Director *the privileges SELECT and UPDATE on column* salary *of the* Staff *table.*

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

We have omitted the keyword WITH GRANT OPTION, so that users Personnel and Director cannot pass either of these privileges on to other users.



Privileges

Grantee	Privileges	Grantor
postgres	daUNKNOWNxrtDw	postgres

☐ ALL ☐ WITH GRANT OPTION
☑ INSERT ☐ WITH GRANT OPTION
☑ SELECT ☐ WITH GRANT OPTION
☑ UPDATE ☐ WITH GRANT OPTION
☑ DELETE ☐ WITH GRANT OPTION
☑ TRUNCATE ☐ WITH GRANT OPTION
☑ REFERENCES ☐ WITH GRANT OPTION
☑ TRIGGER ☐ WITH GRANT OPTION

Security labels

Provider	Security label

Thank You

ayehninnkhine@parami.edu.mm

PARAMI
UNIVERSITY