Next....

- A bit more SQL...today
- Technical writing status check
- Quick look at Physical Design of DBMS
  - File organization: How is data actually stored
  - Transaction processing and concurrency
  - Recovering from failure
- Information retrieval
  - Document search and search engines
- Quick peek at data analytics
  - Data mining
- Quick look at NoSQL data models

A bit more SQL
- Views
- Security
- Triggers & assertions...
- First: status check on technical writing....

Are you reading up on technical writing
- Important reading resource(s):
  1. Michigan State Engineering
     - Chapter 2 and Chapter 4 in particular are relevant
  2. MIT: sentence structure
  3. Georgia Tech ‘cheatsheet’
Writing Homeworks 3, 4

Writing HW3: Your ‘product’ design from Phase 1
- Purpose: inform
- Include technical details: Schema design, normal form, etc.

Writing HW4: Your user guide for complete product
- Purpose: to instruct
- Like a user manual; but also include some technical details

These two will form a big part of your writing HW grade
- Writing 1, 2 together will form 100 points
  Writing 1 is only worth 20.

Creating Views

- Views are created using the `CREATE VIEW` command:

  `CREATE VIEW` viewName [(col1, col2, ..., colN)] AS selectStatement [WITH CHECK OPTION]

- Notes:
  - Select statement can be any SQL query and is called the `defining query` of the view.
  - It is possible to rename the columns when defining the view, otherwise they default to the names produced by the query.
  - `WITH CHECK OPTION` is used to ensure that if updates are performed through the view, the updated rows must satisfy the `WHERE` clause of the query.

Defining Views

Views are relations, except that they are not physically stored.

For presenting different information to different users

`Employee` (ssn, name, department, project, salary)

Payroll has access to `Employee`, others only to `Developers`
Views Example

- Create a view that has only the employees of department 'D2':

  ```sql
  CREATE VIEW empD2
  AS SELECT * FROM emp WHERE dno = 'D2';
  ```

- Create a view that only shows the employee number, title, and name:

  ```sql
  CREATE VIEW staff (Number,Name,Title)
  AS SELECT eno,ename,title FROM emp;
  ```

  - The first example is a horizontal view because it only contains a subset of the rows.
  - The second example is a vertical view because it only contains a subset of the columns.

Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow at runtime
  - Always up to date

- Materialized views
  - Used in data warehouses
  - Pre-computed offline – fast at runtime
  - May have stale data

Views: Example

- Create view of all branch names and IDs of customers with loan or deposit

  ```sql
  CREATE VIEW all-customer AS (SELECT branchname, CustID FROM Deposit)
  UNION (SELECT branchname, CustID FROM Loan);
  ```

Example

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

```sql
CREATE VIEW Seattle-Purchase AS
SELECT y.buyer, y.seller, y.product, y.store
FROM Person x, Purchase y
WHERE x.city = 'Seattle' AND x.name = y.buyer
```

"virtual table"
Seattle-Purchase(buyer, seller, product, store)
Querying a View

We can later use the view:

```sql
SELECT v.name, u.store
FROM Seattle-Purchase u, Product v
WHERE u.product = v.name AND
  v.category = 'shoes'
```

Removing Views

- Views are removed using the `DROP VIEW` command:

  ```sql
  DROP VIEW viewName [RESTRICT|CASCADE]
  ```

- Notes:
  - `RESTRICT` will not delete a view if other views are dependent on it.
  - `CASCADE` deletes the view and all dependent views.

View Resolution

- When a query uses a view, the process of **view resolution** is performed to translate the query into a query over only the base relations.

  ```sql
  CREATE VIEW staff (Number,Name,Job)
  AS SELECT eno,ename,title FROM emp WHERE dno = 'D2';
  ```

  ```sql
  SELECT Number, Name FROM staff
  WHERE job = 'EE' ORDER BY Number;
  ```

  Step #1: Replace column names in `SELECT` clause with names in defining query.

  ```sql
  SELECT eno, ename FROM staff
  WHERE job = 'EE' ORDER BY Number;
  ```

View Resolution (2)

- Step #2: View names in `FROM` clause replaced by `FROM` clause in defining query.

  ```sql
  SELECT eno, ename FROM emp
  WHERE job = 'EE' ORDER BY Number;
  ```

- Step #3: `WHERE` clause of user and defining query are combined. Replace column view names with names in defining query.

  ```sql
  SELECT eno, ename FROM emp
  WHERE title = 'EE' AND dno = 'D2' ORDER BY Number;
  ```
**View Resolution (3)**

Step #4: GROUP BY and HAVING clause copied to user query from defining query. (no change in example)

SELECT eno, ename FROM emp
WHERE title = 'EE' AND dno = 'D2' ORDER BY Number;

Step #5: Rename fields in ORDER BY clause.

