Last Lecture

- Covered SQL constraints
  - **NOT NULL** constraints
  - **CHECK** constraints
  - **PRIMARY KEY** constraints
  - **FOREIGN KEY** constraints
  - **UNIQUE** constraints

- Impact of **NULL** values on constraint enforcement
  - Specifically, **FOREIGN KEY** and **UNIQUE**...

- Automatic resolution of constraint violation
Constraint Names

- Can assign names to constraints
  - When constraint is violated, error indicates which constraint
  - Database usually assigns names to constraints if you don’t
  - Rules on constraint names vary

- Example:
  ```
  CREATE TABLE employee (...
    ...CONSTRAINT emp_pk PRIMARY KEY (emp_id),
    CONSTRAINT emp_ssn_ck UNIQUE (emp_ssn),
    CONSTRAINT emp_mgr_fk FOREIGN KEY (manager_id)
    REFERENCES employee
  ```

- Useful for referring to specific constraints
Temporary Constraint Violation

- Constraints take time to enforce
  - Can dramatically impact performance of large data-import operations

- Some operations may need to temporarily violate constraints
  - The operation is performed within a larger transaction (i.e. a batch of operations that should be treated as a unit)
  - During the transaction, constraints are temporarily violated
  - At end of transaction, constraint is restored

- Defer constraint enforcement to end of transaction
  - At end of transaction, all changes are checked against deferred constraints
Deferring Constraint Application

- Can mark constraints as deferrable
- In constraint declaration, specify:
  - `DEFERRABLE` constraints may be deferred to end of transaction
  - `NOT DEFERRABLE` constraints are always applied immediately
- For `DEFERRABLE` constraints:
  - `INITIALLY IMMEDIATE` is applied immediately by default
  - `INITIALLY DEFERRED` is applied at end of transaction by default
Temporarily Removing Constraints

- To defer constraints in current transaction:
  ```sql
  SET CONSTRAINTS c1, c2, ... DEFERRED;
  ```
  - Specified constraints must be deferrable

- Not all databases support deferred constraints
  - Only option is to temporarily remove and then reapply constraints
  - Will usually affect all users of database! Safest to ensure exclusive access for this.
  - Remove, then reapply constraints with `ALTER TABLE` syntax
SQL provides data types for dates and times

- **DATE**
  - A calendar date, including year, month, and day of month

- **TIME**
  - A time of day, including hour, minute, and second value
  - Doesn’t include fractional seconds

- **TIME (P)**
  - Just like **TIME**, but includes P digits of fractional seconds
  - Typically, $P = [0, 6]$
Date and Time Values (2)

- Can include timezone info as well:
  - `TIME WITH TIMEZONE`
  - `TIME(P) WITH TIMEZONE`

- `TIMESTAMP`
  - A combination of date and time values
  - Includes fractional seconds by default
  - Can also specify `TIMESTAMP(P)`
  - `P = 6` by default

- Timestamps can also include time zone info
  - `TIMESTAMP WITH TIMEZONE`
  - `TIMESTAMP(P) WITH TIMEZONE`
Often a variety of other non-standard types
- **DATETIME** – Like **TIMESTAMP** but \( P = 0 \) by default
- **YEAR** – Just a 4-digit year value
- Nonstandard = not portable
Microsoft SQL Server Date Types

- SQL Server 2005 and earlier provide very different date/time support
  - **DATETIME** – more like standard **TIMESTAMP** type
    - Represents both date and time
    - Jan 1, 1753 – Dec 31, 9999; precision of 3.33ms (…?)
  - **SMALLDATETIME**
    - Jan 1, 1900 – Jun 6, 2079; precision of 1 minute
    - No ability to represent only a date, or only a time!
- SQL Server 2008 adds more standard-like support
  - **DATE, TIME, DATETIME2** – similar to standard types
  - **DATETIMEOFFSET** – date/time value plus timezone
Date and Time Formats

- Date and time values follow specific formats
  - Enclosed in single-quotes

- Examples: MER-A “Spirit” launch time
  - Timestamp value (UT; +0):
    '2003-06-10 17:58:46.773'
  - Date value: '2003-06-10'
  - Time value: '17:58:47'

- Can have invalid date/time values:
  - Invalid time: '25:14:68'
  - Invalid date: '2001-02-31'
  - Some DBMSes can allow partial/invalid dates and times, if required by an application
Most DBMSes support many date/time formats.

