Database Schemas

oauth_connections table (unified)

```
-- Create ENUM type for PostgreSQL
CREATE TYPE oauth_status AS ENUM ('active', 'expired', 'revoked', 'error');

-- Table definition
CREATE TABLE oauth_connections (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR(255) NOT NULL,
  connector_id VARCHAR(50) NOT NULL, -- 'google', 'linkedin', etc.

TODO : connector_data (connector&user specific : email, id, stuff that is relate
d to the platform)

  -- Token data
  access_token TEXT NOT NULL,
  refresh_token TEXT,
  token_type VARCHAR(50) DEFAULT 'Bearer',
  expires_at TIMESTAMP,

  -- Scope & permissions
  scopes TEXT NOT NULL, -- JSON array or space-separated

  -- Status tracking
  status oauth_status DEFAULT 'active',

  -- Timestamps
  granted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  last_refresh_at TIMESTAMP NULL,
  last_used_at TIMESTAMP NULL,

  -- Connector-specific data
```

```
  connector_metadata JSON, -- Store account_id, email, profile_url, etc.

  -- Audit
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Note: auto-upd
ate on row change requires a trigger in PostgreSQL

  -- Constraints
  -- FOREIGN KEY (user_id) REFERENCES TODO(user_id) ON DELETE CASCA
DE,
  UNIQUE (user_id, connector_id)  -- previously UNIQUE KEY
);

-- Indexes (PostgreSQL requires CREATE INDEX outside table)
CREATE INDEX idx_user_connector ON oauth_connections(user_id, connector
_id);
CREATE INDEX idx_status ON oauth_connections(status);
CREATE INDEX idx_expires_at ON oauth_connections(expires_at);
CREATE INDEX idx_last_used ON oauth_connections(last_used_at);
```

Why Unified Schema?

1. **Scalability**: Adding new connectors doesn't require schema migrations
2. **Consistent querying**: Single place to fetch all user connections
3. **Easier maintenance**: One set of token refresh logic
4. **Better analytics**: Easy to query across all connectors
5. **Reduced complexity**: No need to maintain N different table structures

Usage in connector_metadata (JSON field)

```
{
  "google": {
    "email": "user@gmail.com",
```

```
    "account_id": "123456789"
  },
  "linkedin": {
    "profile_url": "<https://linkedin.com/in/>...",
    "member_id": "abc123"
  },
  "shopify": {
    "shop_domain": "mystore.myshopify.com",
    "shop_id": "12345"
  }
}
```

Key Design Decisions

1. **UNIQUE constraint** on `(user_id, connector_id)` - prevents duplicate connections
2. **JSON for connector_metadata** - flexible for connector-specific data without schema changes
3. **TEXT for tokens** - some OAuth tokens can be very long
4. **expires_at index** - critical for background refresh jobs
5. **CASCADE delete** - clean up connections when user is deleted

Token Refresh Strategy

With this schema, you can easily implement:

```
// Pseudo-code for token refresh
async function getValidToken(userId, connectorId) {
  const connection = await db.query(
    'SELECT * FROM oauth_connections WHERE user_id = ? AND connector_id
= ? AND status = "active"',
    [userId, connectorId]
  );

  if (connection.expires_at < new Date()) {
```

```
    // Refresh token
    const newTokens = await refreshOAuthToken(connection);
    await db.query(
      'UPDATE oauth_connections SET access_token = ?, expires_at = ?, last_ref
resh_at = NOW() WHERE id = ?',
      [newTokens.access_token, newTokens.expires_at, connection.id]
    );
    return newTokens.access_token;
  }

  await db.query(
    'UPDATE oauth_connections SET last_used_at = NOW() WHERE id = ?',
    [connection.id]
  );

  return connection.access_token;
}
```

This approach gives you flexibility, maintainability, and clear visibility into all OAuth connections across your platform.