SELECT eno, ename FROM emp
WHERE title = 'EE' AND dno = 'D2' ORDER BY eno;

---

**View Limitations**

- 1) If a view contains an aggregate function, it can only be used in SELECT and ORDER BY clauses of queries.

- 2) A view whose defining query has a GROUP BY cannot be joined with another table or view.

---

**Example 2: What Happens When We Query a View?**

```
SELECT v.name, u.store
FROM Seattle-Purchase u, Product v
WHERE u.product = v.name AND v.category = 'shoes'
```

```
SELECT v.name, u.store
FROM Seattle-Purchase u, Product v
WHERE u.product = v.name AND v.category = 'shoes'
```

---

**Updating Views: Part 1**

Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

CREATE VIEW Expensive-Product AS
SELECT name, maker
FROM Product
WHERE price > 100

INSERT INTO Expensive-Product
VALUES('Gizmo', 'Gadgets INC.')
INSERT INTO Product
VALUES('Gizmo', 'Gadgets INC.', NULL, NULL)
CREATE VIEW Toy-Product AS
SELECT price, maker
FROM Product
WHERE category = 'Toys'

INSERT INTO Toy-Product
VALUES('Gadgets INC.', $100)

INSERT INTO Product
VALUES(NULL, 'Gadgets INC.', 100, NULL)

UPDATE Views: Part 3

CREATE VIEW Buyer-Maker AS
SELECT x.buyer, y.maker
FROM Purchase x, Product y
WHERE x.product = y.name

INSERT INTO Buyer-Maker
VALUES('John Smith', 'Gadgets INC.')

View Updatability

- When a base table is updated, all views defined over that base table are automatically updated. When an update is performed on the view, it is possible to update the underlying table.

- A view is only updatable if:
  - does not use DISTINCT
  - every element in SELECT is a column name (no derived fields)
  - contains all fields of base relation that are non-NUL
  - FROM clause contains only one table
  - WHERE does not contain any nested selects
  - no GROUP BY or HAVING in defining query

- In practice, a view is only updatable if it uses only a single table, and any update through the view must not violate any of the integrity constraints of the table.

Updating Views

- Non-updateable views
  - Multiple relation views
  - Primary key NULL
  - Later versions of SQL allow some updates on multiple relation views
  - Security/privacy and bad use of views?
  - Interesting “side-effects” on view authorization
    - More when we discuss security/authorization
Advantages and Disadvantages of Views

Advantages:
- Data independence - allows base relations to change without affecting users.
- Security - views can be used to limit access to certain data to certain users.
- Easier querying - using views can often reduce the complexity of some queries.
- Convenience/Customization - users only see the parts of the database that they have interest and access to.

Disadvantages:
- Updatable views are not always supported and are restrictive.
- Performance - views add increased overhead: during query parsing and especially if they are materialized.

Database Security

Databases provide
- Shared access
- Controlled access
- Data consistency
- Data integrity
- Minimal redundancy

DB Security Requirements
- Secrecy: Users should not be able to see things they are not supposed to.
  - E.g., A student can’t see other students’ grades.
- Integrity: Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- Availability: authorized users should be able to see and modify things they are allowed to
- Non-repudiation:
  - DB cannot deny getting requests for changes
- Auditability
  - Helps determine if inappropriate disclosure has occurred & track divulged information to prevent inference
  - Difficult to record access of fields; often a field is reported to have been accessed when it has not – e.g. SELECT all entries with ZIP 20007
DB Security
- To achieve the objectives, a security policy should be developed
  - Determine data to be protected, user access to data
- Security mechanisms to enforce the policy
  - Note: when DB is a backend of an application, also need to be able to authenticate users
    Logging in to Amazon through a network
    Assure users they are communicating with amazon and not a malicious server masquerading as amazon
    – Have you seen this?