Most widely supported is ISO-8601 date/time format.

ISO-8601 format:
'2003-06-10 17:58:46.773'
- year-month-day hour:minutes:seconds.milliseconds
- Sometimes date and time are separated by “T” character
- Time is in 24-hour time format
- Optional timezone specification at end

Other formats:
'June 10, 2003 5:58:46 PM'
'10-Jun-2003 17:58:46.773'

Most databases can parse all of these.
“Current Time” Values

- Several functions provide current date and time values
  - `CURRENT_DATE()`
  - `CURRENT_TIME()`
  - `CURRENT_TIMESTAMP()`
    - Include time zone information
  - `LOCALTIME()`
  - `LOCALTIMESTAMP()`
    - Don’t include time zone information
- Usually many other functions too, e.g. `NOW()`
  - Nonstandard, but widely supported
Components of Dates and Times

- Date and time values are not atomic
  - Not really allowed in the Relational Model...
  - (In reality, many SQL types are not atomic)

- SQL provides a function to extract components of dates and times
  - `EXTRACT (field FROM value)`
  - Can specify:
    - `YEAR, MONTH, DAY, HOUR, MINUTE, SECOND`
    - `TIMZEONE_HOUR, TIMEZONE_MINUTE`
  - Many other (nonstandard but common) options too
    - week of year, day of year, day of week, quarter, century, …
Example Date Operation

- Sales records:

  ```sql
  CREATE TABLE salesrecords (
    sale_id INTEGER PRIMARY KEY,
    cust_id INTEGER NOT NULL,
    sale_time TIMESTAMP NOT NULL,
    sales_total NUMERIC(8, 2) NOT NULL,
    ... 
  );
  ```

- Compute monthly sales totals:
  - Start by finding month of each sale
    ```sql
    SELECT sale_id,
    EXTRACT (MONTH FROM sale_time) AS sale_month
    FROM salesrecords;
    ```
  - Build larger query using this information
Time Intervals

- INTERVAL
  - Data type for time intervals
  - Supports operations on dates and times
  - Also supports a precision: `INTERVAL (P)`
- If \( x \) and \( y \) are date values:
  - \( x - y \) produces an INTERVAL
- If \( i \) is an INTERVAL value:
  - \( x + i \) or \( x - i \) produces a date value
- Can use INTERVAL to specify fixed intervals
  - `INTERVAL 1 WEEK`
  - `INTERVAL '1 WEEK'`
Example Date Schema

- Event database schema:

```sql
CREATE TABLE event (
    event_id INTEGER PRIMARY KEY,  
    event_type VARCHAR(20) NOT NULL,  
    event_date DATE NOT NULL,  
    event_desc VARCHAR(200)
);
```

- To generate notices of upcoming events:

```sql
SELECT * FROM event
WHERE event_date >= CURRENT_DATE() AND event_date <= (CURRENT_DATE() + INTERVAL 1 WEEK);
```
Example Date Schema (2)

- Can rewrite to use `BETWEEN` syntax:
  ```sql
  SELECT * FROM event
  WHERE event_date BETWEEN CURRENT_DATE() AND (CURRENT_DATE() + INTERVAL 1 WEEK);
  ```

- Current date/time functions are evaluated only once during a query! 😊
  - e.g. query will see one value for `CURRENT_TIME()` even if it runs for an extended period of time
“Large Object” Types

- SQL `CHAR(N)` and `VARCHAR(N)` types have limited sizes
  - For `CHAR`, usually $N < 256$
  - For `VARCHAR`, usually $N < 65536$

- `BLOB` and `CLOB` types support larger data sizes
  - “LOB” = Large Object
  - Useful for storing images, documents, etc.
  - Support varies widely across DBMSes

- `TEXT` is also rather common
  - Large text fields, e.g. MB or GB of text data
Example Schema

- Schema for storing book reviews:

  ```sql
  CREATE TABLE bookreview (
    review_id INT PRIMARY KEY,
    book_title VARCHAR(50) NOT NULL,
    book_image BLOB,
    reviewer VARCHAR(30) NOT NULL,
    pub_time TIMESTAMP NOT NULL,
    review_text CLOB NOT NULL,
    UNIQUE (book_title, reviewer)
  );
  ```