Means of Achieving Security Requirements
- Access Control
  - Inference is a problem (fields are related; knowledge of race or gender can be a good predictor of salary, for example)
  - Size and granularity different from access control in OS
- User Authentication
  - DBMS does its own authentication because no trusted path between DBMS and OS when DBMS is an application program on top of OS

Access Controls
- A security policy specifies who is authorized to do what.
- A security mechanism allows us to enforce a chosen security policy.
- Two main mechanisms at the DBMS level:
  - Discretionary access control
  - Mandatory access control

Discretionary Access Control
- Based on the concept of access rights or privileges for objects and mechanisms for giving users privileges (and revoking privileges).
  - In Databases, objects refer to Tables and Views
  - Creator of a table or a view automatically gets all privileges on it.
  - DMBS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.
SQL Security

- Security in SQL is based on **authorization identifiers**, **ownership**, and **privileges**.

- An authorization identifier (or user id) is associated with each user. Normally, a password is also associated with an authorization identifier. Every SQL statement performed by the DBMS is executed on behalf of some user.

- The authorization identifier is used to determine which database objects the user has access to.

- Whenever a user creates an object, the user is the owner of the object and initially is the only one with the ability to access it.

SQL Privileges

- **Privileges** give users the right to perform operations on database objects. The set of privileges are:
  - SELECT - the user can retrieve data from table
  - INSERT - the user can insert data into table
  - UPDATE - the user can modify data in the table
  - DELETE - the user can delete data (rows) from the table
  - REFERENCES - the ability to reference columns of a named table in integrity constraints
  - USAGE - the ability to use domains, character sets, and translations (i.e. other database objects besides tables)

- **Notes:**
  - INSERT and UPDATE can be restricted to certain columns.
  - When a user creates a table, they become the owner and have full privileges on the table.

SQL GRANT Command

- The **GRANT** command is used to give privileges on database objects to users.

```
GRANT [privilegeList | ALL [PRIVILEGES]]
  ON ObjectName
TO [AuthorizationIdList | PUBLIC]
  [WITH GRANT OPTION]
```

- The privilege list is one or more of the following privileges:

  - SELECT ([columnName [,...]])
  - DELETE
  - INSERT ([columnName [,...]])
  - UPDATE ([columnName [,...]])
  - REFERENCES ([columnName [,...]])
  - USAGE

- The following **privileges** can be specified:
  - SELECT: Can read all columns (including those added later via ALTER TABLE command).
  - INSERT(col-name): Can insert tuples with non-null or non-default values in this column.
  - INSERT means same right with respect to all columns.
  - DELETE: Can delete tuples.
  - REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.

- If a user has a privilege with the **GRANT OPTION**, can pass privilege on to other users (with or without passing on the **GRANT OPTION**).

- Only owner can execute CREATE, ALTER, and DROP.
**GRANT Examples**

Allow all users to query the Dept relation:

```
GRANT SELECT ON dept TO PUBLIC;
```

Only allow users Manager and Director to access and change Salary in Emp:

```
GRANT SELECT, UPDATE(salary) ON Emp TO Manager,Director;
```

Allow the Director full access to Proj and the ability to grant privileges to other users:

```
GRANT ALL PRIVILEGES ON Proj TO Director WITH GRANT OPTION;
```

**GRANT and REVOKE of Privileges**

- **GRANT INSERT, SELECT ON** Sailors **TO Trevor**
  - Trevor can query Sailors or insert tuples into it.
- **GRANT DELETE ON** Sailors **TO Kunal WITH GRANT OPTION**
  - Kunal can delete tuples, and also authorize others to do so.
- **GRANT UPDATE** *(rating)* **ON** Sailors **TO Sarah**
  - Sarah can update (only) the rating field of Sailors tuples.
- **GRANT SELECT ON** ActiveSailors **TO Guppy, Yuppy**
  - This does NOT allow the 'uppies to query Sailors directly!
  - ActiveSailors is a View over Sailors and Reserves tables
- **REVOKE:** When a privilege is revoked from X, it is also revoked from all users who got it solely from X.

**SQL REVOKE Command**

- The `REVOKE` command is used to remove privileges on database objects from users.

```
REVOKE [GRANT OPTION FOR] {privilegeList | ALL [PRIVILEGES]} ON ObjectName FROM {AuthorizationIdList | PUBLIC} {RESTRICT|CASCADE}
```

- **Notes:**
  - `ALL PRIVILEGES` removes all privileges on object.
  - `GRANT OPTION FOR` removes the ability for users to pass privileges on to other users (not the privileges themselves).
  - `RESTRICT - REVOKE` fails if privilege has been passed to other users.
  - `CASCADE` - removes any privileges and objects created using the revoked privileges including those passed on to others.

**SQL REVOKE Command (2)**

- Privileges are granted to an object to a user from a specific user. If a user then revokes their granting of privileges, that only applies to that user.

- **Example:**
  - User A grants all privileges to user B on table T.
  - User B grants all privileges to user C on table T.
  - User E grants `SELECT` privilege to user C on table T.
  - User C grants all privileges to user D on table T.
  - User A revokes all privileges on table T from B (using `CASCADE`).
  - This causes all privileges to be removed for user C as well, except the `SELECT` privilege which was granted by user E.
  - User D now has only the `SELECT` privilege as well.
REVOKE Examples

Allow no users to query the `dept` relation:

```sql
REVOKE SELECT ON dept FROM PUBLIC;
```

Remove all privileges from user `Joe` on `Emp` table:

```sql
REVOKE ALL PRIVILEGES ON Emp FROM Joe;
```

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the bid’s of boats that have been reserved.
  - Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
  - Together with GRANT/REVOKE commands, views are a very powerful access control tool.
  - How about GUI based views
    - Take the user to a different GUI based on their role?

SQL Security and Views

- Views are used to provide security and access restriction to certain database objects.

- Example: Consider the `Emp` relation. We want the user `Staff` to have only query access but not be able to see users birthdate and salary. How do we accomplish this?
  - Step #1: Create a view on the `Emp` relation.

```sql
CREATE VIEW EmpView AS
SELECT eno, ename, title, supereno, dno FROM emp;
```

SQL Security and Views (2)

- Step #2: Provide `SELECT` privilege to `Staff`.

```sql
GRANT SELECT ON EmpView TO Staff;
```

- Step #3: REVOKE privileges on base relation (`Emp`)

```sql
REVOKE SELECT ON Emp FROM Staff;
```
GRANT/REVOKE on Views
- If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped!
- If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well; so do users who were granted that privilege on the view!

Role-Based Authorization
- In SQL-92, privileges are actually assigned to authorization ids, which can denote a single user or a group of users.
  - CREATE ROLE and DROP ROLE
  - GRANT role and REVOKE role
- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.

Security to the Level of a Field!
- Can create a view that only returns one field of one tuple. (How?)
  - Then grant access to that view accordingly.
- Allows for arbitrary granularity of control, but:
  - Clumsy to specify
  - Performance is unacceptable (Too many view creations and lookups.)

Problem?
- Discretionary control has some flaws, e.g., the Trojan horse problem:
  - Kyle creates Horsie and gives INSERT privileges to Jen (who doesn’t know about this).
  - Kyle modifies the code of an application program, to update grades, used by Jen, to additionally write some secret data (Kyle’s grades) to table Horsie.
  - Now, Kyle can see the secret info.
- The modification of the code is beyond the DBMSs control, but it can try and prevent the use of the database as a channel for secret information.
Mandatory Access Control

- Based on system-wide policies that cannot be changed by individual users.
  - Each DB object is assigned a security class.
  - Each subject (user or user program) is assigned a clearance for a security class.
  - Rules based on security classes and clearances govern who can read/write which objects.
- Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

Intuition

- Idea is to ensure that information can never flow from a higher to a lower security level.
- E.g., If Narahari has security class C, Simha has class S, and the secret table has class S:
  - Narahari's table, Horsie, has Narahari's clearance, C.
  - Simha's application has his clearance, S.
  - So, the program cannot write into table Horsie.
- The mandatory access control rules are applied in addition to any discretionary controls that are in effect.

Internet-Oriented Security

- Key Issues: User authentication and trust.
  - When DB must be accessed from a secure location, password-based schemes are usually adequate.
  - For access over an external network, trust is hard to achieve.
    - If someone with Suraj’s credit card wants to buy from you, how can you be sure it is not someone who stole his card?
    - How can Suraj be sure that the screen for entering his credit card information is indeed yours, and not some rogue site spoofing you (to steal such information)? How can he be sure that sensitive information is not “sniffed” while it is being sent over the network to you?
- Encryption is a technique used to address these issues.