- Review text can be large
- Can also include a book image, if desired
General support for “large object” types is usually focused on smaller objects
- No larger than a few 10s of KBs
- A few MBs is definitely pushing it

Most expensive part is moving large objects into and out of database
- For simple, general purpose DBMSes, can involve constructing large SQL statements with escaped data

Databases also don’t store this information very efficiently
For objects larger than ~100 KB, should definitely use the filesystem
- That’s what it’s designed for!
- Store *filesystem paths* in the database instead

For smaller objects that are frequently retrieved, storing on filesystem can take load off database
- e.g. user icons for a social networking website
- Let webserver serve them directly from the filesystem — again, it knows how to do that kind of thing more quickly

Some DBMSes have specialized support for storing and manipulating very large objects
- Just don’t expect your application to be easily portable…
Default Values

- Can specify default values for columns
  - `colname type DEFAULT expr`
- Can specify an actual value
  - `book_rating INT DEFAULT 3`
- Can specify an expression
  - `pub_time TIMESTAMP DEFAULT NOW()`
- If unspecified, default value is `NULL`
- Affects `INSERT` statements
  - Columns with default values don’t have to be specified
  - Columns without a default value *must* be specified at insert-time!
Many databases offer special support for integer primary keys
- DB will generate unique values for use as primary keys

Examples:
- PostgreSQL and MySQL:
  ```sql
  CREATE TABLE employee (
    emp_id SERIAL PRIMARY KEY,
    ...
  )
  ```
- Microsoft SQLServer:
  ```sql
  CREATE TABLE employee (
    emp_id INT IDENTITY PRIMARY KEY,
    ...
  )
  ```
Updated Book Review Schema

CREATE TABLE bookreview (
    review_id    SERIAL      PRIMARY KEY,
    book_title   VARCHAR(50) NOT NULL,
    book_image   BLOB,
    reviewer     VARCHAR(30) NOT NULL,
    pub_time     TIMESTAMP   NOT NULL DEFAULT NOW(),
    book_rating  INT         NOT NULL DEFAULT 3,
    review_text  CLOB        NOT NULL,
    UNIQUE (book_title, reviewer)
);

- Every new review gets a unique ID value
- Publication time is set to current time when review is added to database
- Default book rating is 3 out of 5
Altering Table Schemas

- SQL ALTER TABLE command allows schema changes
- Wide variety of operations
  - Rename a table
  - Add and remove constraints
  - Add and remove table columns
  - Change the type of a column
  - Change default values for columns
- Very useful for migrating schema to new version
  - Migration process must be carefully designed...
- Again, support varies across DBMSes
Example Alterations

- Rename the `bookreview` table:
  ```sql
  ALTER TABLE bookreview
  RENAME TO item_review;
  ```

- Remove the book image column:
  ```sql
  ALTER TABLE bookreview
  DROP COLUMN book_image;
  ```

- Add a constraint to the `bookreview` table:
  ```sql
  ALTER TABLE bookreview
  ADD CHECK (book_rating BETWEEN 1 AND 5);
  ```
Can drop columns from tables
- What if the column is a key?
- What if the column is referenced by a view?
- Can often specify `CASCADE` to delete dependent objects, if desired

Newly added columns must have a default value
- Existing rows in database get default value for new column

Changing table schema can be very expensive
- Some operations can require scanning or rewriting the entire table
  - Some DBs do this for all schema-alteration commands, e.g. MySQL
- e.g. adding a new constraint requires a table scan
Temporary Tables

- Sometimes want to generate and store relations temporarily
  - Complex operations implemented as multiple queries
  - This is relational algebra assignment operation: \( \leftarrow \)
- SQL provides temporary tables for these cases
  - Table’s contents are associated with client’s session
  - Clients can’t access each others’ temp table data
- SQL standard specifies global temporary tables
  - Temporary table has a global name and schema
  - Only the contents of the temporary table are per-client
  - When client disconnects, their temporary data is purged
Many databases also provide local temporary tables
- Table’s schema is also local to client session
- When client disconnects, the table is dropped
- Different clients can use same table name with different schemas

Client can manually purge data from temp tables when needed
- In case of local temp tables, can also drop them anytime during session
Simple variation of `CREATE TABLE` syntax

- Add `TEMPORARY` (or `GLOBAL TEMPORARY`) to command

Example:

- Make a temporary table to store counts of sales grouped by month
  ```sql
  CREATE TEMPORARY TABLE salesbymonth ( 
    sale_month INT NOT NULL, 
    num_sales INT NOT NULL 
  );
  ```
Temporary Table Example

- Can populate temp table with computed values
  
  ```sql
  INSERT INTO salesbymonth
  SELECT EXTRACT (MONTH FROM sale_time) AS mon,
         COUNT(*)
  FROM salesrecords GROUP BY mon;
  ```

  - Only need to perform computations once
  
  - Can improve efficiency of large or multi-step operations
  
  - Temporary results are cleaned up at end of session

- Issue queries against temporary table and use results
  
  ```sql
  SELECT sale_month, num_sales, promotion_desc
  FROM salesbymonth
  JOIN promotions USING (sale_month);
  ```
Temporary Table Contents

- When to flush temporary table contents?
- Two main options:
  - At end of current transaction
  - When entire client session ends
- Can specify behavior with ON COMMIT clause at end of table declaration
  - To flush temp table at end of each transaction: 
    ON COMMIT DELETE ROWS
  - To flush temp table at end of session: 
    ON COMMIT PRESERVE ROWS
- SQL standard specifies default is DELETE ROWS!
  - Not all DBMSes follow this, but some do!
Example ON COMMIT Clauses

- To flush rows after each transaction:
  ```sql
  CREATE TEMPORARY TABLE salesbymonth (  
    sale_id    INT NOT NULL,  
    sale_month INT NOT NULL  
  ) ON COMMIT DELETE ROWS;
  ```

- To keep rows until end of session:
  ```sql
  CREATE TEMPORARY TABLE salesbymonth (  
    sale_id    INT NOT NULL,  
    sale_month INT NOT NULL  
  ) ON COMMIT PRESERVE ROWS;
  ```
Using Temporary Tables

- Temporary tables can dramatically improve performance of certain queries

- **Approach:**
  - Create temporary table to store useful but costly intermediate results
    - Don’t use many (or any) constraints – want to be fast!
  - Populate temporary table via `INSERT ... SELECT` statement
  - Use temporary table to compute other results
  - Temporary table goes away automatically, at end of transaction, or at end of session
Alternate Temp-Table Syntaxes

- Databases frequently support alternate syntaxes for creating and populating temporary tables
  - Simplify the common case!
- One common syntax (e.g. MySQL, Postgres, Oracle):
  ```sql
  CREATE TEMPORARY TABLE tblname AS 
  select_stmt;
  ```
- Another common syntax (e.g. Postgres, SQLServer):
  ```sql
  SELECT ... INTO TEMPORARY TABLE ...;
  ```
- Both syntaxes can also create non-temporary tables
A query run on a MySQL server:

```sql
SELECT ident, total_a / total_b AS ratio
FROM (SELECT CONCAT(a1, a2) AS ident,
       SUM(val_a) AS total_a
       FROM t1 GROUP BY ident) AS result1,
    (SELECT CONCAT(a1, a2) AS ident,
       SUM(val_b) AS total_b
       FROM t2 GROUP BY ident) AS result2
WHERE result1.ident = result2.ident;
```

- Overall query takes ~15 mins to execute on fast server
- Inner queries complete in << 1 second by themselves
Real-World Example (2)

- **MySQL query:**

  ```sql
  SELECT ident, total_a / total_b AS ratio
  FROM (SELECT CONCAT(a1, a2) AS ident,
          SUM(val_a) AS total_a
          FROM t1 GROUP BY ident) AS result1,
          (SELECT CONCAT(a1, a2) AS ident,
           SUM(val_b) AS total_b
           FROM t2 GROUP BY ident) AS result2
  WHERE result1.ident = result2.ident;
  ```

- **Problem is that MySQL cannot efficiently join two derived results using a computed column**
  - A limitation of MySQL’s join processor 😞
Real-World Example (3)

- A solution:
  - First, create temporary tables to hold intermediate results
    ```sql
    CREATE TEMPORARY TABLE temp1 AS
    SELECT CONCAT(a1, a2) AS ident,
           SUM(val_a) AS total_a
    FROM t1 GROUP BY ident;
    ```
  - ...same with other inner query...
  - Second, create indexes on temporary tables
  - Finally, issue outer query against temporary tables

- Result:
  - Entire process, including create/drop temp tables, takes < 1 second (as opposed to ~15 minutes)